R Users Guide to Stat 201: Chapter 5

Michael Shyne, 2017

Chapter 5: Discrete Probability Distributions

In Chapter 5 we are introduced to probability distributions. R has many built-in functions which make it easy to work with many common distributions.

Probability Distributions

Before we work with the standard distributions, we need to handle some arbitrary probability distributions. As discussed in a previous guide, data in MyStatLab can be exported to Excel, saved as a `csv` file and then imported into R as a data frame. I've recreated such a table by hand below, so this example code can be used without an additional data file.

```
prob.dist <- data.frame(x=0:5, P.x.=c(0.03, 0.13, 0.25, 0.34, 0.16, 0.09))
prob.dist
```

```
##   x P.x.
## 1 0 0.03
## 2 1 0.13
## 3 2 0.25
## 4 3 0.34
## 5 4 0.16
## 6 5 0.09
```

The first step is to determine if this is really a probability distribution, if the probabilities add to 1.

```
sum(prob.dist$P.x.)
```

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

Now that we've verified that we are indeed working with a true distribution, we can calculate mean and standard deviation. There are no R functions expressly for this purpose. However, as we noted that the mean of a distribution is merely the weighted mean of the values with the probabilities as weights, R does have a function for that.

```
pd.mean <- weighted.mean(prob.dist$x, prob.dist$P.x.)
pd.mean
```

```
## [1] 2.74
```

Standard deviation will take a little more effort. Variance is the weighted mean of the difference from the mean squared, again with probabilities as weights. And standard deviation is the squared root of variance.

```
# Find variance
pd.var <- weighted.mean((prob.dist$x - pd.mean)^2, prob.dist$P.x.)
pd.var
```

```
## [1] 1.4924
```

```
# SD is square root of variance
pd.sd <- sqrt(pd.var)
pd.sd
```

```
## [1] 1.221638
```

To find probabilities of compound or complex events, for small distribution tables like this one, it is probably easiest to merely add to relevant probabilities. For larger tables or data sets, we will want to be able to specify subsets conditionally.

Recall for the Chapter 1 guide, we can subset vectors or data frames by providing index numbers.

```
# Display the first, third and fourth rows, all columns
prob.dist[c(1,3,4),]
```

```
##   x P.x.
## 1 0 0.03
## 3 2 0.25
## 4 3 0.34
```

Instead of index numbers, we can provide a conditional statement, which will be true or false for every row.

```
# Display the rows where x < 3, all columns
prob.dist[prob.dist$x<3, ]
```

```
##   x P.x.
## 1 0 0.03
## 2 1 0.13
## 3 2 0.25
```

The conditional statement itself produces a vector of `TRUE` or `FALSE` values. We can see this if we examine it directly.

```
prob.dist$x < 3
```

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

Thus, we could produce the same subset by directly passing a vector of `TRUE` or `FALSE` values.

```
# Display first 3 rows (Remember TRUE/FALSE can be abbr. T/F)
prob.dist[c(T,T,T,F,F,F), ]
```

```
##   x P.x.
## 1 0 0.03
## 2 1 0.13
## 3 2 0.25
```

Examples of kinds of conditional statements and joining statements are below.

```
# Testing equality, use two equal signs (==)
prob.dist$x == 4
```

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

```
# Negation
prob.dist$x != 4     # Not equal to 4
```

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

```
!(prob.dist$x <= 2)     # Not less than or equal to 2
```

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

```
# Joining
# x >= 2 AND x < 5
prob.dist$x >= 2 & prob.dist$x < 5
```

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

```
# X < 3 OR prob > 0.2
prob.dist$x < 3 | prob.dist$P.x. > 0.2
```

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

This is a powerful feature of R. The conditional can be based on just about anything, columns of a date frame or even seemingly unrelated variables, as long as the resulting `TURE/FALSE` vector is the same length as the object being subsetted.

Our problem, however, is a simple one. Suppose we want the probability of x being at least 4.

```
sum(prob.dist$P.x.[prob.dist >= 4])
```

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

Binomial Probability Distributions

In the Chapter 4 guide, we discussed functions with names in the form `r`+distribution name. There are other functions with similar naming conventions for working with the standard distributions (which you may have noticed if you examined the documentation for `rbinom`, for instance).

First are the density functions, such as `dbinom()`. For discrete distributions, the density function will give the probability of a single value (we will see later that density has a slightly different interpretation for continuous distributions). Suppose we are flipping a fair coin 10 times. What is the probability of getting exactly 4 heads?

```
# Probability of 4 successes out of 10 trials with p=0.5
dbinom(4, 10, prob=0.5)
```

```
## [1] 0.2050781
```

The probability functions, such as `pbinom()`, give probabilities for a range of values. The probability of 4 or less heads is...

```
pbinom(4, 10, prob=0.5)
```

```
## [1] 0.3769531
```

To find a range of probabilities greater than some value, we can use the optional parameter `lower.tail`. (The parameter name can be abbreviated `lower` or even `low`. Keep readability in mind, however, if there is a chance your code will be revisited in the future.) The default value for `lower.tail` is `TRUE` and returns probabilities for number of successes less than or equal to the specified value. If set to `FALSE`, the function will return probabilities for number of successes greater than the specified value. Notice the distinction. `lower.tail = TRUE` gives $P(X \leq x)$ (less than *or equal to*) whereas `lower.tail = FALSE` gives $P(X > x)$ (greater than). Thus, to find probabilities for complex events ("at least x"), an adjustment in the parameters needs to be made. To find the probability of *at least* 7 heads,

```
# At least 7 successes = greater than 7-1 successes
pbinom(6, 10, prob=0.5, lower.tail=F)
```

```
## [1] 0.171875
```

Finally, the quantile functions, such as `qbinom()`, are, in a sense, the inverse of the probability functions. They give a value which will yield a specified probability. For example, in 10 coin flips, what number of heads, or less, will occur 75% of the time?

```
# What number of successes or less have the probability of 0.75
qbinom(.75, 10, prob=.5)
```

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

They, like the probability functions, have the optional parameter `'lower.tail` which operates in a similar fashion. So, to find a value where the number of heads is greater than the value 25% of the time,

```
qbinom(.25, 10, prob=0.5, lower.tail=FALSE)
```

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

The quantile functions can be used to find the boundary values between usual and unusual values (more commonly known as *critical values* as we will learn in a future chapter). Thus, the boundary for unusually number of heads in 10 flips is

```
qbinom(0.05, 10, prob=0.5)
```

```
## [1] 2
```

Thus, if you flipped a coin 10 times and got 2 or fewer heads, you might question whether it was a fair coin.

Poisson Probability Distribution

The Poisson distribution has the same functions available as the binomial, named `dpois()`, `ppois()`, etc. The main difference to note is that where the binomial had two parameters (n, p), the Poisson only has one (λ). Remember, λ should be in the same scale that you are testing. For example, if you are given a rate per hour, but you wish to test for events per minute, send rate / 60 as your λ.

Suppose a store gets 100 customers per hour. What is the probability that 15 or fewer customers will arrive in the next 10 minutes? What would be an unusually high number of customers to get in the next 10 minutes?

```
# 100 customers per hour
rate <- 100

# Probability of 15 or fewer events (customers)
ppois(15, lambda=rate/6)
```

```
## [1] 0.4022305
```

```
# Only a 0.05 probability of getting more than...
qpois(0.05, lambda=rate/6, lower=F)
```

```
## [1] 24
```

License