Baiamonte Matteo,
Licciardi Maria Teresa,
Mucelli Eleonora

BOGGLE GAME AND PROBABILITY: HOW TO COMPUTE PROBABILITY IN R

CA' FOSCARI UNIVERSITY

ECONOMICS AND MANAGEMENT

ACADEMIC YEAR 2011-2012

LECTURER PELLIZZARI PAOLO

INTRODUCTION

"Boogle" is a game in which there are 16 different dice, each one with 6 letters, that can be vowels and consonants, disposed in 4 columns and 4 rows (see the picture on the right). The goal of this game is to create logical and existing words after rolling all the 16 dice simultaneously. Specifically the aim of this user guide is to calculate, using the R-program, the probability that by throwing the dice we will get no vowels.

How to set the problem with R:

STEP1. Create a dummy variable for consonants and vowels;

STEP2. Count the probability of consonants in each die;

STEP3. Compute the total probability.

GETTING STARTED

To begin you have to prepare a list, using the function *list()* in R*,* containing 16 vectors, each of which represents the 6 letters of each dice. In the example we have called each vector “d1, d2,...,d16”. Let's have a more detailed look at what is going on:

```
l<-list(d1=c('S','R','E','L','A','C'),
d2 =c('D','P','A','C','E','M'),
d3=c('Q','B','A','O','J','M'),
d4 =c('D','U','T','O','K','N'),
d5=c('O','M','H','R','S','A'),
d6 =c('E','I','F','E','H','Y'),
d7 =c('B','R','I','F','O','X'),
d8 =c('R','L','U','W','I','G'),
d9 =c('N','S','O','W','E','D'),
d10 = c('Y ','L','I','B','A','T'),
d11 =c('T','N','I','G','E','V'),d12 =c('T','A','C','I','T','O'),
d13 =c('P','S','U','T','E','L'),d14 =c('E','P','I','S','H ','N'),
d15 =c('Y','K','U','L','E','G'),d16 =c('N','Z','E','V','A','D'))
```

WARNING:

Remember to use properly " " for each letter, otherwise you will get the error "*object not found*" because R does not automatically recognize the letters of the alphabet as objects of a vector. Besides, even though the assignment operator “<-“ is the same as “=” avoid using the arrow because you are writing vectors within the *list* function.

We used the 16 dice provided by the boggle game, but you can create your own random group of dice in R and generalize this problem in order to test the probability of getting all consonants on randomly selected die. Try this command:

```
v<-c("d1","d2","d3")
f<-function(x) x<-sample(letters,6,replace=T)
sapply(v,f)
```

STEP1. CREATING A DUMMY VARIABLE

Now we have to create a dummy variable (a variable with only two levels: 0 and 1) that assigns the value 0 to consonants and the value 1 to vowels. You can also do the opposite: assign 1 to consonants and 0 to vowels, it is up to you. In this way we are able to count vowels and consonants for each dice.

```
di<-c(l$d1,l$d2,l$d3,l$d4,l$d5,l$d6,l$d7,l$d8,l$d9,l$d10,
l$d11,l$d12,l$d13,l$d14,l$d15,l$d16)
```

Create a new vector that concatenate all sixteen vectors created before; this is done in order to create a generalized formula for the dummy variable, otherwise we should have to rewrite it for sixteen times changing the dice of reference. Be aware, the vector does not contain only 16 objects. Through the dollar sign $ we extract from the list “l” each vector of six components, then with the function “c” we concatenate all this vectors of 6 objects together in the new vector “di”.

1. Call “n” the length of the new vector where the length function gives the number of objects contained into the vector “di”, in this case it is 96.

```
n<-length(di)
```

2. Create a vector named “l$dummy” of 16 components, all equal to 0. The function *`rep(x, times)`* in fact replicates the value 0 (first term in the parenthesis) for n times (second term in the parenthesis), in this case for 96 times. We called it “l$dummy” because we are still working on the values of our list but we are transforming them into a vector where all components can assume value 0 or 1.

```
l$dummy<-rep(0,n)
```

3. Use the square brackets to extract from the vector “l$dummy” all the components that corresponds to vowels. Specifically, we want each component of “di”, that is identically equal (==) to one vowel, to be assigned the value 1, overwriting the previous vector where all components had value zero (l$dummy). Note that we used the or operator (|), because we want R to look for either the vowel “A” or the vowel “O”, and so on.

```
l$dummy[(di=="A")|(di=="E")|(di=="I")|(di=="O")|(di=="U")]<-1
```

STEP2. COMPUTE THE PROBABILITY OF CONSONANTS FOR EACH DIE

In this section we explain how to compute the probability of getting all consonants when throwing the sixteen dice of the game at once. Recall from statistics theory that those events are stochastically independent, as a result the probability of getting all consonants is simply the product of the single probabilities to have a consonant per die.

Our aim is to create a command that induces R to compute the probability of each segment of the dummy variable “l$dummy”, that represents each single dice “d1, d2, …, d16”. In this way we will be able to compute separately the probabilities of getting consonants for each die and then we just have to multiply the results.

1. Use function `seq()` to create a succession of numbers indicating the extremes of the intervals that include letters for each die. In fact, as you can read from the help page (enter `help(seq)` to see it on R) this function generates regular sequences, where by regular we mean that the increment is fixed.

```
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),length.out = NULL)
```

As you can see this is a generic function with many defaults. Hence, the reader has to provide:

- `From, to`: The extremes of the sequence. Notice that if these values are not supplied by the user, R will assume by default that they are equal to 1.
- `By`: The increment of the sequence, roughly speaking the distance between consequent values of the list.

- `Length.out`: the desired length of the sequence. Usually if the increment is given, the length.out will be computed automatically by R. The opposite is true.

Using this function we create two different sequences:

```
beg<-seq(1,96,by=6)

end<-seq(6,96,by=6)
```

- "beg": the sequence of lower hand extremes of intervals, where intervals are simply the faces of each die. As a consequence we provide as beginning value of the sequence 1, as ending value 96 and as increment 6.
  This command creates the following sequence: 1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91.
- "end": the sequence of upper hand extremes of intervals. In this case the progression we get is: 6 12 18 24 30 36 42 48 54 60 66 72 78 84 90 96.

WARNING:
In this step there are not large difficulties in using function "seq" because you have to provide just from, to and either "by" or "length.out" and R makes the job for you. Nevertheless, it is important that you have clear in mind before you start typing what is the sequence you want to create according to the intervals you want to extract from the dummy variable you created above.

```
dprob<-rep(0,16)
```

2. Create a vector with sixteen zeros using the function rep() which stands for replicate.
   You may now wonder: Why are we are creating another vector of zeros?
   This vector is just a temporary object that we will overwrite in the next step substituting zeros with the values of the probabilities of each dice. If we do not define dprob, R cannot find this object and it is unable to store the results of the following computations into this vector.

3. Compute probability of getting consonants in each die, remembering that these probabilities are just the sum of number of consonants in each die divided by 6 (number of faces).
   You could compute this probability using formula `sum(l$dummy[0:6]==0)/6`. Nevertheless, in this way you will compute only the probability for the first die ad you will have to repeat this formula 16 times in order to have the probability of all dice in the game.

   To compute multiple sums of different elements use the loop `for(var in seq)`. This command performs a function several times, using in each cycle different inputs.

   When dealing with "for" loop users have to provide:
   - `Var`: the name of the variable they will use in the subsequent function. We used "i" that stands for index, but you can set the name you like the most (mickeymouse is also fine). It is important that you understand that this variable will be the argument of the function you want to iterate.
   - `Seq`: a sequence or a vector describing the values that you want "var" to take when iterating the function. In this case, we set `for(i in 1:16),` because we want to use "i" to indicate the i^{th} component of vectors beg and end described above (remember that those vectors have both length equal to 16, indicating the number of dice in the game).

4. In order to produce some result, the "for" loop has always to be associated to another function. The function we are iterating in this case is `sum(x)`. To compute those values we simply use the results of our previous computations: "beg, end, l$dummy".

```
for(i in 1:16)
dprob[i]<-sum(l$dummy[beginning[i]:end[i]]==0)/nfaces
```

om the factor "l$dummy", we extract the values of the dummy variable corresponding to each die.

To perform this operation:

- First of all, extract the i^{th} component of the beg vector. This value will indicate the beginning of the interval in which we are counting consonants. For example, if `beg[i]=1` we are creating an interval that starts from 1 and in which we will count the consonants present in the first die.
- Extract the i^{th} component of the end vector, which, on the other hand, will indicate the end of the interval. By extracting "i" from these vectors, and not a specific value, we let R perform this operation for each value of "i", which varies from 1 to 16.
- Sum only consonants for each die. To perform this operation we use the sum function but we need to constrain the field of action of the function to elements that correspond to consonants. This is done by setting elements we extracted from "l$dummy" (our dummy variable) identical to zero. In fact, our dummy variable was created by assigning 1 to vowels and 0 to consonants.

WARNING:
Notice that in vector "dprob[i]" we extracted value "i" in order to make R understand that we want to store results of the function for all values of "i", defined in the "for" loop. In fact, if you simply call this vector dprob, the program will store just the first result of the computation for "i" equal to the first component of sequences "beg" and "end".

As you can see the result that you get are counts of number of consonants per die.

5. Now we have to divide each count by the number of faces.
We define the vector "nfaces" taking as reference die "d1", since we know that all dice have same number of faces.

```
nfaces<-length(l$d1)
```

We use this result to divide the counts obtained in the previous formula.
We then store the results of our computation in vector "dprob", which was defined in point 2 and that we are now overwriting.
The content of vector "dprob" is now the following:

```
[1] 0.6666667 0.6666667 0.6666667 0.6666667 0.6666667 0.5000000 0.6666667
[8] 0.6666667 0.6666667 0.6666667 0.6666667 0.5000000 0.6666667 0.6666667
[15] 0.6666667 0.6666667
```

STEP3. COMPUTE THE TOTAL PROBABILITY

```
Prod(dprob)
```

Now that we have all single independent probabilities we have to take their product, in order to get the final probability.
To reach this objective we use function `prod(..., na.rm = FALSE)`.
The result of this computation is: **0.0008563718**. We can conclude that the probability of getting all consonants in the Boggle game is equal to approximately 0.08%. Considering 10 000 trials the player will get all consonants in 8 turns and he will be unable to create meaningful words.

WARNING:
Notice that function `prod()` can be applied only to perform vector elements products.