In-Class Programming Task 9

Math 253: Statistical Computing & Machine Learning

Counting with Conditionals

This activity is about simulating complex situations where there are many different actions that can be taken depending on earlier results. In terms of programming, you'll be writing functions that use *conditionals* to make decisions.

Poker anyone?

These are the rules for a poker game called 5-card draw.

1. You are dealt 5 cards.
2. After examining your cards, you can select to replace up to three of them with new cards drawn from the deck.

The initial draw

Wikipedia (`https://en.wikipedia.org/wiki/Five-card_draw`) lists these probabilities for the initial draw of five cards.

- Royal flush: <0.001% five cards of the same suit with ranks 10, ack, queen, king, ace. This could also be called a royal straight flush, but a royal flush is by necessity a straight flush.
- Straight flush (not including royal flush): <0.002% five cards of the same suit in consecutive order
- Four of a kind 0.02% four cards of the same value (and, necessarily, different suits)
- Full house: 0.14% Three of a kind and the other two cards forming a pair
- Flush (excluding royal flush and straight flush): 0.20%
  - five cards of the same suit
- Straight (excluding royal flush and straight flush): 0.39%
  - five consecutively numbered cards
- Three of a kind: 2.11%
- Two pair: 4.75%
- One pair: 42.30%
- No pair: 50.10%

The deck of cards

You're going to be dealing cards from a deck. To do this, you need to have a good computer representation of a deck.

As you know, a poker deck consists of 52 cards. Each card has

- a *rank*: 2, 3, 4, . . . , 10, J, Q, K, A
- a *suit*: clubs, diamonds, hearts, spades

In designing a computer representation, think about the kinds of tasks that you will need to perform. For example, you'll want to be able to determine whether you have all cards of the same suit, or all cards in order.

A reasonable representation is to give each card a number. The first two digits represent the rank (with 11 for J, 12 for Q, and so on). The last digit represents the suit: 1, 2, 3, 4. For instance, the card 123 is the queen of hearts, since 12 in the first two digits stands for Q and 3 in the last digit stands for hearts.

Here's how to make such a deck:

```
poker_deck <- c(outer((10 * 2:14), 1:4, "+"))
```

The `outer()` function takes two vectors and an operation. The operation is applied to every possible pair of an element from the first vector and an element from the second vector.

Calculating suits and ranks

Write two functions, `suits()` and `ranks()` that take a set of cards and return respectively the suits and ranks of those cards. The calculations are very simple when performed on the card representation given above. Just as a hint, each of the two calculations can be done with simple arithmetic. One involves the `%%` operation and the other the `%/%` operation. To see what's going on, try `123 %% 10` and then `123 %/% 10`, one of which should return 12 and the other 3.

What kind of hand?

Write these functions to evaluate a 5-card hand:

1. `is_royal_flush()`
2. `is_straight_flush()`
3. `is_four_of_a_kind()`
4. `is_full_house()`
5. `is_flush()`
6. `is_straight()`
7. `is_three_of_a_kind()`
8. `is_two_pair()`
9. `is_pair()`

Each should take a 5-card hand and return `TRUE` or `FALSE` depending whether the five cards satisfy the criteria for the type of hand. In writing these functions, you may find expressions like this useful:

- `min(table(suits(five_cards))),`
- `max(table(suits(five_cards))) == 5,`
- `min(table(ranks(five_cards))),`
- `max(diff(sort(ranks(five_cards)))) == 1`
- `min(ranks(five_cards)) == 10`
- and so on.

You will also find the `&&` function useful, as in this test for a full house.[1]

```
counts <- table(ranks(five_cards))
min(counts) == 2 && max(counts) == 3
```

[1] There are two similar functions, `&` and `&&`. Use `&` when you want to compare vectors of any length; it works pairwise on all the elements. Use `&&` when combining vectors of length 1, as in `if` statements. `&&` (and its cousin `||`) doesn't evaluate additional conditions once it knows what the result will be.

Make a few test hands with cards of your choosing. Call these `test_hand_1`, `test_hand_2`, and so on.

Try out your `is_` functions on these hands and confirm that your functions work.

Estimating probabilities

Write a function `before_draw()` that draws five random cards from the deck and calculates the best hand that can be assembled from those cards. It will look like:

```
before_draw <- function(...) {
  five_cards <- sample(poker_deck, 5)
  if (is_full_house(five_cards))  ... and so on ...
}
```

Tabulating the probabilities

Run your `before_draw()` function many times, collecting the results. Tabulate these and compare your results to those claimed on the Wikipedia page. A command to do this will look like:

```
table(sapply(1:10000, FUN = before_draw))
```

Just for fun . . . Texas Hold'em and Seven Card Stud

Use the functions you've written to produce one that gives the best 5-card hand possible out of 7 cards dealt.