vitC.r: Explanation of code

Goals of code:

- Reading a contingency table in “count” form
- Contingency table tests, proportions, and measures of association
- Reading event data in “individual” form
- Exact tests for general contingency tables (more than 2x2).

Note: There are many different methods to analyze contingency table data. Everything we’ve done previously in the course has had alternative methods. Contingency table data has a lot of alternatives. The answers are usually only slightly different, especially with moderate to large sample sizes. I show how to get large sample results and one of the small sample (“exact”) results.

Reading “count form” data: vitc.txt
When the counts in a contingency table are pre-tabulated, the easiest way to do stats on that table is to enter the table with one row of data for each cell (combination of row and column) in the table. The key piece of information is the number of observations in that cell. Here we store that in the \( n \) variable. The row and column identifiers can be stored as character or numeric variables. \( n \) must be numeric.

Contingency table computations: chisq.test()
Most computations want the contingency to be represented by a table. The matrix() function converts the vector of counts in \( \text{vitc}\$n \) into two rows and two columns. The arguments are the vector of counts, information about the size of the matrix, and information about the ordering of numbers. The \( \text{nrow=} \) argument specifies the number of rows. You can also specify \( \text{ncol=} \) or both. If you only specify one, the missing value is determined from length of the starting vector and the specified number. Here, there are four counts in the vitc data frame and we specify 2 rows. Hence, there are 2 columns. If there had been 6 counts, we would have gotten 3 rows.

The \( \text{byrow=}T \) specifies how R is to fill the matrix. By default, matrices are filled going “down the columns”, so the vector 1, 2, 3, 4 produces the matrix:

\[
\begin{array}{cc}
1 & 3 \\
2 & 4 \\
\end{array}
\]

It is customary to put groups or treatments in rows and the different outcomes in columns. That means the values in the vitamin C data frame should fill the matrix “going across”, that is 1, 2, 3, 4 should become

\[
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}
\]

This is specified by adding \( \text{byrow=}T \).
The Chi-square test is provided by the `chisq.test()` function. The input is the \( r \times c \) matrix of counts in the contingency table. The output includes the \( \chi^2 \) statistic, its df, and the p-value. In lecture, I briefly discuss the continuity correction and why I prefer to not use it. The `correct=F` argument turns off that correction.

Fisher’s exact test is provided by the `fisher.test()` function. Again, the input is the matrix of counts. This can be used for any size contingency table, not just a 2x2 table (although Fisher only described the 2x2 analysis).

**Manipulating tables of counts:**
The matrix `counts` is just the numbers. Sometimes, you want to reverse the rows (or the columns). This can be done various ways. One is to subscript rows in the desired new order. That’s what `counts[c(2,1),]` does. It is acting on rows because the vector of subscripts are before the comma. If subscripts are after the comma, i.e., `counts[, c(2,1)]`, that would switch the columns.

Sometimes it helps to have names for the rows and columns. For a matrix, these are stored as the `dimnames` attribute. This is a list of two vectors. The first is the row names; the second is the column names. A list is a way of storing related pieces of information. We’ve seen various lists before, i.e. in the output from `predict()` with `se.fit=T`. They are usually things with `$itemname`. Those are named lists, where each piece of information has a name. An unnamed list is just that, where each item is accessed or stored by double-bracket subscripting, `[[ ]]`. If you want to add both row and column names, it is easier to specify the list, as done here. If you only want to change one, it is easier to do the following:

```r
dimname(vitc)[[1]] <- c(’A’, ’B’) which changes the first list item (row names) to the specified vector. You can access or print the row or column names by `dimnames(vitc)` or `dimnames(vitc)[[1]]` or `dimnames(vitc)[[2]]`.
```

**Odds ratio and relative risk statistics:**
Neither is part of basic R. Lots of libraries have been written to do these computations. I illustrate the functions in the epitools library.

The `oddsratio()` function estimates the odds ratio, calculates a confidence interval, and tests whether the odds ratio = 1.

The odds ratio is calculated as the odds of the second column in each row divided by the odds of the second column in the first row. So for the 2 x 2 table for the VitC and colds data, the odds ratio for the first row is reported as 1 (row 1 / row 1). The odds ratio for the second row is calculated as \((105/302) / (76/335)\), where 105/302 is the odds of ’not cold’ in the VitC group and 76/335 is the odds of ’not cold’ in the Placebo group. If you want to change the “event” for which odds are calculated, switch the columns in the count matrix or add the `rev=` option (example a few lines down). The `rev=` accepts ’columns’, ’rows’, or ’both’.

You can provide either the matrix of counts or the vector of counts, which is interpreted as two outcomes, values specified by row.
The output has three major items:

- \textbf{$\$data$} contains the data table, after converting a vector to a matrix, if necessary, and reversing rows or columns, if requested.
- \textbf{$\$measure$} contains the estimated odds ratio, with a confidence interval. The confidence interval is calculated as discussed in lecture, by working on the log OR scale and backtransforming.
- \textbf{$\$p.value$} contains the p-value, computed using 3 different approaches. The one discussed in lecture is the Chi-square approach.

The other two pieces say whether a continuity correction was used and include more information about the method used in the analysis.

You can add various options, all described in the help file (type \texttt{?oddsratio} or \texttt{help(oddsratio)} to see the help file). We’ve already discussed \texttt{rev=}. Another useful one is \texttt{conf.level=} , which changes the coverage for the confidence intervals. Specifying 0.90 gets you a 90% confidence interval.

The \texttt{riskratio()} function is identical to the \texttt{oddsratio()} function, except that it calculates the risk ratio, not the odds ratio.

\textbf{Confidence intervals for binomial proportions: binom.approx()}\par
The epitools library provides three functions to calculate CI’s for proportions. We’ve focused on the large sample approximation. That’s provided by the \texttt{binom.approx()} function. The two arguments are the number of events and the number of trials.

\textbf{NOTE:} The \texttt{binom.approx()} function requires different information from the other functions discussed here. Other functions want \# events and \# non-events (i.e. cold and not cold). \texttt{binom.approx()} wants the \# colds and the total \# people in that group. If you give it \# events and \# non-events, you will either get the wrong proportion or a non-informative warning about NaN’s produced.

If you want one version of an exact confidence interval, that is provided by \texttt{binom.exact()}.

The output includes the number of events and number of trials (x and n), the estimated proportion, the confidence interval (lower, upper), and the confidence level. You can change the confidence level by adding the \texttt{conf.level=} argument.

\textbf{Analyzing “individual” data:} \par
The madeup.txt file contains a small data set where each row represents one individual. Each individual is categorized by their value for variable a (2 levels) and their value for variable b (3 levels).

The \texttt{table()} function calculates the number of observations in each cell of a table. If you give it
two arguments, you get a two-way table of counts. I used with() in the example code. You could have just used `table(madeup$a, madeup$b).

When each row is an individual, you can give the original data to `chisq.test()`. You could compute the counts first, but you don’t have to.

You get a warning message because the expected counts are small.

**Fisher’s test for general tables (more than 2 x 2):**
The R implementation of fisher.test allows for any contingency table. Just don’t try to use it for a large table, because the computations can be slow.