Rdice-vignette

Gennaro Tedesco

September 12, 2016

Introduction

The following guide provides a description with examples of the main features of the package `Rdice`. Such package provides basic functions to experiment dice rollings and corresponding outputs frequencies, coin flips, non-transitive Efron dice generators and simulations of percolations in one dimension.

Dice experiments

The function `dice.roll`

The function `dice.roll` simulates rolling of a set of dice given their number of faces and the probability of each face to appear. Arguments to be passed are

- faces: the number of faces the dice have; if unspecified, it defaults to 6.
- dice: the number of dice to roll; if unspecified, it defaults to 2.
- rolls: the number of times to roll the die; if unspeciefid, it defaults to 5.
- weights: a vector of probability weights to assign to each face of the die; if unspecified, it defaults to a fair die with weights $1/N$. If specified, its lenght must obviously be equals to the number of faces and all the single weights must sum up to 1.

The function returns an object of class *diceRoll*, namely a list whose values are themselves `data.table` objects, in turn, so that one can directly apply any `data.table` function thereupon. The values returned are:

- results: the numerical results rolled.

```
set.seed(1902)
dice.roll(faces = 6, dice = 2, rolls = 5)$results
   die_1 die_2
1:     6     1
2:     1     5
3:     6     5
4:     6     1
5:     6     5
```

- frequencies: a table containing each single occurrency (permutation) of results and the corresponding frequencies.

```
set.seed(1902)
dice.roll(faces = 6, dice = 2, rolls = 5)$frequencies
   die_1 die_2 N freq
1:     6     1 2  0.4
2:     1     5 1  0.2
3:     6     5 2  0.4
```

- sums_freq: a table containing the frequencies of the sums of the values obtained in each single roll by all the dice. A cumulative sum is provided too.

```
set.seed(1902)
dice.roll(faces = 6, dice = 2, rolls = 5)$sums_freq
   sum N freq cum_sum
1:   6 1  0.2     0.2
2:   7 2  0.4     0.6
3:  11 2  0.4     1.0
```

- exp_value_sum: the expectation value of the sum of the values obtained

```
set.seed(1902)
dice.roll(faces = 6, dice = 2, rolls = 5)$exp_value_sum
[1] 8.4
```

The function `dice.combinations`

The function `dice.combinations` calculates all possible combinations as result of rolling a set of dice. Similar permutations are identified under the same combination and counted as many times as many occurrencies. Thee user can choose wheter to match exact values or to perform partial matches. The function accepts as arguments:

- faces: the number of faces the dice have; if unspecified, it defaults to 6.
- dice: the number of dice to roll; if unspecified, it defaults to 2.
- rolls: the number of times to roll the die; if unspeciefid, it defaults to 5.
- weights: a vector of probability weights to assign to each face of the die; if unspecified, it defaults to a fair die with weights $1/N$. If specified, its lenght must obviously be equals to the number of faces and all the single weights must sum up to 1.
- getExact: a vector containing values to be matched *exactly*, namely the function returns only those combinations containing *all* the above mentioned variables.
- getPartial: a vector containing values to be matched *partially*, namely the function returns only those combinations containing *at least one* of the above mentioned variables. If missing, it defaults to `c(1:faces)`, namemly the function returns all combinations.
- toSum: a logical value, defaulting to FALSE. If TRUE, the function returns the sum of the frequencies of the matches (to be used together with getExact or getPartial)

As an example let us roll 3 six-faces dice many times and look for the frequencies of obtaining a one, a four and either a five or a six.

```
set.seed(1987)
dice.combinations(faces = 6, dice = 3, rolls = 1000,
                  getExact = c(1,4), getPartial = c(5,6))
   value_1 value_2 value_3  freq
1:       1       4       5 0.018
2:       1       4       6 0.027
```

If we want to just obtain the overall probability of the above events then

```
set.seed(1987)
dice.combinations(faces = 6, dice = 3, rolls = 1000,
                  getExact = c(1,4), getPartial = c(5,6), toSum = TRUE)
[1] 0.052
```

The case `faces = 2` corresponds to the `coin.flip`.

The function `coin.flip`

The function `coin.flip` is a particular case of the function `dice.combinations`. Valid arguments are

- coins: the number of coins to flip. If unspecified, it defaults to 5.
- flips: the number of flips. If missing, it defaults to 100.
- weights: a vector of probability weights to assign to each face of the coin; if unspecified, it defaults to a fair coin with equally likely faces. If specified, its lenght must obviously be a vector of length two whose values sum up to 1.
- getExact: a vector containing values to be matched *exactly*, namely the function returns only those combinations containing *all* the above mentioned variables. Since this is a coin flip, values must be specified in the form `c("H", "H", "T")` as head and tails (make sure to provide the labels in quotation marks).

Let us experiment with 5 coins and see what the probability of tossing five heads is, assumed the coin is fair.

```
set.seed(1902)
coin.flip(coins = 5, flips = 10000, getExact = replicate(5, c("H")))
   value_1 value_2 value_3 value_4 value_5   freq
1:       H       H       H       H       H 0.0311
```

Notice that if we had chosen a p-value of 0.05 this would allow us to reject the hypothesis that the coin is fair, although we know by default it is. A good indication that one experiment against the p-value threshold does not make much sense. By the way the value we obtained, 0.0311, is pretty close to the actual probability value of tossing five heads in a row $P(5 \text{ heads}) = (1/2)^5 = 0.03125$.

Non-transitive dice

The function `is.nonTransitive`

The function `is.nonTransitive` checks whether a given set of dice is non-transitive. The dice must be inputted as a data frame with one column for each die:

```
# generates a data frame of dice
df <- data.frame(
  die1 = c(5,4,6,15),
  die2 = c(3,6,16,2),
  die3 = c(0,2,14,15),
  die4 = c(8,10,1,9))
df
```

```
##   die1 die2 die3 die4
## 1    5    3    0    8
## 2    4    6    2   10
## 3    6   16   14    1
## 4   15    2   15    9
```

```
is.nonTransitive(df, prob = 9/16)
```

```
## [1] TRUE
```

where we have specified the non-transitive probability to check against. For general non-transitive dice with different mutual probabilities we can leave the argument `prob` unspecified and let `R` do the verification for us: for example

```
# generates a data frame of dice
df <- data.frame(
  die1 = c(5,14,0,14),
  die2 = c(7,10,4,13),
  die3 = c(12,9,2,8),
  die4 = c(1,7,16,8))
df
```

```
##   die1 die2 die3 die4
## 1    5    7   12    1
## 2   14   10    9    7
## 3    0    4    2   16
## 4   14   13    8    8
```

```
is.nonTransitive(df)
```

```
## [1] TRUE
```

as different dice have different probabilities to mutually win against each other.

The function `nonTransitive.generator`

The function `nonTransitive.generator` generates a random set of non-transitive dice matching the conditions given as arguments. It internally generates random sets of dice and applies the `is.nonTransitive` thereupon until a match is found. If no matches occur, the function times out after a certain amount of time and returns a `NULL` value. Valid arguments are:

- dice: The number Z of non-transitive dice to generate
- faces: The number of faces of each die.
- max_value: The maximum integer allowed as nominal value for the faces. Standard choices are usually max_value = faces (default) or max_value = faces^2.
- prob: The probability one wants the set of dice to be non-transitive. If unspecified, a set of dice with different non-transitive probabilities for each pairing will be generated.
- error: Computational error to check for machine precision equality. It defaults to 0.001: no need to be specified.

```
set.seed(1902)
nonTransitive.generator(faces = 3, dice = 3, max_value = 9, prob = 5/9)

   die_1 die_2 die_3
1:     4     3     8
2:     6     2     5
3:     3     9     0
```

Had we not specified the probability, an asymettric non-transitive set of dice would have been generated.

```
nonTransitive.generator(4,4)
   die_1 die_2 die_3 die_4
1:     5     7    12     1
2:    14    10     9     7
3:     0     4     2    16
4:    14    13     8     8
```

Data sets

The package contains the following data sets for testing:

```
data(efron)
   die1 die2 die3 die4
1:    0    3    2    1
2:    0    3    2    1
3:    4    3    2    1
4:    4    3    2    5
5:    4    3    6    5
6:    4    3    6    5

data(oskar)
   die1 die2 die3 die4 die5 die6
1:    2    7    5    1    6    4
2:    2    7    5    1    6    4
3:   14   10   13   12    8   11
4:   14   10   13   12    8   11
5:   17   16   15   20   19   18
6:   17   16   15   20   19   18

data(miwin)
   die1 die2 die3
1:    1    1    2
2:    2    3    3
3:    5    4    4
4:    6    5    6
5:    7    8    7
6:    9    9    8
```