IE6200: Engineering Probability and Statistics

LAB 02: Sample Space and Probabilities

Prof. Mohammad Dehghani

Contents

1 R Packages

R packages are collections of functions and data sets developed by the **R** community. They increase the power of R by improving existing base **R** functionalities, or by adding new ones. In this lab, we will make use of prob package, which is a framework for performing elementary probability calculations on finite sample spaces, which may be represented by data frames or lists. Functionality includes setting up sample spaces, counting tools, defining probability spaces, performing set algebra, calculating probability and conditional probability, tools for simulation and checking the law of large numbers, adding random variables, and finding marginal distributions. that share a common design philosophy.

1.1 Installing and loading packages

We can install the package usinge the install.packages() function. This downloads the package from an online repository such as CRAN and installs it to your local machine.

```
install.packages("prob") # install the prob package
```

You only need to install a package once, but you need to *reload* it every time you start a new session.

In order to load the package in the R session, we use the library() function. This loads the package in R so that the functions associated with it can be used.

```
library(prob) # load the prob package
```

To know more about the prob package type ?prob in the editor

2 Sample Spaces

The first step is to set up a **sample space**, or set of possible outcomes of an experiment. In theprob package, the most simple form of a sample space is represented by a dataframe, that is, a rectangular collection of variables. Each row of the dataframe corresponds to an outcome of the experiment.

2.1 Some Standard Sample Spaces

2.1.1 tosscoin() Function

Consider the experiment of tossing a coin. The outcomes are **H** or **T**. We can set up the sample space quickly with the tosscoin() function:

where: * **times:** the number of times we want to toss a coin. * **makespace:** this argument takes a logical value. If TRUE the dataframe shows the probabilities for each event in the sample space. If FALSE the data frame only shows the sample space.

Example 1: Find the sample space and the probabilities of tossing a coin once

```
tc1 <- tosscoin(times = 1, makespace = TRUE)
tc1
```

```
##   toss1 probs
## 1     H   0.5
## 2     T   0.5
```

Example 2: Find the sample space of tossing a coin once

```
tc1 <- tosscoin(times = 1, makespace = FALSE)
tc1
```

```
##   toss1
## 1     H
## 2     T
```

📢 **Activity**: Try finding the sample space of tossing a coin two times.

2.1.2 rolldie() function

Consider the experiment of rolling a fair die: For this experiment we use the rolldie() function.

where: * **times:** the number of times we want to toss a coin. * **makespace:** this argument takes a logical value. If TRUE the dataframe shows the probabilities for each event in the sample space, If FALSE the data frame only shows the sample space. * **nsides:** the number of sides the die has.

Example 3: Find the sample space and the probabilities of rolling a fair die once

```
rolldie(times = 1, nsides = 6, makespace = TRUE)
```

```
##   X1     probs
## 1  1 0.1666667
## 2  2 0.1666667
## 3  3 0.1666667
## 4  4 0.1666667
## 5  5 0.1666667
## 6  6 0.1666667
```

📢 **Activity**: Can you find the sample space of rolling a die twice ?.

2.1.3 cards() function

Consider the experiment of drawing cards from a pack of cards: For this experiment we use the cards() function.

where: * **jokers:** takes a logical value, if FALSE indicates that jokers should not be included in the deck. * **makespace:** this argument takes a logical value. If TRUE the dataframe shows the probabilities for each event in the sample space, if FALSE the data frame only shows the sample space.

Example 4: Finding the sample space and probabilities for all the cards (joker excluded) present in a deck of cards.

```
cards(jokers = FALSE, makespace = TRUE)
```

```
##    rank    suit      probs
## 1     2    Club 0.01923077
## 2     3    Club 0.01923077
## 3     4    Club 0.01923077
## 4     5    Club 0.01923077
## 5     6    Club 0.01923077
## 6     7    Club 0.01923077
## 7     8    Club 0.01923077
## 8     9    Club 0.01923077
## 9    10    Club 0.01923077
## 10    J    Club 0.01923077
## 11    Q    Club 0.01923077
## 12    K    Club 0.01923077
## 13    A    Club 0.01923077
## 14    2 Diamond 0.01923077
## 15    3 Diamond 0.01923077
## 16    4 Diamond 0.01923077
## 17    5 Diamond 0.01923077
## 18    6 Diamond 0.01923077
## 19    7 Diamond 0.01923077
## 20    8 Diamond 0.01923077
## 21    9 Diamond 0.01923077
## 22   10 Diamond 0.01923077
## 23    J Diamond 0.01923077
## 24    Q Diamond 0.01923077
## 25    K Diamond 0.01923077
## 26    A Diamond 0.01923077
## 27    2   Heart 0.01923077
## 28    3   Heart 0.01923077
## 29    4   Heart 0.01923077
## 30    5   Heart 0.01923077
## 31    6   Heart 0.01923077
## 32    7   Heart 0.01923077
## 33    8   Heart 0.01923077
## 34    9   Heart 0.01923077
## 35   10   Heart 0.01923077
## 36    J   Heart 0.01923077
## 37    Q   Heart 0.01923077
## 38    K   Heart 0.01923077
## 39    A   Heart 0.01923077
## 40    2   Spade 0.01923077
```

```
## 41    3   Spade 0.01923077
## 42    4   Spade 0.01923077
## 43    5   Spade 0.01923077
## 44    6   Spade 0.01923077
## 45    7   Spade 0.01923077
## 46    8   Spade 0.01923077
## 47    9   Spade 0.01923077
## 48   10   Spade 0.01923077
## 49    J   Spade 0.01923077
## 50    Q   Spade 0.01923077
## 51    K   Spade 0.01923077
## 52    A   Spade 0.01923077
```

Example 5: Finding the sample space and probabilities for all the cards (joker included) present in a deck of cards.

```
cards(jokers = TRUE, makespace = TRUE)
```

```
##     rank    suit      probs
## 1      2    Club 0.01851852
## 2      3    Club 0.01851852
## 3      4    Club 0.01851852
## 4      5    Club 0.01851852
## 5      6    Club 0.01851852
## 6      7    Club 0.01851852
## 7      8    Club 0.01851852
## 8      9    Club 0.01851852
## 9     10    Club 0.01851852
## 10     J    Club 0.01851852
## 11     Q    Club 0.01851852
## 12     K    Club 0.01851852
## 13     A    Club 0.01851852
## 14     2 Diamond 0.01851852
## 15     3 Diamond 0.01851852
## 16     4 Diamond 0.01851852
## 17     5 Diamond 0.01851852
## 18     6 Diamond 0.01851852
## 19     7 Diamond 0.01851852
## 20     8 Diamond 0.01851852
## 21     9 Diamond 0.01851852
## 22    10 Diamond 0.01851852
## 23     J Diamond 0.01851852
## 24     Q Diamond 0.01851852
## 25     K Diamond 0.01851852
## 26     A Diamond 0.01851852
## 27     2   Heart 0.01851852
## 28     3   Heart 0.01851852
## 29     4   Heart 0.01851852
## 30     5   Heart 0.01851852
## 31     6   Heart 0.01851852
## 32     7   Heart 0.01851852
## 33     8   Heart 0.01851852
## 34     9   Heart 0.01851852
```

```
## 35    10   Heart 0.01851852
## 36     J   Heart 0.01851852
## 37     Q   Heart 0.01851852
## 38     K   Heart 0.01851852
## 39     A   Heart 0.01851852
## 40     2   Spade 0.01851852
## 41     3   Spade 0.01851852
## 42     4   Spade 0.01851852
## 43     5   Spade 0.01851852
## 44     6   Spade 0.01851852
## 45     7   Spade 0.01851852
## 46     8   Spade 0.01851852
## 47     9   Spade 0.01851852
## 48    10   Spade 0.01851852
## 49     J   Spade 0.01851852
## 50     Q   Spade 0.01851852
## 51     K   Spade 0.01851852
## 52     A   Spade 0.01851852
## 53 Joker    <NA> 0.01851852
## 54 Joker    <NA> 0.01851852
```

2.2 Sampling From Urns

Perhaps the most fundamental of statistical experiments consists of drawing distinguishable objects from an urn. The prob package addresses this topic with the urnsamples() function.

where: * **x:** represents the urn from which sampling is to be done. * **size:** argument tells how large the sample will be. * **ordered:** if TRUE order in which sampling is done matters. * **replace:** if TRUE the sample will be returned back to the population before the selection of the next sample

Let us define a character vector balls that contains the red, green and blue colored balls that have to be picked from an urn.

```
balls <- c("red", "green", "blue")
balls
```

```
## [1] "red"   "green" "blue"
```

2.2.1 Ordered, with replacement

If sampling is with **replacement**, then we can get any outcome red, green, blue on any draw. Further, by **"ordered"** we mean that we shall keep track of the order of the draws that we observe

Example 6: Find the sample space of picking 2 balls with replacement from an urn containing 3 different colored balls where order matters.

```
urnsamples(balls, size = 2, replace = TRUE, ordered = TRUE)
```

```
##      X1    X2
## 1   red   red
## 2 green   red
## 3  blue   red
## 4   red green
## 5 green green
## 6  blue green
## 7   red  blue
## 8 green  blue
## 9  blue  blue
```

Notice that rows 2 and 4 are not identical, the order in which the balls are picked does matter here.

2.2.2 Ordered, without replacement

Here sampling is without replacement, so we may not observe the same colored ball twice in any row. Order is still important, however, so we expect to see the outcomes red, green and green, red somewhere in our data frame as before. **Example 7: Find the sample space of picking 2 balls with replacement from an urn containing 3 different colored balls where order matters.**

```
urnsamples(balls, size = 2, replace = FALSE, ordered = TRUE)
```

```
##      X1    X2
## 1   red green
## 2 green   red
```

```
## 3   red  blue
## 4  blue   red
## 5 green  blue
## 6  blue green
```

Notice that there are less rows in this answer, due to the restricted sampling procedure.

2.2.3 Unordered, with replacement

Here,we replace the balls after every draw, but we do not remember the order in which the draws come

Example 7: Find the sample space of picking 2 balls with replacement from an urn containing 3 different colored balls where order does not matter.

```
urnsamples(balls, size = 2, replace = TRUE, ordered = FALSE)
```

```
##      X1    X2
## 1   red   red
## 2   red green
## 3   red  blue
## 4 green green
## 5 green  blue
## 6  blue  blue
```

2.2.4 Unordered, without replacement

Example 8: Find the sample space of picking 2 balls without replacement from an urn containing 3 different colored balls where order does not matter.

Again, we may not observe the same outcome twice, but in this case, we will only keep those outcomes which (when jumbled) would not duplicate earlier ones.

```
urnsamples(balls, size = 2, replace = FALSE, ordered = FALSE)
```

```
##      X1    X2
## 1   red green
## 2   red  blue
## 3 green  blue
```

This experiment is equivalent to reaching in the urn, picking a pair, and looking to see what they are. This is the default setting of urnsamples(), so we would have received the same output by simply typing urnsamples(balls,2).

3 Counting Tools

In many cases, we do not need to actually generate the sample spaces of interest; it suffices merely to count the number of outcomes. The nsamp() function will calculate the number of rows in a sample space made by urnsamples(), without actually devoting the memory resources necessary to generate the space. The nsamp() function takes the following arguments:

- **n:** number of (distinguishable) objects in the urn.
- **k:** the sample size.
- **ordered:** If TRUE order in which sampling is done matters.
- **replace:** If TRUE the sample will be returned back to the population before the selection of the next sample

We will compute the number of outcomes for each of the four urnsamples() examples that we saw in the last section. Recall that we took a sample of size two from an urn with three distinguishable element

```
nsamp(n=3, k=2, replace = TRUE, ordered = TRUE)
```

```
## [1] 9
```

```
nsamp(n=3, k=2, replace = FALSE, ordered = TRUE)
```

```
## [1] 6
```

```
nsamp(n=3, k=2, replace = FALSE, ordered = FALSE)
```

```
## [1] 3
```

```
nsamp(n=3, k=2, replace = TRUE, ordered = FALSE)
```

```
## [1] 6
```

4 Subsets

4.1 Finding Subsets

There are plenty of existing methods for finding subsets of data frames, so we will only mention some of them here.

4.1.1 Method 1 (Indexing)

Given a data frame sample/probability space S, we may extract rows using the *[]* operator:

Let S be the event of tossing a fair coin twice.

```
S <- tosscoin( times = 2, makespace = TRUE)
S
```

```
##   toss1 toss2 probs
## 1     H     H  0.25
## 2     T     H  0.25
## 3     H     T  0.25
## 4     T     T  0.25
```

```
S[1:3, ]
```

```
##   toss1 toss2 probs
## 1     H     H  0.25
## 2     T     H  0.25
## 3     H     T  0.25
```

4.1.2 Method 2 (subset() funtion)

We may also extract rows that satisfy a logical expression using the subset() function, for instance

```
S <- cards()
subset(S, suit == "Heart")
```

```
##    rank  suit
## 27    2 Heart
## 28    3 Heart
## 29    4 Heart
## 30    5 Heart
## 31    6 Heart
## 32    7 Heart
## 33    8 Heart
## 34    9 Heart
## 35   10 Heart
## 36    J Heart
## 37    Q Heart
## 38    K Heart
## 39    A Heart
```

The ouput contains only those cards that belong to the suit heart.

```
subset(rolldie(3), X1+X2+X3 > 16)
```

```
##     X1 X2 X3
## 180  6  6  5
## 210  6  5  6
## 215  5  6  6
## 216  6  6  6
```

The output will ony contain those events where the sum of the first roll, second roll, and third roll of a die is greater than 16.

4.2 Functions for Finding Subsets

4.2.1 The function isin() or %in%

The **%in%** can be used in the following way.

```
x <- 1:10
y <- 3:7
y %in% x
```

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

Notice that the value is a vector of length 5 which tests if each element of y is in x, in turn. But perhaps we would like to know whether the whole vector y is in x. We can do this with the isin() function.

```
isin(x,y)
```

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

Of course, one may ask why we did not try something like \textcolor{blue}{all(y %in% x)}, which would give a single result, TRUE. The reason is that the answers are different in the case that y has repeated values. Compare:

```
x <- 1:10
y <- c(3,3,7)
all(y %in% x)
```

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

```
isin(x,y)
```

```
## [1] FALSE
```

The reason is of course that x contains the value 3, but it doesn't have two 3's. The difference becomes significant when rolling multiple dice, playing cards, etc.

There is an argument **ordered**, which tests whether the elements of y appear in x in the order in which they are specified in.

```
isin(x, c(3,4,5), ordered = TRUE)
```

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

```
isin(x, c(3,5,4), ordered = TRUE)
```

```
## [1] FALSE
```

5 Calculating probabilities

Now that we have ways to find subsets, we can at last move to calculating the probabilities associated with them. This is accomplished with the Prob() function. Consider the experiment of drawing a card from a standard deck of playing cards.

where: ***S:*** *the probability space associated with the experiment.* **A:** event of having cards that belong to the suit of hearts. ***B:** event of having cards with the rank from 7 to 9.

```
S <- cards(makespace = TRUE)
A <- subset(S, suit == "Heart")
B <- subset(S, rank %in% 7:9)
```

Now it is easy to calculate the probabilities of each event.

```
Prob(A)
```

```
## [1] 0.25
```

```
Prob(B)
```

```
## [1] 0.2307692
```

```
Prob(S)
```

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

5.1 Conditional probability

The conditional probability of the subset A given the event B is defined to be $\boxed{P(A \cap B) = P(A \mid B)P(B)}$

Only one additional **given** argument has to be added to the Prob() function,to find the conditional probability. Using the above example with S={draw a card}, A ={suit = "Heart"}, and B={rank is 7, 8, or 9}.

Example 9: What is the probability that a card drawn with rank 7,8,or 9 belongs to the suit of hearts?.

```
Prob(A, given = B)
```

```
## [1] 0.25
```

Example 10: What is the probability that a card drawn belonging to the suit of hearts has a rank 7,8,or 9?.

```
Prob(B, given = A)
```

```
## [1] 0.2307692
```

6 Adding Random Variables

The following section is for self study.You'll can try running the functions given below.

This section describes how to define random variables based on an experiment and how to examine their behavior once defined. The primary vessel we use for this task is the addrv() function. There are two ways to use it, and will describe both.

6.1 Supplying a defining formula

The idea is to write a formula defining the random variable inside the function, and it will be added as a column to the data frame. As an example, let us roll a 6-sided die three times, and let's define the random variable $\boxed{U = X1 - X2 + X3}$

```
S <- rolldie(3, nsides = 4, makespace = TRUE)
S <- addrv(S, U = X1-X2+X3)
```

Now let's take a look at the values of U. In the interest of space, we will only reproduce the first few rows of S.

```
S[1:9,]
```

```
##   X1 X2 X3  U    probs
## 1  1  1  1  1 0.015625
## 2  2  1  1  2 0.015625
## 3  3  1  1  3 0.015625
## 4  4  1  1  4 0.015625
## 5  1  2  1  0 0.015625
## 6  2  2  1  1 0.015625
## 7  3  2  1  2 0.015625
## 8  4  2  1  3 0.015625
## 9  1  3  1 -1 0.015625
```

We can now answer questions like Find the probability such that X1-X2+X3 $>$ 6.

```
Prob(S, U > 6)
```

```
## [1] 0.015625
```

6.2 Supplying a function

Sometimes we have a function laying around that we would like to apply to some of the outcome variables, but it is unfortunately tedious to write out the formula defining what the new variable would be. The addrv() function has an argument **FUN** specifically for this case. Its value should be a legitimate function from R, such as sum, mean, median, etc. Or, you can define your own function.

Continuing the previous example, let's define V = max(X1, X2, X3) and W = X1 + X2 + X3.

```
S <- addrv(S, FUN = max, invars = c("X1","X2","X3"), name = "V")
S <- addrv(S, FUN = sum, invars = c("X1","X2","X3"), name = "W")
S[1:9,]
```

```
##   X1 X2 X3  U V W    probs
## 1  1  1  1  1 1 3 0.015625
## 2  2  1  1  2 2 4 0.015625
## 3  3  1  1  3 3 5 0.015625
## 4  4  1  1  4 4 6 0.015625
## 5  1  2  1  0 2 4 0.015625
## 6  2  2  1  1 2 5 0.015625
## 7  3  2  1  2 3 6 0.015625
## 8  4  2  1  3 4 7 0.015625
## 9  1  3  1 -1 3 5 0.015625
```

Notice that addrv() has an invars argument to specify exactly to which columns one would like to apply the function **FUN**. If no input variables are specified, then addrv() will apply **FUN** to all non-probs columns. Further, addrv() has an optional argument name to give the new variable; this can be useful when adding several random variables to a probability space (as above). If not specified, the default name is "X"