ISTA 116 Lab: Week 10

Last Revised October 24, 2011

1 Homework 4 Questions?

2 The `prob` package

We will be working with the `prob` package for `R`, which you will need to install, using the following command. If you don't have admin permissions on your machine, you will have to run this command each time you log in.

```
> install.packages('prob', repos='http://cran.opensourceresources.org')
```

```
> library(prob)
```

Note: Most of these examples are lifted from chapter 4 of Kerns (2010) [1], available here.

3 Sample Spaces

The sample space Ω is the set of all possible distinguishable outcomes of a random experiment. The `prob` package has several common sample spaces built in. For example, coin tosses:

```
> tosscoin(1)
```

```
  toss1
1     H
2     T
```

The argument of `tosscoin()` tells how many tosses to include. If we toss a coin 3 times, the sample space consists of 8 possible outcomes:

```
> tosscoin(3)
```

```
  toss1 toss2 toss3
1     H     H     H
2     T     H     H
3     H     T     H
4     T     T     H
5     H     H     T
6     T     H     T
7     H     T     T
8     T     T     T
```

Similarly, we can specify the number of rolls of a die. The shape of the die can be specified using the `nsides` keyword, as in the following example which shows the sample space for 2 rolls of a 4-sided die:

```
> rolldie(2, nsides=4)
```

```
   X1 X2
1   1  1
2   2  1
3   3  1
4   4  1
5   1  2
6   2  2
7   3  2
8   4  2
9   1  3
10  2  3
11  3  3
12  4  3
13  1  4
14  2  4
15  3  4
16  4  4
```

Other built in sample spaces include `cards()`, `roulette()`

3.1 urnsamples()

Sampling from urns is perhaps the most fundamental type of random experiment. We have an urn that contains a bunch of distinguishable objects (balls) inside. We shake up the urn, reach inside, grab a ball, and take a look. That's all.

Note that this one general class of random experiments contains as a special case all of the common elementary random experiments. Tossing a coin twice is equivalent to selecting two balls labeled H and T from an urn, with replacement (meaning we put the ball back in the urn after recording it). The die-roll experiment is equivalent to selecting a ball from an urn with six elements, labeled 1 through 6. We can use `urnsamples()` to create any sample space for this type of experiment.

The `urnsamples()` function lets us specify the following arguments: `x, size, replace, ordered`.

- `x` specifies the distinguishable objects in the urn,
- `size` tells how many times we pick an object,
- `replace` specifies whether we put the object back into the urn before the next sample.
- `ordered` specifies whether we keep track of the order that the samples were drawn.

We can define the sample space for tossing a coin 3 times as above with the `urnsamples()` function. We define `replace` as TRUE, because we can't take away one side of the coin after a toss. `ordered` is TRUE as well since we are interested in the outcome of 3 sequential tosses:

```
> urnsamples(x=c('H', 'T'), size=3, replace=TRUE, ordered=TRUE)

  X1 X2 X3
1  H  H  H
2  T  H  H
3  H  T  H
4  T  T  H
5  H  H  T
6  T  H  T
7  H  T  T
8  T  T  T
```

Exercise: use `urnsamples()` to create the sample space for rolling a 20-sided die 2 times. *Hint: you can provide a sequence as the* `x` *argument for* `urnsamples()`*, such as* `1:6`.

4 Events

An event is a collection of outcomes, or in other words, a subset of the sample space. In `R`, a subset is created with the `subset()` function. The first argument is the sample space, and the second argument is an indexing statement.

```
> 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
```

```
> subset(S, rank %in% 7:9)
```

```
   rank    suit
6     7    Club
7     8    Club
8     9    Club
19    7 Diamond
20    8 Diamond
21    9 Diamond
32    7   Heart
33    8   Heart
```

```
34    9   Heart
45    7   Spade
46    8   Spade
47    9   Spade
```

```
> 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
```

4.1 %in%

The function `%in%` tells whether each value of one vector lies somewhere inside another vector.

```
> x <- 1:10
> y <- 8:12
> y %in% x
```

```
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

Notice that the returned value is a vector of length 5 which tests whether each element of y is in x, in turn.

4.2 isin()

It is more common to want to know whether the whole vector y is in x, in other words if y is a subset of x. We can do this with the `isin()` function.

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

```
[1] FALSE
```

We can use `isin()` to find patterns in sample spaces. For example, if we want to see how many times the sequence 2, 2, 6 occurs when rolling a die 4 times, we can do the following:

```
> S <- rolldie(4)
> subset(S, isin(S, c(2, 2, 6), ordered=TRUE))
```

Exercise: Assume the following sets of integers:

```
> x <- 1:10
> y <- c(3,3,7)
```

Compare the results of `all(y %in%x)` vs. `isin(x, y)`. Why are the answers different?

```
> all(y %in%x)
[1] TRUE
> isin(x,y)
[1] FALSE
```

4.3 Set Algebra

`R` also has functions for union, intersection, and difference:

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

- `union(A, B)`: In A or B or both
- `intersect(A, B)`: In both A and B
- `setdiff(A, B)`: in A but not in B
- `setdiff(B, A)`: in B but not in A

Exercise: Find $(A \cup B)^c$

5 Assigning probabilities

We use the function `probspace()` to assign probabilities to a sample space. If we have a fair six-sided die, each outcome has probability of $\frac{1}{6}$:

```
> outcomes <- rolldie(1)
> (p <- rep(1/6, times=6))

[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667

> probspace(outcomes, probs = p)

  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
```

The `probspace()` function will by default assign equal probability to each outcome in the sample space, so the above code can be simplified to:

```
> probspace(1:6)

  x     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
```

Similarly, the built-in sample space methods have an optional argument `makespace` that will assign equal probability to each outcome:

```
> rolldie(1, 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
```

Exercise: Use `probspace()` to assign probabilities to the sample space of a biased coin, where $P(\text{Heads}) = 0.7$ and $P(\text{Tails}) = 0.3$.

5.1 Probabilities of events

Now we can use R to calculate the probabilities of certain events. For example, to calculate the probability of drawing a card with rank 2, we do the following:

```
> S <- cards(makespace=TRUE)
> (A <- subset(S, rank==2))
```

```
   rank    suit      probs
1     2    Club 0.01923077
14    2 Diamond 0.01923077
27    2   Heart 0.01923077
40    2   Spade 0.01923077
```

```
> prob(A)
```

```
[1] 0.07692308
```

```
> #This corresponds to 4/52
> 4/52
```

```
[1] 0.07692308
```

Exercise: Find the probability of the sum of three rolls of a 6-sided die being greater than 16, i.e. $(X1 + X2 + X3 > 16)$

References

[1] G. Jay Kerns, *Introduction to Probability and Statistics Using* R. 1st Edition, 2010.