Introduction to Statistical Thought

Michael Lavine

December 18, 2006

Contents

List of Figures

List of Tables

Preface

This book is intended as an upper level undergraduate or introductory graduate textbook in statistical thinking with a likelihood emphasis for students with a good knowledge of calculus and the ability to think abstractly. By "statistical thinking" is meant a focus on ideas that statisticians care about as opposed to technical details of how to put those ideas into practice. By "likelihood emphasis" is meant that the likelihood function and likelihood principle are unifying ideas throughout the text. Another unusual aspect is the use of statistical software as a pedagogical tool. That is, instead of viewing the computer merely as a convenient and accurate calculating device, we use computer calculation and simulation as another way of explaining and helping readers understand the underlying concepts.

Our software of choice is `R`. `R` and accompanying manuals are available for free download from `http://www.r-project.org`. You may wish to download **An Introduction to R** to keep as a reference. It is highly recommended that you try all the examples in `R`. They will help you understand concepts, give you a little programming experience, and give you facility with a very flexible statistical software package. And don't just try the examples as written. Vary them a little; play around with them; experiment. You won't hurt anything and you'll learn a lot.

Chapter 1

Probability

1.1 Basic Probability

Let $\mathcal{X}$ be a set and $\mathcal{F}$ a collection of subsets of $\mathcal{X}$. A *probability measure*, or just a *probability*, on $(\mathcal{X}, \mathcal{F})$ is a function $\mu : \mathcal{F} \to [0, 1]$. In other words, to every set in $\mathcal{F}$, μ assigns a probability between 0 and 1. We call μ a *set function* because its domain is a collection of sets. But not just any set function will do. To be a probability μ must satisfy

1. $\mu(\emptyset) = 0$ ($\emptyset$ is the empty set.),

2. $\mu(\mathcal{X}) = 1$, and

3. if A_1 and A_2 are disjoint then $\mu(A_1 \cup A_2) = \mu(A_1) + \mu(A_2)$.

One can show that property 3 holds for any finite collection of disjoint sets, not just two; see Exercise 1. It is common practice, which we adopt in this text, to assume more — that property 3 also holds for any countable collection of disjoint sets.

When $\mathcal{X}$ is a finite or countably infinite set (usually integers) then μ is said to be a *discrete* probability. When $\mathcal{X}$ is an interval, either finite or infinite, then μ is said to be a *continuous* probability. In the discrete case, $\mathcal{F}$ usually contains all possible subsets of $\mathcal{X}$. But in the continuous case, technical complications prohibit $\mathcal{F}$ from containing all possible subsets of $\mathcal{X}$. See Casella and Berger [2002] or Schervish [1995] for details. In this text we deemphasize the role of $\mathcal{F}$ and speak of probability measures on $\mathcal{X}$ without mentioning $\mathcal{F}$.

In practical examples $\mathcal{X}$ is the set of outcomes of an "experiment" and μ is determined by experience, logic or judgement. For example, consider rolling a six-sided die. The set of outcomes is $\{1, 2, 3, 4, 5, 6\}$ so we would assign $\mathcal{X} \equiv \{1, 2, 3, 4, 5, 6\}$. If we believe the die to be fair then we would also assign $\mu(\{1\}) = \mu(\{2\}) = \cdots = \mu(\{6\}) = 1/6$. The laws of probability then imply various other values such as

$$\begin{aligned} \mu(\{1, 2\}) &= 1/3 \\ \mu(\{2, 4, 6\}) &= 1/2 \\ &\text{etc.} \end{aligned}$$

Often we omit the braces and write $\mu(2)$, $\mu(5)$, etc. Setting $\mu(i) = 1/6$ is not automatic simply because a die has six faces. We set $\mu(i) = 1/6$ because we believe the die to be fair.

We usually use the word "probability" or the symbol P in place of μ. For example, we would use the following phrases interchangeably:

- The probability that the die lands 1
- P(1)
- P[the die lands 1]
- $\mu(\{1\})$

We also use the word *distribution* in place of *probability measure*.

The next example illustrates how probabilities of complicated events can be calculated from probabilities of simple events.

Example 1.1 (The Game of Craps)
Craps is a gambling game played with two dice. Here are the rules, as explained on the website `www.online-craps-gambling.com/craps-rules.html`.

> For the dice thrower (shooter) the object of the game is to throw a 7 or an 11 on the first roll (a win) and avoid throwing a 2, 3 or 12 (a loss). If none of these numbers (2, 3, 7, 11 or 12) is thrown on the first throw (the Come-out roll) then a Point is established (the point is the number rolled) against which the shooter plays. The shooter continues to throw until one of two numbers is thrown, the Point number or a Seven. If the shooter rolls the Point before rolling a Seven he/she wins, however if the shooter throws a Seven before rolling the Point he/she loses.

Ultimately we would like to calculate P(shooter wins). But for now, let's just calculate

$$\mathrm{P}(\text{shooter wins on Come-out roll}) = \mathrm{P}(7 \text{ or } 11) = \mathrm{P}(7) + \mathrm{P}(11).$$

Using the language of page 1, what is $\mathcal{X}$ in this case? Let d_1 denote the number showing on the first die and d_2 denote the number showing on the second die. d_1 and d_2 are integers from 1 to 6. So $\mathcal{X}$ is the set of ordered pairs (d_1, d_2) or

$$\begin{array}{cccccc}
(6,6) & (6,5) & (6,4) & (6,3) & (6,2) & (6,1) \\
(5,6) & (5,5) & (5,4) & (5,3) & (5,2) & (5,1) \\
(4,6) & (4,5) & (4,4) & (4,3) & (4,2) & (4,1) \\
(3,6) & (3,5) & (3,4) & (3,3) & (3,2) & (3,1) \\
(2,6) & (2,5) & (2,4) & (2,3) & (2,2) & (2,1) \\
(1,6) & (1,5) & (1,4) & (1,3) & (1,2) & (1,1)
\end{array}$$

If the dice are fair, then the pairs are all equally likely. Since there are 36 of them, we assign $\mathrm{P}(d_1, d_2) = 1/36$ for any combination (d_1, d_2). Finally, we can calculate

$$\begin{aligned}
\mathrm{P}(7 \text{ or } 11) &= \mathrm{P}(6,5) + \mathrm{P}(5,6) + \mathrm{P}(6,1) + \mathrm{P}(5,2) \\
&+ \mathrm{P}(4,3) + \mathrm{P}(3,4) + \mathrm{P}(2,5) + \mathrm{P}(1,6) = 8/36 = 2/9.
\end{aligned}$$

The previous calculation uses desideratum 3 for probability measures. The different pairs $(6,5)$, $(5,6)$, $\ldots$, $(1,6)$ are disjoint, so the probability of their union is the sum of their probabilities.

Example 1.1 illustrates a common situation. We know the probabilities of some simple events like the rolls of individual dice, and want to calculate the probabilities of more complicated events like the success of a Come-out roll. Sometimes those probabilities can be calculated mathematically as in the example. Other times it is more convenient to calculate them by computer simulation. We frequently use `R` to calculate probabilities. To illustrate, Example 1.2 uses `R` to calculate by simulation the same probability we found directly in Example 1.1.

Example 1.2 (Craps, continued)
To simulate the game of craps, we will have to simulate rolling dice. That's like randomly sampling an integer from 1 to 6. The `sample()` command in `R` can do that. For example, the following snippet of code generates one roll from a fair, six-sided die and shows `R`'s response:

```
> sample(1:6,1)
[1] 1
>
```

When you start R on your computer, you see >, R's prompt. Then you can type a command such as `sample(1:6,1)` which means "take a sample of size 1 from the numbers 1 through 6". (It could have been abbreviated `sample(6,1)`.) R responds with `[1] 1`. The `[1]` says how many calculations R has done; you can ignore it. The `1` is R's answer to the `sample` command; it selected the number "1". Then it gave another >, showing that it's ready for another command. Try this several times; you shouldn't get "1" every time.

Here's a longer snippet that does something more useful.

```
> x <- sample ( 6, 10, replace=T ) # take a sample of
                                   # size 10 and call it x
> x # print the ten values
[1] 6 4 2 3 4 4 3 6 6 2

> sum ( x == 3 ) # how many are equal to 3?
[1] 2
>
```

Note

- `#` is the comment character. On each line, R ignores all text after `#`.
- We have to tell R to take its sample *with replacement*. Otherwise, when R selects "6" the first time, "6" is no longer available to be sampled a second time. In `replace=T`, the `T` stands for `True`.
- `<-` does *assignment*. I.e., the result of `sample ( 6, 10, replace=T )` is assigned to a variable called `x`. The assignment symbol is two characters: < followed by `-`.
- A variable such as `x` can hold many values simultaneously. When it does, it's called a *vector*. You can refer to individual elements of a vector. For example, `x[1]` is the first element of `x`. `x[1]` turned out to be `6`; `x[2]` turned out to be `4`; and so on.

- == does *comparison*. In the snippet above, (x==3) checks, for each element of `x`, whether that element is equal to 3. If you just type `x == 3` you will see a string of `T`'s and `F`'s (True and False), one for each element of `x`. Try it.

- The `sum` command treats `T` as `1` and `F` as 0.

- `R` is almost always tolerant of spaces. You can often leave them out or add extras where you like.

On average, we expect 1/6 of the draws to equal 1, another 1/6 to equal 2, and so on. The following snippet is a quick demonstration. We simulate 6000 rolls of a die and expect about 1000 1's, 1000 2's, etc. We count how many we actually get. This snippet also introduces the `for` loop, which you should try to understand now because it will be *extremely* useful in the future.

```
> x <- sample(6,6000,replace=T)



> for ( i in 1:6 ) print ( sum ( x==i ))
[1] 995
[1] 1047
[1] 986
[1] 1033
[1] 975
[1] 964
>
```

Each number from 1 through 6 was chosen about 1000 times, plus or minus a little bit due to chance variation.

Now let's get back to craps. We want to simulate a large number of games, say 1000. For each game, we record either 1 or 0, according to whether the shooter wins on the Come-out roll, or not. We should print out the number of wins at the end. So we start with a code snippet like this:

```
# make a vector of length 1000, filled with 0's
     wins <- rep ( 0, 1000 )
     for ( i in 1:1000 ) {
```

```
  simulate a Come-out roll
   if shooter wins on Come-out, wins[i] <- 1
}

sum ( wins ) # print the number of wins
```

Now we have to figure out how to simulate the Come-out roll and decide whether the shooter wins. Clearly, we begin by simulating the roll of two dice. So our snippet expands to

```
# make a vector of length 1000, filled with 0's
          wins <- rep ( 0, 1000 )
for ( i in 1:1000 ) {
  d <- sample ( 1:6, 2, replace=T )
         if ( sum(d) == 7 || sum(d) == 11 ) wins[i] <- 1
}
sum ( wins ) # print the number of wins
```

The "`||`" stands for "or". So that line of code sets `wins[i] <- 1` if the sum of the rolls is either 7 or 11. When I ran this simulation `R` printed out 219. The calculation in Example 1.1 says we should expect around $(2/9) \times 1000 \approx 222$ wins. Our calculation and simulation agree about as well as can be expected from a simulation. Try it yourself a few times. You shouldn't always get 219. But you should get around 222 plus or minus a little bit due to the randomness of the simulation.

Try out these `R` commands in the version of `R` installed on your computer. Make sure you understand them. If you don't, print out the results. Try variations. Try any tricks you can think of to help you learn `R`.

1.2 Probability Densities

So far we have dealt with *discrete* probabilities, or the probabilities of at most a countably infinite number of outcomes. For discrete probabilities, $\mathcal{X}$ is usually a set of integers, either finite or infinite. Section 1.2 deals with the case where $\mathcal{X}$ is an interval, either of finite or infinite length. Some examples are