Post 01: Intro to Probability in R

Stat 133, Fall 2017

Woo Sik (Lewis) Kim

Introduction

This post will provide you with a very basic introduction to probability and its intersections in R. We'll talk about 3 main things (technically 4): Visualization of Probability, Computing Probability with Data, and Distributions.

Hopefully, after going through this post, you'll have a newfound appreciation of probability, and realize how intuitive and simple it is to compute them with R.

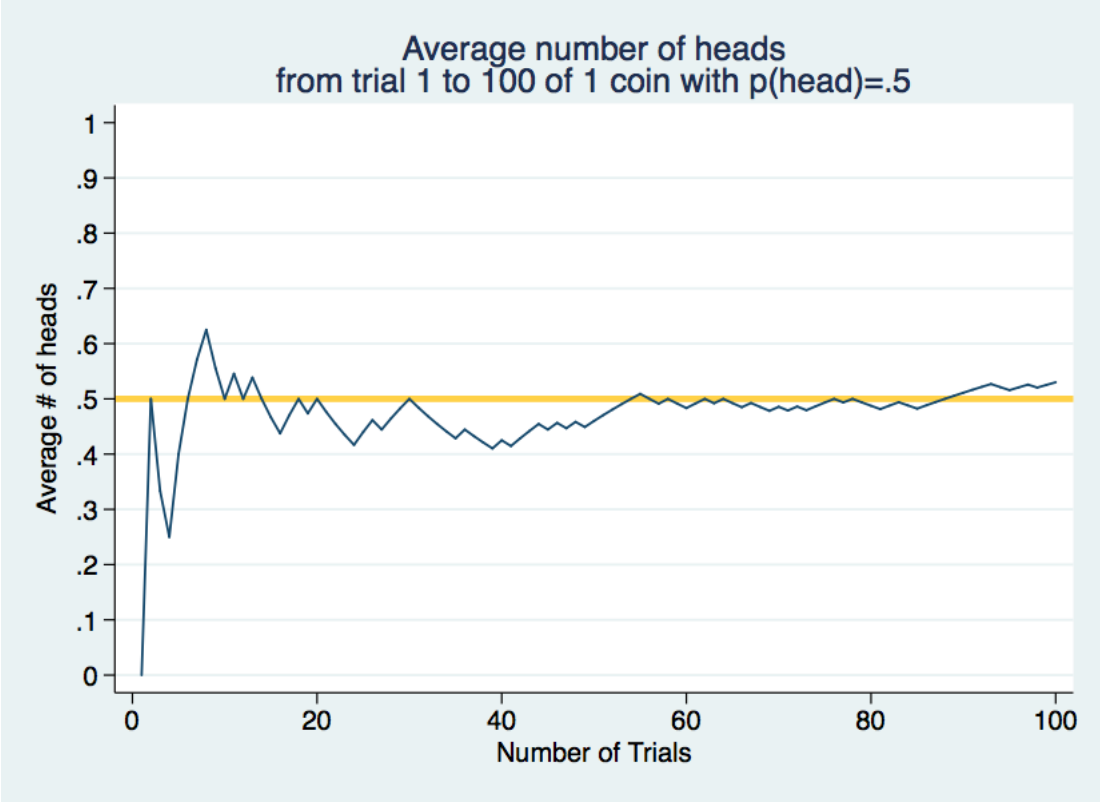
Visualizing Probability in R: The Law of Large Numbers

One great thing about the advent of computers is that they allow us to visualize various mathematical properties in a clear, concise way, in ways that wasn't done before. Even One such property that's very important in both Statistics and Mathematics is the Law of Large Numbers

So, what is the Law of Large Numbers? In Probability Theory, the Law of Large Numbers, or "LLN", is a theory that states the average of the results obtained from a large number of trials in an experiment should be close to the expected value, and will become closer and closer to it as more trials are performed. So, what does this mean? It means that as our number of trials approach infinity, our average will become closer and closer to the true probability, or expected value.

For instance, let's say we're flipping a fair, unbiased coin. What is our expected number of heads in, say, 2 coin flips? Since the probability of getting heads is 50%, our expected value should be 1 heads. But, even though 1 is our expected value, this may not be the case in reality. We could get 0 heads, or 2 heads. What if we flip 10 coins? Even though our expectation is 5 heads, we could get something much less, or much more. So, if we flip 10,000, 100,000, or even 1,000,000 coins, what's to say our observed number of heads is much more or less than the expected value of 50%? This is where LLN comes in. As we increase our number of trials, our observed value will approach the expected value.

With R, we're actually able to visualize this phenomenon:



As you can see, as we increase our number of trials, our number of heads approach its expected value: 50%.

Computing Probabilities from Data

At the core of probabiltity, our values come from data. If we want the probability of event A happening, we need to count the number of times we observe event A, and divide by the total number of possibilities. So, for instance, if we want to calculate the probability a player scores, we count the number of times that player has successfully made a shot and divide it by the total number of shots he's made, both successfully and not. Let's apply this concept to the NBA data we've been using throughout the course.

Using the NBA .csv file, let's calculate some basic probabilities using functions, both given and user-written.

First, we import the NBA player statistics data:

```
require(readr)
```

```
## Loading required package: readr
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
dataset <- read_csv('data/nba2017-player-statistics.csv',
                    col_types =
                      list(
                        Player = col_character(),
                        Team = col_character(),
                        Position = col_factor(c('C', 'PF','PG','SF','SG')),
                        Experience = col_character(),
                        Salary = col_double(),
                        Rank = col_integer(),
                        Age = col_integer(),
                        GP = col_integer(),
                        GS = col_integer(),
                        MIN = col_integer(),
                        FGM = col_integer(),
                        FGA = col_integer(),
                        Points3 = col_integer(),
                        Points3_atts = col_integer(),
                        Points2 = col_integer(),
                        Points2_atts = col_integer(),
                        FTM = col_integer(),
                        FTA = col_integer(),
                        OREB = col_integer(),
                        DREB = col_integer(),
                        AST = col_integer(),
                        STL = col_integer(),
                        BLK = col_integer(),
                        TO = col_integer()
                        ))
```

Now, let's pick out the players' salaries and count the number of entries using sum():

```
salaries <- dataset$Salary #select the Salary column from our NBA dataset.
total <- sum(salaries > 0) # Returns the number of elements whose value is greater than 0; in other words, it coun
ts the number of elements in 'salaries.'
total
```

```
## [1] 441
```

We can see that the total number of salary entries in our NBA player statistics dataset is 441. Using this information, and our salaries data frame, let's write a basic function using a for loop that tells us what the probability of a player making over X dollars is (since we're not using salary buckets, our calculated probabilities are very rough estimates; they're probably not close to the true probabilities. But, for the purposes of this demonstration, this function will do).

```
salaryprob <- function(x) {
  count <- 0
  for (s in salaries) {
    if (s > x) {
      count <- count + 1
    }
  }
  return (count/total)
}
```

So, what is the probability that some player, Ben Dover, makes more than $1,000,000?

```
salaryprob(1000000)
```

```
## [1] 0.814059
```

Our rough estimate is about 81.4%.

Using our 'salaryprob()' function as pseudocode, we can write a general, do-it-all function to calculate probabilities for any numerical type of data from our NBA dataset. Let's create a 'generalProb()' function that takes in 3 arguments: "x", the number we want to compare all our values from

our selected data (type numeric), “colName,” the column we want to select from our NBA data (type character), and “args,” which takes on values of >, =, <, >=, or <= (type character).

```
generalProb <- function(colName, arg, x) {
  col_data <- dataset[[colName]] # Select our column from the NBA dataset.
  total_num <- sum(col_data > 0) # Count the number of entries in our column.
  count <- 0
  if (arg == '>') { # Calculate the probability our values are > x.
    for (val in col_data) {
      if (val > x) {
        count <- count + 1
      }
    }
    return (count/total_num)
  }
  else if (arg == '=') { # Calculate the probability our values are == x.
    for (val in col_data) {
      if (x == val) {
        count <- count + 1
      }
    }
    return (count/total_num)
  }
  else if (arg == '<') { # Calculate the probability our values are < x.
    for (val in col_data) {
      if (val < x) {
        count <- count + 1
      }
    }
    return (count/total_num)
  }
  else if (arg == '<=') { # Calculate the probability our values are <= x.
      for (val in col_data) {
      if (val <= x) {
        count <- count + 1
      }
    }
    return (count/total_num)
  }
  else if (arg == '>=') { # Calculate the probability our values are >= x.
    for (val in col_data) {
      if (val >= x) {
        count <- count + 1
      }
    }
    return (count/total_num)
  }
  else {
    return ('Error: invalid args')
  }
}
```

Notice that our ‘generalProb()’ function ONLY accounts for numeric types of data. If we select our column to be any other type, the function will return an error. Now, let’s use our new function to see what the chance of a player being at or over the age of 30 is.

```
generalProb('Age', '>=', 30)
```

```
## [1] 0.2312925
```

It looks like it’s about 23.1%.

How about the probability that a player has less than 10 games played from the season?

```
generalProb('GP', '<', 10)
```

```
## [1] 0.06122449
```

According to our function, it’s about 6%.

What about the probability that a player made at most 100 free throws?

```
generalProb('FTM', '<=', 100)
```

```
## [1] 0.7076566
```

It looks like it’s around 70%.

As you can see, with R, it’s very easy to compute probabilities for many different types of data (albeit rough estimates for numeric data in this demonstration for the sake of simplicity). These probabilities can be very basic, like the ones we’ve computed, or it can be much more advanced, such as using various probability distributions, or even something like using Time Series Analysis (data that has a “timestamp” attached to its values, such as the variations in stock value during a trading day. Take Stat 153 to learn more!). Nevertheless, R is a powerful tool that helps us analyze the likelihoods of our values and gives us insight into what kind of data it is.

Intro to Distributions: The Binomial Distribution

Not only can we compute basic probabilities in R, but we can also compute probabilities of data that follow certain distributions. So, what is a distribution? Simply put, a distribution is some function that provides us with the probabilities of occurences of different possible outcomes in an experiment.

Distributions can be split up into 2 different parts: discrete and continuous. Discrete distributions mean that our outcomes must be countable ("in whole numbers"); they are discrete values. For example, flipping 10 coins is a discrete distribution. Whether we get 1, 2, or 10 heads, our resulting number heads must be countable. We can't have something like 2.5 heads, or 5.7 heads; it doesn't make any sense.

The discrete distribution we'll talk about today is quite simple, but also very useful: the binomial distribution. The binomial distribution results when we have "n" independent trials, with each trial having only 2 outcomes: "successes," a.k.a. events we want, and "failures," whenever a success does not occur. Each success can occur with probability "p", while our failures occur with probability "1 - p". For example, let's say we're rolling a dice 10 times, and want to roll 6's. Each dice roll is independent; one roll does not affect another. Whenever we roll a 6, we call it a success; this happens with probability 1/6. Whenever we roll a 1, 2, 3, 4, or 5, we call it a failure, which occurs with probability 5/6 (which is also equal to 1 - 1/6 = 5/6).

The most crucial thing for events to have a binomial distribution is that each trial MUST be independent, and each trial MUST have the same probability of success, "p." If it fails to satisfy these 2 requirements, then it cannot be a binomial distribution.

The binomial distribution follows the following formula:

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

In R, this formula is equivalent to 'dbinom(k, n, p),' where "k" is the number of successes, "n" is the number of trials, and "p" is the probability of success.

Using this, let's apply it to the coin toss experiment we talked about with LLN. Let's say I toss 10 fair coins. What's the probability that I get exactly 5 heads? Here, "k" is 5, since that's the number of "successes" we want: 5 heads. "n" is the total number of trials, which is 10. Finally, "p" is our probability of success, or the probability of getting heads, which is 0.5.

```
# Probability of getting exactly 5 heads in 10 coin tosses (fair coin)
prob <- dbinom(5, 10, 0.5)
prob
```

```
## [1] 0.2460938
```

Here, we can see that the probability of getting exactly 5 heads in 10 coin tosses is approximately 24.6%.

Now, can we apply this to our NBA data? We could simply plug-and-chug, but the results wouldn't make any sense. This is because our NBA data gives us player statistics, and doesn't involve independent trials with each trial having the same chance of success. Applying the binomial distribution to NBA player statistics would be like trying to use a pillow as a calculator, or trying to put anchovies on a pizza. It just won't make any sense.

We can use the binomial formula to solve much more exciting and complicated problems. For example, let's look at some sporting event. We have 2 teams playing: the Cats and the Dogs. Let's say that they are playing 20 games, and the Dogs have a 30% chance of winning each game. Each time the Dogs win, you make $100. What is the probability that you can make at least $1000 from this event? In other words, what is the chance that the Dogs win at least 10 games?

This problem uses a summation of the binomial formula:

$$\sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$$

This formula involves a sum because we need to account for the cases where they win more than 10 games. Sure, if they win 10 games, we're done, but we want to know what the chances of them winning AT LEAST 10 games is. It's like asking, what's the chance that they win exactly 10 games, or exactly 11 games, or exactly 12 games, … , or exactly 20 games? To compute this in R, we use '1 - pbinom(k-1, n, p).' (We use 1 - pbinom(k-1, n, p) because pbinom(k-1, n, p) tells us the probability of at most k-1 successes out of n trials is; so, to find the chance of success on at least k trials, we take the complement of the chance of at most k-1 successes through 1 - pbinom(k-1, n, p). If you don't understand this, don't worry about it. You'll learn a lot more on binomial distributions in Stat 134).

```
1 - pbinom(9, 20, 0.3)
```

```
## [1] 0.0479619
```

It looks like you have about a 4.79% chance of making $1000 from this event.

Just as we wrote a "general" function in our previous section on computing basic probabilities with our NBA dataset, let's write a "general" function for binomial probabilities, 'generalBin().' Let's write the function (and its order of variable inputs) to as closely resemble the sentence "exactly/at least/at most 'k' successes out of 'n' trials with success rate 'p'."

```r
generalBin <- function(arg, k, n, p) { # args takes in whether we want exactly/at least/at most; k is the number of successes, n is the number of trials, and p is the chance of success.
  if (arg == 'exactly') {
    return (dbinom(k, n, p))
  }
  else if (arg == 'at least') {
    return (1 - pbinom(k-1, n, p))
  }
  else if (arg == 'at most') {
    return (pbinom(k, n, p))
  }
  else {
    return ("Error: invalid arg")
  }
}
```

Let's apply our new general binomial function. In a class of 20 people, what is the probability that exactly 2 students will get an A if historically about 10% of the class received A's (independently)?

```r
generalBin('exactly', 2, 20, 0.1)
```

```
## [1] 0.2851798
```

How about the probability that at least 5 students will get A's? (So, 5 A's, or 6 A's, or 7 A's, …. , or 20 A's)

```r
generalBin('at least', 5, 20, 0.1)
```

```
## [1] 0.0431745
```

How about the probability that at most 5 students get an A? (So, 0 A's, or 1 A's, or 2 A's, … , or 5 A's)

```r
generalBin('at most', 5, 20, 0.1)
```

```
## [1] 0.9887469
```

Before computers, these probabilities were sometimes quite hard and very time-consuming to solve. However, now, with computers and languages like R, it's as simple as plugging the necessary values into a function.

Continuous Distributions

On the flip side of discrete distributions, we have continuous distributions. Unlike discrete distributions, whose values must be countable (i.e. "in whole numbers"), a continuous distribution can take on any real value. The best example for continuous distribution is time. Sure, we can have "whole" seconds, but we can also have 1.2, or 5.4 seconds.

Because continuous distributions can take on any real value between 'a' and 'b', it's a bit more complicated to compute continuous probabilities compared to discrete probabilities. Computing continuous probabilities involve everyone's favorite: computing integrals!

Here's the general formula for calculating continuous probability between 'a' and 'b' (called the "Probability Density Function"):

Probability Density Function

$$F(x) = P(a \leq x \leq b) = \int_a^b f(x)dx \geq 0$$

Computing integrals, and consequently, computing probability density functions, is quite easy using R. For the sake of simplicity, let's work with 1 / x^2 as our function for some continuous distribution.

First, we define our integrand (or, the function):

```r
integrand <- function(x) {
  1/x^2
}
```

Now, let's say we want to find the probability of a value being between 2 and 4. Then, we integrate using the 'integrate()' function in R:

```r
integrate(integrand, lower = 2, upper = 4)
```

```
## 0.25 with absolute error < 2.8e-15
```

It looks like it's about 25%. How about from 0 to infinity?

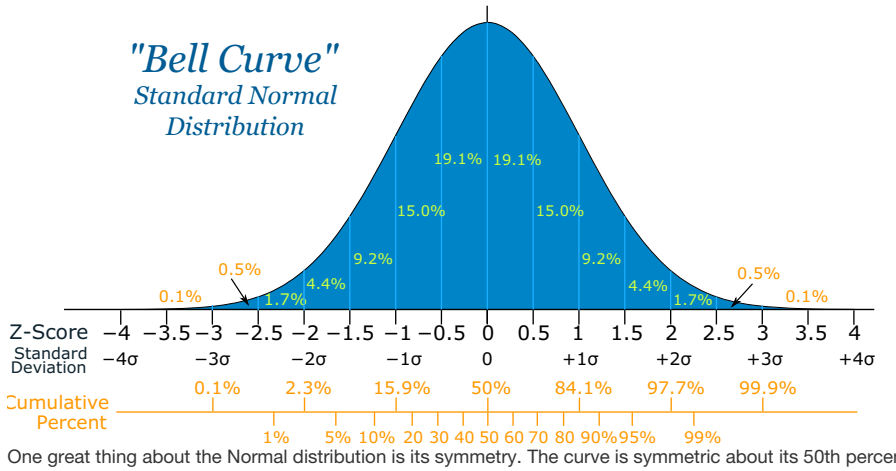
```
integrate(integrand, lower = 0, upper = Inf)
```

This is a trick question. The integral from 0 to infinity for our integrand is actually divergent, and R will tell you right away with this message:

```
Line 280 Error in integrate(integrand, lower = 0, upper = Inf) : the integral is
         probably divergent Calls: <Anonymous> ... withCallingHandlers -> withVisible -
         > eval -> eval -> integrate Execution halted
```

As you can see, the entire process of computing probability density functions, which could involve some very time-consuming calculations for much more complicated functions, can be done in 2 lines of code using R.

Outside of continuous distributions that we can define ourselves with different probability density functions, there are “pre-determined” ones as well. Arguably the most well-known continuous distribution is the Normal distribution (also known as the “normal curve” or “bell curve” among students). This is what the Normal distribution looks like:



One great thing about the Normal distribution is its symmetry. The curve is symmetric about its 50th percentile, so it makes computing probabilities much more elegant.

Here’s the Probability Density Function for Normal distributions :

Normal Probability Density Function

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$$

Luckily, we don’t have to worry about writing a function that for this messy-looking equation. It’s already built into R. To “plug-in” values into the probability density function for Normal distributions, we use ‘dnorm(x, mean = m, sd = s), where ’x’ is the argument of the function, ‘mean’ is the mean of the distribution, and ‘sd’ is the standard deviation of the distribution.

```
dnorm(2, 1, 1) # Calculates f(2) for the Normal probability density function with mean = 1 and standard deviation
= 1.
```

```
## [1] 0.2419707
```

What if we want to find the probability of everything below or above ‘x’? i.e., the integral from negative infinity to ‘x’, or from ‘x’ to infinity?

For negative infinity to ‘x’, a.k.a. lower tails, we use ‘pnorm(x, mean, SD, lower.tail = TRUE)’. Similarly, for ‘x’ to infinity, a.k.a. upper tails, we use ‘pnorm(x, mean, SD, lower.tail = FALSE)’.

Let’s apply these concepts to every student’s favorite: standardized exams. Let’s say some college entrance exam, FAT, has a mean of 70 and SD of 15. Let’s say John scored an 85 on the exam. How well did John do? First, let’s find the % of students who scored worse than John using the lower tail function.

```
pnorm(85, 70, 15, lower.tail = TRUE) # % of students who scored worse than John.
```

```
## [1] 0.8413447
```

It looks like John did better than 84.1% of his fellow students. How about students who did better than John? using the upper tail function, we get:

```
pnorm(85, 70, 15, lower.tail = FALSE) # % of students who scored better than John.
```

```
## [1] 0.1586553
```

It seems about 15.9% of students did better than John.

As you can see, R makes what were/are complicated calculations very intuitive, and easy to solve, with simply a few lines of code.

There are many, many more different types of continous distributions, and there are also numerous fascinating properties that go along with continuous distributions/probability. But for now, this section serves as a good introduction to what continuous distributions are and how to compute them in R.

Take-Home Message

Probability is a very important part of statistics. It provides us with insights about our data, how likely something is/is not going to happen, information about outliers, how dense/spread out our data is (e.g. Probability Density Function), and much more.

Not only can we write various functions ourselves to analyze our data, like we've seen with the NBA playe statistics dataset, we can also use built-in functions, like dbinom() or pbinom(), to quickly compute and analyze what were once difficult equations to solve.

With R, computing probabilities and solving various problems is extremely simple; even though equations/distributions/density functions may look intimidating, once you get the hang of it, most functions you can write to compute these probabilities follow a general form. R also allows us to visualize very important mathematical properties like the Law of Large Numbers so that they become easier to understand.

I hope this post was a good introduction to probability theory and its intersections with R!

Sources

Definitions:

• Law of Large Numbers: http://whatis.techtarget.com/definition/law-of-large-numbers

• Binomial Distribution:http://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm

• Discrete and Continuous Distributions: http://stattrek.com/probability-distributions/discrete-continuous.aspx

• Normal Distribution: https://en.wikipedia.org/wiki/Normal_distribution

R Functions and Methods:

• Binomial Formula: http://www.r-tutor.com/elementary-statistics/probability-distributions/binomial-distribution

• Numeric Integrals in R: http://homepages.math.uic.edu/~jyang06/stat522/handouts/handout8.pdf

• Normal Distribution in R: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Normal.html

Misc. Debugging:

• Prof. Gaston Sanchez

• Stack Overflow: https://stackoverflow.com/questions/23316161/the-condition-has-length-1-and-only-the-first-element-will-be-used