Improved implementation of the marginal independence test

Eugeny Chankov

2017-12-21

Abstract

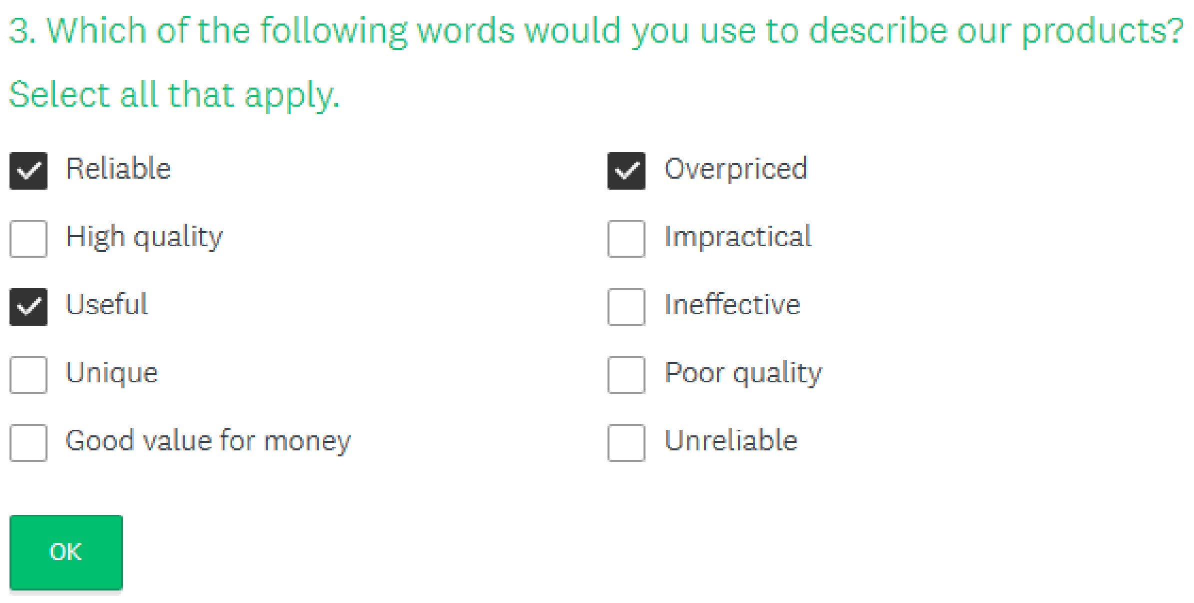
R package MRCV is used to analyse data with multiple responses categorical variables and perform statistical tests for their independence. Applying these tests to relative large tables takes a lot of time or leads to errors due to memory limitation. This paper provides more efficient computational methods for realizing tests of independence for this type of variables.

Introduction

We will begin with three questions to describe the subject of the article and its purpose.

What are the multiple response variables?

Surveys often contain questions in which respondents are able to choose from the list of answer options as many answers as they like. The example of such question below is taken from the Customer Satisfaction Survey Template provided by Survey Monkey.



3. Which of the following words would you use to describe our products?

Select all that apply.

[✓ Reliable]	[✓ Overpriced]
High quality	Impractical
[✓ Useful]	Ineffective
Unique	Poor quality
Good value for money	Unreliable

Customer Satisfaction Survey Template. Source: Survey Monkey

What is the independence between two multiple-response variables?

When both tested variables are allowed only single response, then their joint distribution is fully determined by the contingency table of the variables. To test independence between these variables, well-known Pearson chi-squared test is often used. The chi-squared test compares values of this contingency table and table with the expected values.

When at least one of the variables is allowed multiple response, it is possible define so-called marginal table. This table is similar to the contingency table. The difference is that the marginal table do not determine the variables joint distribution. Applying directly the chi-squared test to the marginal table do not give us answer about the variables independence.

We will be following (Bilder and Loughin 2004) and use the notations from this article. The hypothesis of independence between two multiple response variables is called simultaneous pairwise marginal independence (SPMI) and formulated in the following way. Let

- M be $r \times c$ marginal table of two multiple response variables;
- π_{ij} be frequencies of M and $\pi_{i\bullet}, \pi_{\bullet j}$ be rows and columns frequencies of M.

Than the hypothesis for a test of SPMI are

$$\begin{aligned} \mathrm{H}_0 &: \pi_{ij} = \pi_{i\bullet}\pi_{\bullet j} \quad \text{for } i = 1, \dots, r \text{ and } j = 1, \dots, c, \\ \mathrm{H}_a &: \text{At lest one equality does not hold.} \end{aligned}$$

To test SPMI hypothesis Builder and Loughin propose using Rao-Scott second-order adjustment to the chi-squared test.

Why do we need to improve the computations in the MRCV package?

Let us return to the example with multiple response variable from the Customer Satisfaction survey. Suppose we have two such variables with the answers about two products. We want to know whether these variables are independent.

We may use the function `MI.test()` of the `MRCV` package (Koziol and Bilder 2014) to test the marginal independence between the variables. `MI.test()` provides three approaches to perform the SPMI test.

1. **Bootstrap.** The most accurate method among these approaches, but it needs a lot of time to compute the result, in particular when the sample size is large enough. This method is difficult to use if we need to compare relatively many pairs of multiple-response variables.
2. **Bonferroni correction.** The fastest method among these approaches, but its results should be treated skeptically, especially for 10×10 table as in our example.
3. **Rao-Scott 2nd order correction.** This method should be a good balance between the bootstap and the Bonferroni correction. Actually, applying `MI.test()` function for our table we got an error, since `R` requires more than 32 GB RAM to calculate it.

This paper will show how to implement Rao-Scott 2nd order correction to the SPMI test, without requiring a large volume of RAM and get good computational performance.

Guide to this paper

- The **Example** section provides details about data set that will be used in benchmarks and gives the example where `MI.test()` function return the error due to memory limitation.
- In the **Optimisation of the SPMI test calculation** section explains how to improved calculation of the SPMI test.
- The **Benchmarks** section shows that the improved `MI.test()` is $100 - 1000$ times faster than its standard realization in `MRCV` package and free from errors with RAM usage.
- In the last section **Another improvement: Adjusting non-conformable arguments** reviewed the cases when the standard `MI.test()` function returns non-conformable arguments error.

1. Example

We will be used dataset with cars attributes ratings(Van Gysel 2011) that is included to R-package `plfm` (Meulders and De Bruecker 2017). The dataset contains the ratings of 78 respondents about the association between each of 14 car models and each of 27 car attributes. That is, each respondent put tick to a cell (i, j) of a 14 x 27 table if he/she supposed that car i has attribute j.

The table below shows, for 2 car models and 8 attributes, how many of 78 respondents indicate an association between each car model and each attribute.

Table with totals

	Agile	Economical	Popular	Practical	Reliable	Safe	Sustainable	Versatile
Ford Focus Cmax	22	16	23	49	33	25	20	31
Volkswagen Golf	40	29	55	44	58	40	43	25

Sample size: 78; Source: car dataset from package plfm

These rating variables were allowed multiple response. Here is their marginal table.

Marginal table

	Volkswagen Golf							
	Agile	**Economical**	**Popular**	**Practical**	**Reliable**	**Safe**	**Sustainable**	**Versatile**
Ford Focus Cmax								
Agile	17	11	17	16	18	14	14	10
Economical	10	9	15	12	14	11	13	6
Popular	15	13	18	17	19	13	15	6
Practical	31	18	41	33	42	29	31	18
Reliable	23	13	26	23	28	20	20	12
Safe	19	13	21	19	23	17	20	8
Sustainable	13	9	16	14	17	15	16	8
Versatile	26	14	26	21	29	17	19	14

Sample size: 78; Source: car dataset from package plfm

We want to test weathers these two multiple-response variables are independent. Firstly let us prepare data to `MI.test()`

```r
library(plfm); data(car)
library(data.table)

carData <-data.table(car$datalongformat[,c("IDrater", "rating",
                                           "objectlabel", "attributelabel")])
carData[, rating := as.integer(rating)]
setnames(carData, c(1,3,4), c("rater", "brand", "attribute"))
setkey(carData, brand, attribute)
```

```
# selecting necessary car models and attributes
selected.cars <- c("Ford Focus Cmax", "Volkswagen Golf")
selected.attr <-c("Agile", "Economical",  "Popular", "Practical", "Reliable", "Safe",
 "Sustainable", "Versatile")
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
```

Trying to perform SPMI test with rs2 method to the data leads to the error.

```
library(MRCV)
MI.test(as.data.frame(data), 8, 8, type = "rs2")
```

```
## Error: cannot allocate vector of size 32.0 Gb
```

2. Optimisation of the SPMI test calculation

We are talking about "RS2" approach for the SPMI test. We need to repeat the notation used in (Bilder and Loughin 2004).

Let W and Y denote the multiple-response categorical variables. Suppose W has r items (categories) and Y has c items (categories). Also suppose n subjects are sampled at random. In our example $r = c = 8$ and $n = 78$.

Let $W_{si} = 1$ if a positive response is given for item i by subject s for $i = 1, \ldots, r$ and $s = 1, \ldots, n$ and $W_{si} = 0$ for a negative response (= no response). Definition of Y_{sj} is the same.

Let m_{ij} denotes the number of positive observed responses to W_i and Y_j. Thus, in our marginal table $m_{43} = 41$ is the number of observed responses of respondents who supposed that Ford Focus C-max is "Practical" and Volkswagen Golf is "Popular".

A joint table $2^r \times 2^c$ gives the cross-classification of responses to each possible set of item responses for W and Y. Cell counts in the joint table are denoted by $\mathbf{n} = (n_{11}, n_{12}, \ldots, n_{2^r 2^c})$. Each value of the marginal table $\mathbf{m} = (m_{11}, m_{12}, \ldots, m_{rc})$ can be found as the following sum

$$m_{ij} = \sum_{g,h\,:\,W_i=1\ \&\ Y_j=1} n_{gh},$$

where n_{gh} is cell count of the joint table for the gth possible $(W_1, \ldots, W_r)$ and hth possible $(Y_1, \ldots, Y_c)$. The corresponding probability for the cell gh is denoted by τ_{gh} and we get that $\sum_{g,h} \tau_{gh} = 1$.

Also, let $\mathbf{G}$ be a $r \times 2^r$ matrix with columns containing all possible values of $(W_1, \ldots, W_r)'$, and let $\mathbf{H}$ be a $c \times 2^c$ matrix with columns containing all possible values of $(Y_1, \ldots, Y_c)'$. Then values of the marginal table can be represented in the matrix form as follows:

$$\mathbf{m} = (\mathbf{G} \otimes \mathbf{H})\mathbf{n}.$$

This can be written equivalently as

$$\hat{\pi} = (\mathbf{G} \otimes \mathbf{H})\hat{\tau},$$

where $\hat{\pi} = \mathbf{m}/n$ and $\hat{\tau} = \mathbf{n}/n$.

The main performance gain

We skip some additional definitions and focus on the most computationally intensive part of calculations. It is estimation of the covariance matrix Σ that calculated as follows

$$\Sigma = F\left[\mathrm{Diag}(\tau) - \tau\tau'\right]F',$$

where F is some matrix.

The length of the vector of multinomial probabilities τ is 2^{r+c}. That is why dimension of the matrix $\mathrm{Diag}(\tau) - \tau\tau'$ is $2^{r+c} \times 2^{r+c}$. In our example this matrix size is $2^{16} \times 2^{16}$.

We can see that zero-valued multinomial probabilities can be excluded from calculation of the matrix Σ. That is

$$F\left[\mathrm{Diag}(\tau) - \tau\tau'\right]F' = \mathcal{F}\left[\mathrm{Diag}(\mathcal{T}) - \mathcal{T}\mathcal{T}'\right]\mathcal{F}',$$

where $\mathcal{T} = \tau[\,\tau > 0\,]$ and $\mathcal{F} = F[\,\bullet\,, \tau > 0\,]$.

Actually τ contains no more than n non-zero elements (in practice less than n). Even if r and c are not so large, $n \ll 2^{r+c}$.

Package data.table as another tool to reach performance

The `data.table` package (Dowle and Srinivasan 2017) offers fast and memory efficient work with data and provides flexible and readable syntax. We will be use exactly the same input data as `MI.test()` function but wrapped to the data.table class.

Modified script for the SPMI test

```r
n <- nrow(data)
setkeyv(data, names(data))
# get all non-zeros combos of W and Y categories
N <- data[,list(freq = .N), by = key(data)]
N.freq <- N[, freq] # store the frequencies in a separate vector
# remove column freq from N and convert N to the matrix
N <- as.matrix(N[, freq := NULL])

Tau <- N.freq/n # the vector of non zero-value multinomial probabilities
GH <- apply(N, 1, function(x) x[1:I] %x% x[(1:J) + I]) # kroneker product of G and H

# the vector of totals for W and Y categories
Totals <- apply(N, 2, function(x) sum(x*N.freq))
Pi.row <- Totals[1:I]/n
Pi.col <- Totals[1:J + I]/n

F <- GH - kronecker(Pi.row, t(N[,1:J + I])) - kronecker(t(N[,1:I]), Pi.col)
# where  G.ij = t(N[,1:I]) and H.ji = t(N[,(1+I):(I+J)])
# Find the sum of the squares of the eigenvalues of the matrix D^{-1}\Sigma
Mult.cov <- diag(Tau) - tcrossprod(Tau, Tau)
sigma <- F%*%tcrossprod(Mult.cov, F)
D.inv <- diag(1/as.vector(kronecker(Pi.row, Pi.col)*kronecker(1-Pi.row, 1-Pi.col)))
Di.sigma <- D.inv%*%sigma
Di.sigma.eigen <- Re(eigen(Di.sigma, only.values = TRUE)$values)
sum.Di.sigma.eigen.sq <- sum(Di.sigma.eigen^2)
```

Testing the independence between a single-response and a multiple-response variables

The SPMI test is performed to testing independence between two multiple response variables. When one of the two variables is single response, multiple marginal independence (MMI) is tested. This kind of test is also supported by the `MI.test()` function of the `MRCV` package.

As for the SPMI test, realization of the MMI test (type = “RS2”) can be improved in the similar way. We omit the details about this improvements here. You may find them in the source code.

3. Benchmarks

We compare performance characteristic of SPMI and MMI tests of “RS2” type implemented in the function `MI.test(type = 'rs2')` of the `MRCV` package with their modified versions implemented in a function `multiple.chiRS2.test()`. These functions return the same results of the “RS2” test.

In additional to the `data.table` package, the `Matrix` (Bates and Maechler 2017) and the `checkmate` (Lang and Bischl 2017) packages are used.

```
library(Matrix)
library(checkmate)
library(microbenchmark)
source("mi.test_improvements.R") # reading multiple.chiRS2.test() function
```

Return to the example with Ford Focus C-max and Volkswagen Golf and 8 attributes. As we have already known, applying the `MI.test()` to this data leads to the error. Remove the “Versatile” attribute for both variables and perform the SPMI test for this data.

SPMI test for two 7-category multiple response variables

```
selected.cars <- c("Ford Focus Cmax", "Volkswagen Golf")
selected.attr <- c("Agile", "Economical",  "Popular", "Practical","Reliable","Safe",
"Sustainable")
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
I <- J <- ncol(data)/2
df <- as.data.frame(data)
```

At first we compare the functions output results.

```
spmi.result_mrcv.approach <- MI.test(df, I, J, type = "rs2")
spmi.result_improved.approach <- multiple.chiRS2.test(data, I, J)

all.equal(spmi.result_mrcv.approach$rs2, spmi.result_improved.approach[-1])
```

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

Then we compare performance of these functions.

```
microbenchmark(MI.test(df, I, J, type = "rs2"), times = 10, unit = "ms")
```

```
## Unit: milliseconds
##                             expr      min       lq     mean   median
##  MI.test(df, I, J, type = "rs2") 22358.43 23645.25 25810.51 25694.69
##       uq      max neval
##  27390.9 30396.94    10
```

```
microbenchmark(multiple.chiRS2.test(data, I, J), times = 100, unit = "ms")
```

```
## Unit: milliseconds
##                              expr      min      lq     mean   median
##  multiple.chiRS2.test(data, I, J) 8.795603 9.57288 12.97078 11.17836
##        uq      max neval
##  14.38676 49.29342   100
```

As we see `multiple.chiRS2.test()` give the same results as `MI.test(type = 'rs2')` more than 2000 times faster for two multiple response variables with 7 categories.

Why this performance is important?

If we need perform SPMI test for each of 91 pairs of car models with some fixed 7 attributes at a laptop with Intel Core i7 and 8GB RAM we spent **about 39 minutes** with `MI.test()` function and **1 second** with `multiple.chiRS2.test()` . Moreover, every time when attributes set is change we have to recompute all these routines again.

SPMI test for two 27-category multiple response variables

```
selected.cars <- c("Ford Focus Cmax", "Volkswagen Golf")
data <- carData[.(selected.cars)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
I <- J <- ncol(data)/2
```

```
microbenchmark(multiple.chiRS2.test(data, I, J), times = 10, unit = "ms")
```

```
## Unit: milliseconds
##                                expr      min       lq     mean   median
##  multiple.chiRS2.test(data, I, J) 1035.469 1101.231 1260.893 1164.486
##        uq      max neval
##  1287.499 1977.665    10
```

MMI test for 2-category single-response variable and 13-category multiple response variable

```
selected.cars <- c("Ford Focus Cmax", "Volkswagen Golf")
selected.attr <- levels(carData$attribute)[1:13]
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
data <- data[, 1:(length(selected.attr)+1), with = FALSE]

df <- as.data.frame(data)
I <- length(selected.attr)
J <- 1
```

Make sure that the results are the same.

```
mmi.result_mrcv.approach <- MI.test(df, I, J, type = "rs2")
mmi.result_improved.approach <- multiple.chiRS2.test(data, I, J)

all.equal(mmi.result_mrcv.approach$rs2, mmi.result_improved.approach[-1])
```

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

These are performance test results.

```
microbenchmark(MI.test(df, I, J, type = "rs2"), times = 10, unit = "ms")
```

```
## Unit: milliseconds
##                               expr      min       lq     mean  median
##  MI.test(df, I, J, type = "rs2") 14143.01 14685.03 15769.42 15797.6
##       uq      max neval
##  16635.4 18122.52    10
```

```
microbenchmark(multiple.chiRS2.test(data, I, J), times = 100, unit = "ms")
```

```
## Unit: milliseconds
##                                expr      min       lq     mean  median
##  multiple.chiRS2.test(data, I, J) 13.14583 14.50321 17.01227 15.7167
##        uq      max neval
##  18.25834 28.81259   100
```

MMI test for 2-category single response variable and 15-category multiple response variable

```
selected.attr <- levels(carData$attribute)[1:15]
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
data <- data[, 1:(length(selected.attr)+1), with = FALSE]

df <- as.data.frame(data)
I <- length(selected.attr)
J <- 1
```

```
mmi.result_mrcv.approach <- MI.test(df, I, J, type = "rs2")
```

```
## Error: cannot allocate vector of size 32.0 Gb
```

```
mmi.result_improved.approach <- multiple.chiRS2.test(data, I, J)
mmi.result_improved.approach[-1]
```

```
## $X.sq.S.rs2
## [1] 22.31872
##
## $df.rs2
## [1] 10.18669
##
## $p.value.rs2
## [1] 0.01490186
```

MMI test for 2-category single response varaible and 27-category multiple response variable

```
selected.cars <- c("Ford Focus Cmax", "Volkswagen Golf")
selected.attr <- levels(carData$attribute)
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
data <- data[, 1:(length(selected.attr)+1), with = FALSE]

df <- as.data.frame(data)
I <- length(selected.attr)
J <- 1
```

```
microbenchmark(multiple.chiRS2.test(data, I, J), times = 10, unit = "ms")
```

```
## Unit: milliseconds
##                                expr      min       lq   mean   median
##  multiple.chiRS2.test(data, I, J) 13.12057 13.33571 14.287 13.55165
##        uq      max neval
##  14.09444 20.55623    10
```

Benchmark test for simulated data with larger sample size

Perhaps, 78 respondents is too small to compare performance of SPMI test implementation. We consider a data set with $78 \cdot 7 = 546$ observations for 14 = 7 +7 items of two multiple-response variables.

Here is the data

```
selected.attr <- c("Agile", "Economical",  "Popular", "Practical","Reliable","Safe",
"Sustainable")
data1 <- rbindlist(lapply(levels(carData$brand)[1:7], function(x) {
                     dt <- carData[CJ(x, selected.attr)]
                     dt <- dcast(dt, rater ~ brand + attribute, value.var = "rating")
                     dt[, rater := NULL]
                }))
data2 <- rbindlist(lapply(levels(carData$brand)[8:14], function(x) {
                     dt <- carData[CJ(x, selected.attr)]
                     dt <- dcast(dt, rater ~ brand + attribute, value.var = "rating")
                     dt[, rater := NULL]
                }))
data <- cbind(data1, data2)
df <- as.data.frame(data)
```

Time test for `multiple.chiRS2.test()`, `MI.test(type = "rs2")` and `MI.test(type = "boot")` — bootstrap approach with default number of resamples.

```
microbenchmark(multiple.chiRS2.test(data, 7, 7), times = 100, unit = "ms")
```

```
## Unit: milliseconds
##                                expr     min       lq     mean   median
##  multiple.chiRS2.test(data, 7, 7) 27.2383 29.4671 34.96926 31.94026
##        uq      max neval
##  35.44887 121.5006   100
```

```
microbenchmark(MI.test(df, 7, 7, type = "rs2"), times = 10, unit = "ms")
```

```
## Unit: milliseconds
##                               expr      min       lq    mean  median
##  MI.test(df, 7, 7, type = "rs2") 22573.85 24576.74 26947.5 27043.5
##        uq      max neval
##  28910.64 32016.49    10
```

```
microbenchmark(MI.test(df, 7, 7, type = "boot", print.status = FALSE), times = 1, uni
t = "ms")
```

```
## Unit: milliseconds
##                                                     expr      min       lq
##  MI.test(df, 7, 7, type = "boot", print.status = FALSE) 213431.4 213431.4
##      mean   median       uq      max neval
##  213431.4 213431.4 213431.4 213431.4     1
```

4. Another improvement: Adjusting non-conformable arguments

```
selected.cars <- c("Audi A4", "Renault Espace")
selected.attr <- c("Agile", "High trade-in value", "Safe")
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
df <- as.data.frame(data)
```

`MI.test()` will return the following error message

```
MI.test(df, 3, 3, type = "rs2")
```

```
## Error in H %*% kronecker(t(j.2r), i.2c): non-conformable arguments
```

The reason is that none of the respondent chose the “High trade-in value” attribute to Renault Escape.

The function `multiple.chiRS2.test()` removes such variables on the fly and apply the SPMI or the MMI test for suitable data.

```
multiple.chiRS2.test(data, 3, 3)
```

```
## $X.sq.S.ij
##                             Renault Espace_Agile Renault Espace_Safe
## Audi A4_Agile                          7.7547826          0.01761717
## Audi A4_High trade-in value            0.8114987          4.13618412
## Audi A4_Safe                           0.3229814          4.85458611
##
## $X.sq.S.rs2
## [1] 13.74267
##
## $df.rs2
## [1] 4.607085
##
## $p.value.rs2
## [1] 0.01310041
```

is the same as SPMI test for the data without “Renault Espace_High trade-in value” variable.

Here is the similar case when the SPMI test is reduced for suitable data to the SPMI test.

```
selected.cars <- c("Audi A4", "Renault Espace")
selected.attr <- c("Agile", "High trade-in value")
data <- carData[CJ(selected.cars, selected.attr)]
data <- dcast(data, rater ~ brand + attribute, value.var="rating")
data[, rater := NULL]
```

```
multiple.chiRS2.test(data, 2, 2)
```

```
## $X.sq.S.ij
##               Audi A4_Agile Audi A4_High trade-in value
##                   7.7547826                   0.8114987
##
## $X.sq.S.rs2
## [1] 11.95638
##
## $df.rs2
## [1] 2.791498
##
## $p.value.rs2
## [1] 0.006144614
```

References

Bates, D., and M. Maechler. 2017. *Matrix: Sparse and Dense Matrix Classes and Methods* (version 1.2-11). https://CRAN.R-project.org/package=Matrix (https://CRAN.R-project.org/package=Matrix).

Bilder, C., and T. Loughin. 2004. “Testing for Marginal Independence Between Two Categorical Variables with Multiple Responses.” *Biometrics* 60 (1): 241–48. http://dx.doi.org/10.1111/j.0006-341X.2004.00147.x (http://dx.doi.org/10.1111/j.0006-341X.2004.00147.x).

Dowle, M., and A. Srinivasan. 2017. *Data.table: Extension of ‘Data.frame‘* (version 1.10.4). https://CRAN.R-project.org/package=data.table (https://CRAN.R-project.org/package=data.table).

Koziol, N., and C. Bilder. 2014. *MRCV: Methods for Analyzing Multiple Response Categorical Variables* (version 0.3-3). https://cran.r-project.org/package=MRCV (https://cran.r-project.org/package=MRCV).

Lang, M., and B. Bischl. 2017. *Checkmate: Fast and Versatile Argument Checks* (version 1.8.4). https://CRAN.R-project.org/package=checkmate (https://CRAN.R-project.org/package=checkmate).

Meulders, M., and Ph. De Bruecker. 2017. *Plfm: Probabilistic Latent Feature Analysis* (version 2.2.1). https://CRAN.R-project.org/package=plfm (https://CRAN.R-project.org/package=plfm).

Van Gysel, E. 2011. “Perceptuele Analyse van Automodellen Met Probabilistische Feature Modellen.” Master’s thesis, Hogeschool-Universiteit Brussel.