Analysis of the Sorting Task 'Ramen Noodles' with DistatisR

Hervé Abdi, Dominique Valentin, & Sylvie Chollet

2018-11-28

Contents

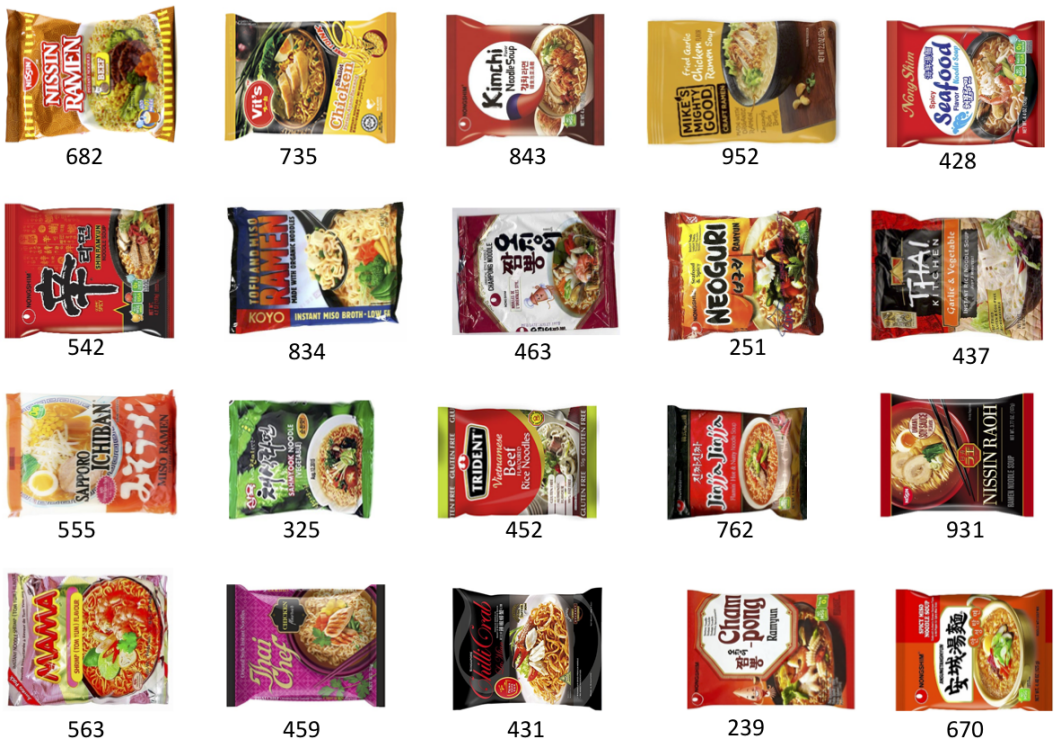


Figure 1: The Ramen Noodles Pictures


```
# A clean start
rm(list = ls())
graphics.off()
```

1 Introduction

This document presents the analysis of the Ramen Noodle sorting task collected with 30 participants on Tuesday July 24 2018 in the Advanced SPISE2018 workshop.

Participants were provided with 20 pictures (each printed on a 8cm by 13cm card) of packets of Ramen Noodles (see Figures below). Participants were given the following instructions:

"Look at the pictures and put together he Ramen noodles that for you belong to the same category.
You can make as many groups as you want (at least two groups but less than 20) and you can put as many samples as you want in each group."

2 Prelude to the analysis

If you want to make sure that you have a clean start, you can execute the following commands:

```
rm(list = ls())
graphics.off()
```

Or, better, you can (should? must?) use an `Rproject` for this project (see preamble below).

3 Preamble

3.1 Rprojects are best

Make sure that you start this analysis as a new `Rproject` so that the default directory will be correctly set.

Before we start the analysis, we need to have our three standard packages installed (from `Github`) and the corresponding libraries loaded:

- `DistatisR`
- `PTCA4CATA`
- `R4SPISE2018`

We also need some other packages namely:

- `Matrix`
- `factoextra`
- `ExPosition`

All these packages are installed or loaded with the following instructions:

```
# Decomment all/some these lines if the packages are not installed
# devtools::install_github('HerveAbdi/PTCA4CATA')
# devtools::install_github('HerveAbdi/DistatisR')
#  install.packages(prettyGraphs)
#  install.packages('Matrix')
#  install.packages('factoextra')
#  install.packages('ExPosition')
#  install.packages('pander') # nice pdf-tables
#
#  load the libraries that we will need
suppressMessages(library(Matrix))
suppressMessages(library(factoextra))
suppressMessages(library(DistatisR))
suppressMessages(library(PTCA4CATA))
suppressMessages(library(prettyGraphs))
suppressMessages(library(ExPosition))
```

3.2 The parameters

The name of the excel data file and of the excel sheets are given below:

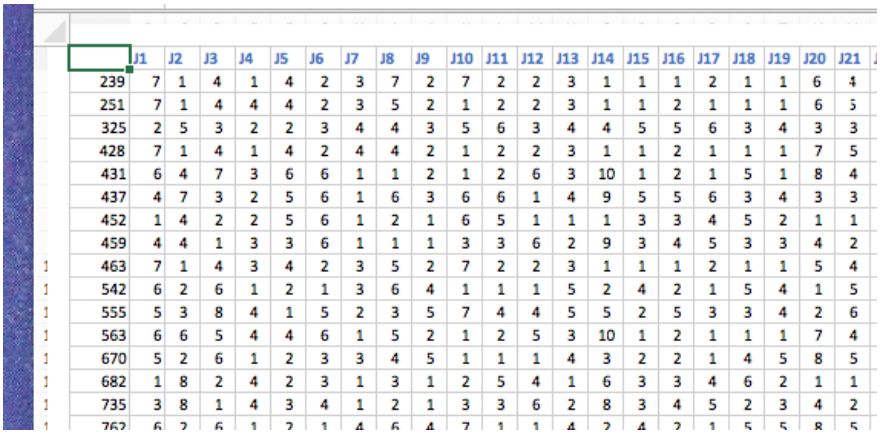
```
file2read.name <- 'dataSortingRamen.xlsx'
path2file <- system.file("extdata", file2read.name, package = "R4SPISE2018")
sheetName4Data       <- 'DataSort'
sheetName4Vocabulary <- "Vocabulary"
sheetName4Judges     <- "JudgesDescription"
```

4 Introduction to the analysis

The data are stored in an excel file called `dataSortingRamen.xlsx` whose location is stored in the variable `path2file`.

```
path2file <- system.file("extdata",
       "dataSortingRamen.xlsx", package = "R4SPISE2018")
```

if you open this excel file, it will look like the Figure above. The sorting data are stored in the `sheet` called DataSort (whose name is stored in the variable `sheetName4Data`). In this sheet, the first column gives the



	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10	J11	J12	J13	J14	J15	J16	J17	J18	J19	J20	J21
239	7	1	4	1	4	2	3	7	2	7	2	2	3	1	1	1	2	1	1	6	4
251	7	1	4	4	4	2	3	5	2	1	2	2	3	1	1	2	1	1	1	6	5
325	2	5	3	2	2	3	4	4	3	5	6	3	4	4	5	5	6	3	4	3	3
428	7	1	4	1	4	2	4	4	2	1	2	2	3	1	1	2	1	1	1	7	5
431	6	4	7	3	6	6	1	1	2	1	2	6	3	10	1	2	1	5	1	8	4
437	4	7	3	2	5	6	1	6	3	6	6	1	4	9	5	5	6	3	4	3	3
452	1	4	2	2	5	6	1	2	1	6	5	1	1	1	3	3	4	5	2	1	1
459	4	4	1	3	3	6	1	1	1	3	3	6	2	9	3	4	5	3	3	4	2
463	7	1	4	3	4	2	3	5	2	7	2	2	3	1	1	1	2	1	1	5	4
542	6	2	6	1	2	1	3	6	4	1	1	1	5	2	4	2	1	5	4	1	5
555	5	3	8	4	1	5	2	3	5	7	4	4	5	5	2	5	3	3	4	2	6
563	6	6	5	4	4	6	1	5	2	1	2	5	3	10	1	2	1	1	1	7	4
670	5	2	6	1	2	3	3	4	5	1	1	1	4	3	2	2	1	4	5	8	5
682	1	8	2	4	2	3	1	3	1	2	5	4	1	6	3	3	4	6	2	1	1
735	3	8	1	4	3	4	1	2	1	3	3	6	2	8	3	4	5	2	3	4	2
762	6	2	6	1	2	1	4	6	4	7	1	1	4	2	4	2	1	5	5	8	5

Figure 2: The Data Excel File

(#fig:xls.datafile)

names of the products (here spices) and the following columns give how each Judge sorted the products: The products that were sorted together are assigned the same number (arbitrarily chosen).

The contingency table storing the number of Judges using a descriptor (column) for a product (row), is stored in the sheet Vocabulary (whose name is stored in the variable `sheetName4Vocabulary`).

The description of the Judges (e.g., country, age, gender) is stored in the sheet JudgesDescription (whose name is stored in the variable `sheetName4Judges`).

When you record you own data, make sure that you follow the same format, this way the script described in this vignette will apply to your own analysis with minimal change.

We will first compute the results of the analysis, then create the graphics, and finally save everything into a PowerPoint.

5 Run the statistical analysis

5.1 Read the sorting data and the vocabulary

The excel file name and location (i.e., path) are stored in the variable `path2file`. To read the sorted data and the vocabulary contingency table we will use the function `DistatisR::read.df.excel()` (based upon the function `readxl::read_excel()`). We will use again this function to read the description of the judges.

```
# read the sortinig data and the vocabulary
multiSort.list <- read.df.excel(path = path2file,
                    sheet = sheetName4Data,
                    voc.sheet = sheetName4Vocabulary)
multiSort  <- multiSort.list$df.data
vocabulary <- multiSort.list$df.voc
```

The sorting data and the vocabulary are now stored into the list `multiSort.list` in which the sorting data are stored in the `dataframe` called `multiSort.list$df.data` and in which the vocabulary contingency table is stored in the dataframe called `multiSort.list$df.voc`. To facilitate coding, we put the sorting data and the vocabulary in the data frames called (respectively) `multisSort` and `vocabulary`.

5.1.1 Check (eye-balling) the data

To make sure that we have read the correct file we can peek at the dataframe `multiSort` and look at the data for the first 5 spices of the first 10 assessors:.

```
library(pander)
# knitr::kable(multiSort[1:5,1:10])
pander::pander(multiSort[1:5,1:10])
```

	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10
239	7	1	4	1	4	2	3	7	2	7
251	7	1	4	4	4	2	3	5	2	1
325	2	5	3	2	2	3	4	4	3	5
428	7	1	4	1	4	2	4	4	2	1
431	6	4	7	3	6	6	1	1	2	1

6 Read the Judges Description

The description of the judges can be found in the sheet `JudgesDescription and it is read with the functionDistatisR::read.df.excel‘` as:

```
# read the sortinig data and the vocabulary
judgesDescription <- read.df.excel(path = path2file,
                                   sheet = sheetName4Judges)$df.data
nVarJudges  <- ncol(judgesDescription)
```

The judges' description is now stored in the data frame `JudgesDescription`.

7 Participant analysis

7.1 First descriptor

```
k <- 1 # this is the first descriptor for the judges
```

Here we analyze the judges' descriptor number 1 (Gender). In order to run an analysis for another descriptor, we just need to change the value of `k` and re-run the analysis.

```
descJudges <- judgesDescription[,k ]
```

To color the graph of the participants, we need to associate a color to each level of the Judges' variable Gender; to do so we use the function `createColorVectorsByDesign()` from the package `prettyGraphs`.

```
# Create a 0/1 group matrix with ExPosition::makeNominalData()
nominal.Judges <- makeNominalData(as.data.frame(descJudges))
# get the colors
color4Judges.list <- prettyGraphs::createColorVectorsByDesign(nominal.Judges)
# color4Judges.list
```

7.2 Get the brick of distance

The first step of the analysis is to transform the sorting data into a brick of distance matrices. This is done with the function `DistanceFromSort()` as:

```
DistanceCube <- DistanceFromSort(multiSort)
```

The previous line of code, created the brick of distance (from the raw data) and stored it in the data frame `DistanceCube`.

7.3 Run plain DISTATIS

The brick of distance matrices (i.e., the data frame `DistanceCube`) is used as the argument of the function `distatis` that will compute the plain DISTATIS method for the sorting task.

```
resDistatis <- distatis(DistanceCube)
```

The list `resDistatis` contains the results of the analysis. This list contains two lists: The first list is called `resDistatis$res4Cmat`, it contains the results for the R_V analysis (because the R_V matrix is also called the **C** matrix); The second list is called `resDistatis$res4Splus`, it contains the results for the analysis of the compromise (called the **S** matrix) . If you have forgotten the output format of `distatis`, or want to have more information: have a look at the help for the function`distatis`: In the console type: `?distatis`, or print `resDistatis`.

7.4 Get group means in the RV space

We want to find if there are differences between the different groups of Judges corresponding to the levels of the variable Gender. The first step is to compute the mean of these groups and their bootstrap confidence intervals. This is done as:

```
# Get the factors from the Cmat analysis
G <- resDistatis$res4Cmat$G
# Compute the mean by groups of HJudges
JudgesMeans.tmp <- aggregate(G, list(descJudges), mean)
JudgesMeans <- JudgesMeans.tmp[,2:ncol(JudgesMeans.tmp )]
rownames(JudgesMeans) <- JudgesMeans.tmp[,1]
# Get the bootstrap estimates
BootCube <- PTCA4CATA::Boot4Mean(G, design = descJudges,
                        niter = 100,
                        suppressProgressBar = TRUE)
# head(BootCube)
```

7.5 Compute partial map (by groups of Judges)

The compromise is obtained as a weighted sum of the $\alpha_j \mathbf{S}_j$ with each of the J judges belonging to one of K groups (e.g., Gender). In plain `DISTATIS`, each observation can be projected onto the compromise; in a similar manner, when the assessors are nested into another group factor (e.g., here Gender), we can project these groups onto the compromise. The easiest way to do so is to use the partial projections and compute the weighted sum corresponding to each group.

```
F_j     <- resDistatis$res4Splus$PartialF
alpha_j <- resDistatis$res4Cmat$alpha
# create the groups of Judges
#groupsOfJudges <- substr(names(alpha_j),1,1)
groupsOfJudges <- descJudges
code4Groups <- unique(groupsOfJudges)
nK <- length(code4Groups)
# initialize F_K and alpha_k
F_k <- array(0, dim = c(dim(F_j)[[1]], dim(F_j)[[2]],nK))
dimnames(F_k) <- list(dimnames(F_j)[[1]],
                      dimnames(F_j)[[2]], code4Groups)
alpha_k <- rep(0, nK)
names(alpha_k) <- code4Groups
Fa_j <- F_j
# A horrible loop
for (j in 1:dim(F_j)[[3]]){ Fa_j[,,j]  <- F_j[,,j] * alpha_j[j] }
```

```r
# Another horrible loop
for (k in 1:nK){
  lindex <- groupsOfJudges == code4Groups[k]
  alpha_k[k] <- sum(alpha_j[lindex])
  F_k[,,k] <- (1/alpha_k[k])*apply(Fa_j[,,lindex],c(1,2),sum)
}
```

7.6 Projection of the vocabulary as supplementary elements

To compute the projection of the vocabulary onto the compromise space, we use the function `DistatisR::projectVoc`.

```r
F4Voc <- projectVoc(multiSort.list$df.voc, resDistatis$res4Splus$F)
```

8 Graphics

Most of the graphics will be created with either `prettyGraphs` or with `PTCA4CATA`, The graphs will be saved in a PowerPoint file.

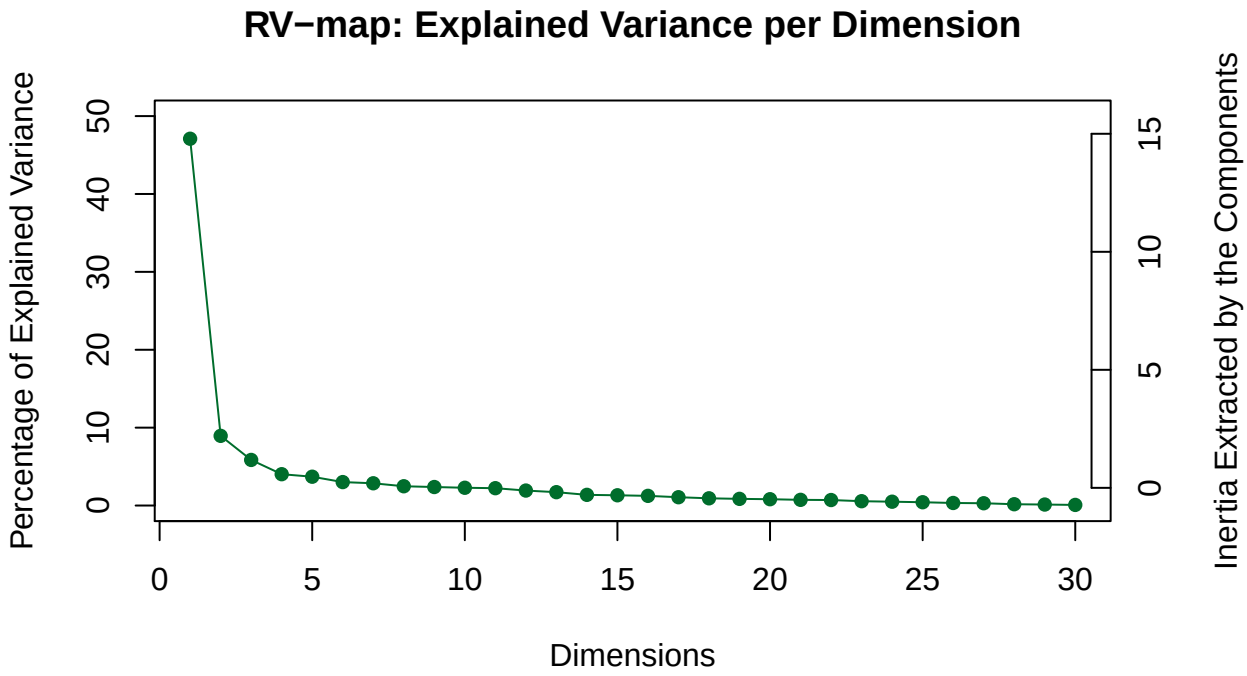
8.1 The RV analysis

In `DISTATIS`, the R_V analysis describes the Judges' similarity structure.

8.1.1 Scree plot for RV

In the first map shows the scree plot of the eigenvalues of the R_V between-Judges matrix.

```r
# 5.A. A scree plot for the RV coef. Using standard plot (PTCA4CATA)
scree.rv.out <- PlotScree(ev = resDistatis$res4Cmat$eigValues,
                  title = "RV-map: Explained Variance per Dimension")
```




```r
a1.Scree.RV <- recordPlot() # Save the plot
```

8.1.2 Factor Map for RV

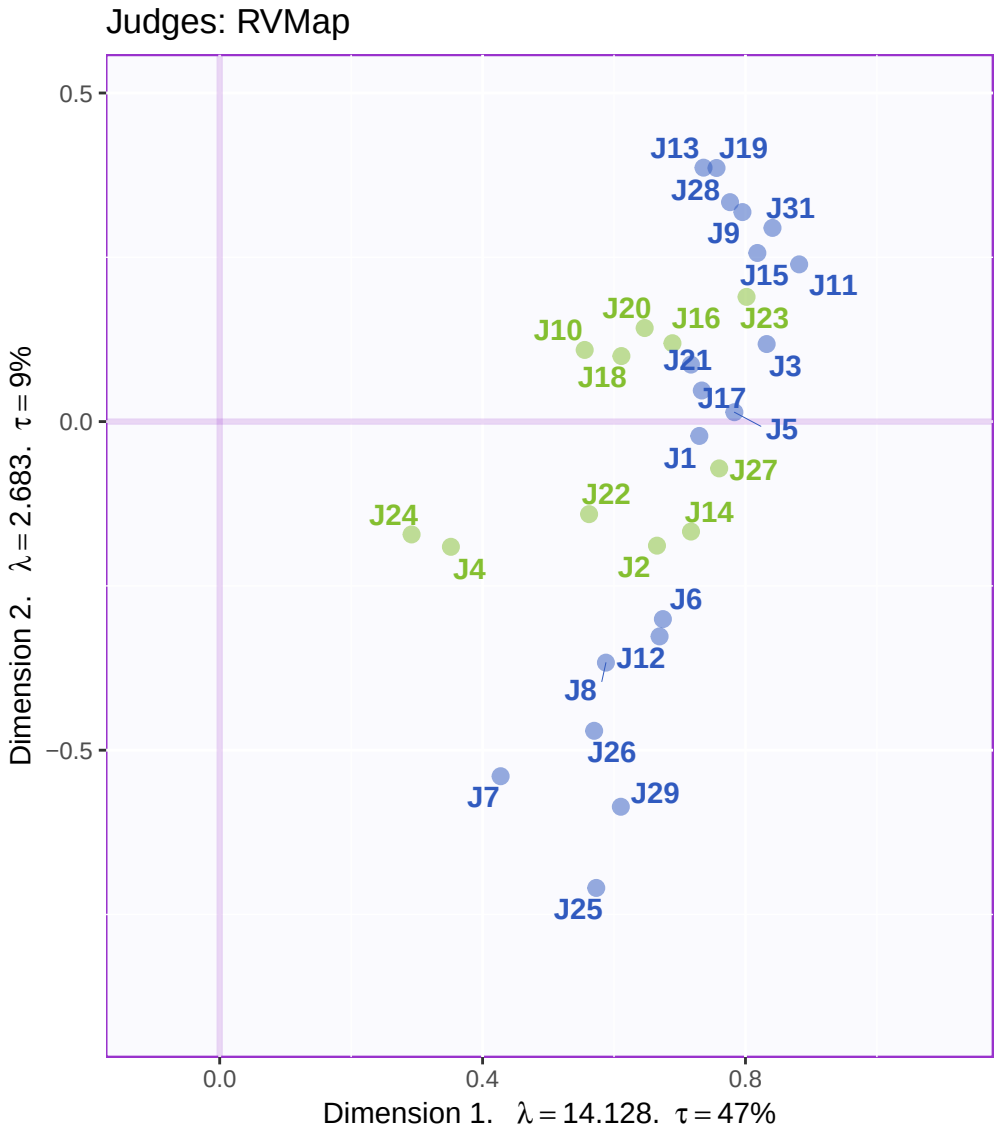
The eigen-analysis of the R_V between-Judges matrix gives the Judges' factor scores that are used to create the factor maps describing the between Judges similarity structure

```
# Create the layers of the map
gg.rv.graph.out <- createFactorMap(X = resDistatis$res4Cmat$G,
                           axis1 = 1, axis2 = 2,
                           title = "Judges: RVMap",
                           col.points = color4Judges.list$oc,
                           col.labels = color4Judges.list$oc)
# create the labels for the dimensions of the RV map
labels4RV <- createxyLabels.gen(
                  lambda = resDistatis$res4Cmat$eigValues ,
                  tau    = resDistatis$res4Cmat$tau,
                  axisName = "Dimension ")
# # Create the map from the layers
# Here with lables and dots
a2a.gg.RVmap <- gg.rv.graph.out$zeMap + labels4RV
# Here with colored dots only
a2b.gg.RVmap <- gg.rv.graph.out$zeMap_background +
                gg.rv.graph.out$zeMap_dots + labels4RV
```

8.1.3 Print the RV map

To print the RV map, we simply use the function `print()` as described below:

```
print(a2a.gg.RVmap )
```



8.1.4 An RV map with group means and confidence intervals

```
# First the means
# A tweak for colors
in.tmp    <- sort(rownames(color4Judges.list$gc), index.return = TRUE)$ix
col4Group <- color4Judges.list$gc[in.tmp]
#
gg.rv.means <- PTCA4CATA::createFactorMap(JudgesMeans,
                      axis1 = 1, axis2 = 2,
                      constraints = gg.rv.graph.out$constraints,
                      col.points =  col4Group ,
                      alpha.points = 1, # no transparency
                      col.labels = col4Group)
#
 dimnames(BootCube$BootCube)[[2]] <-
                     paste0('dim ',1: dim(BootCube$BootCube)[[2]])
  #c('Dim1','Dim2')
```

```
GraphElli.rv <- MakeCIEllipses(BootCube$BootCube[,1:2,],
                  names.of.factors = c("dim 1","dim 2"),
                  col = col4Group,
                  p.level = .95)
a2d.gg.RVMap.CI <- a2b.gg.RVmap + gg.rv.means$zeMap_dots + GraphElli.rv
```

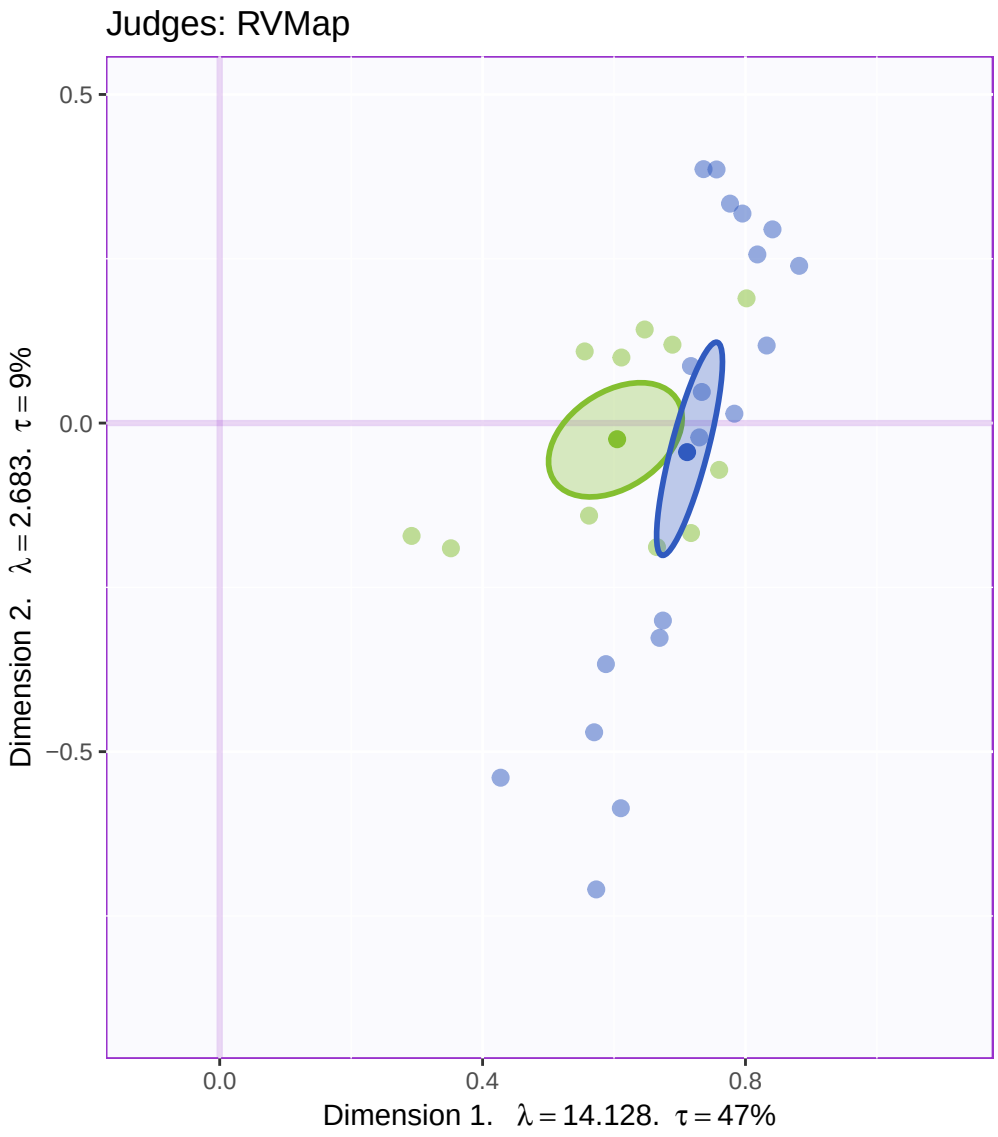
The values of the means of the assessors from the 2 groups of Judges can be seen in the table of means for the first three dimensions

```
knitr::kable(JudgesMeans[,1:3])
```

	dim 1	dim 2	dim 3
M	0.6047460	-0.0245617	-0.0857117
W	0.7112029	-0.0442066	0.0302969

To evaluate the significance of these differences, we plot the means and their bootstrapped derived confidence intervals on the R_V map.

```
print(a2d.gg.RVMap.CI )
```



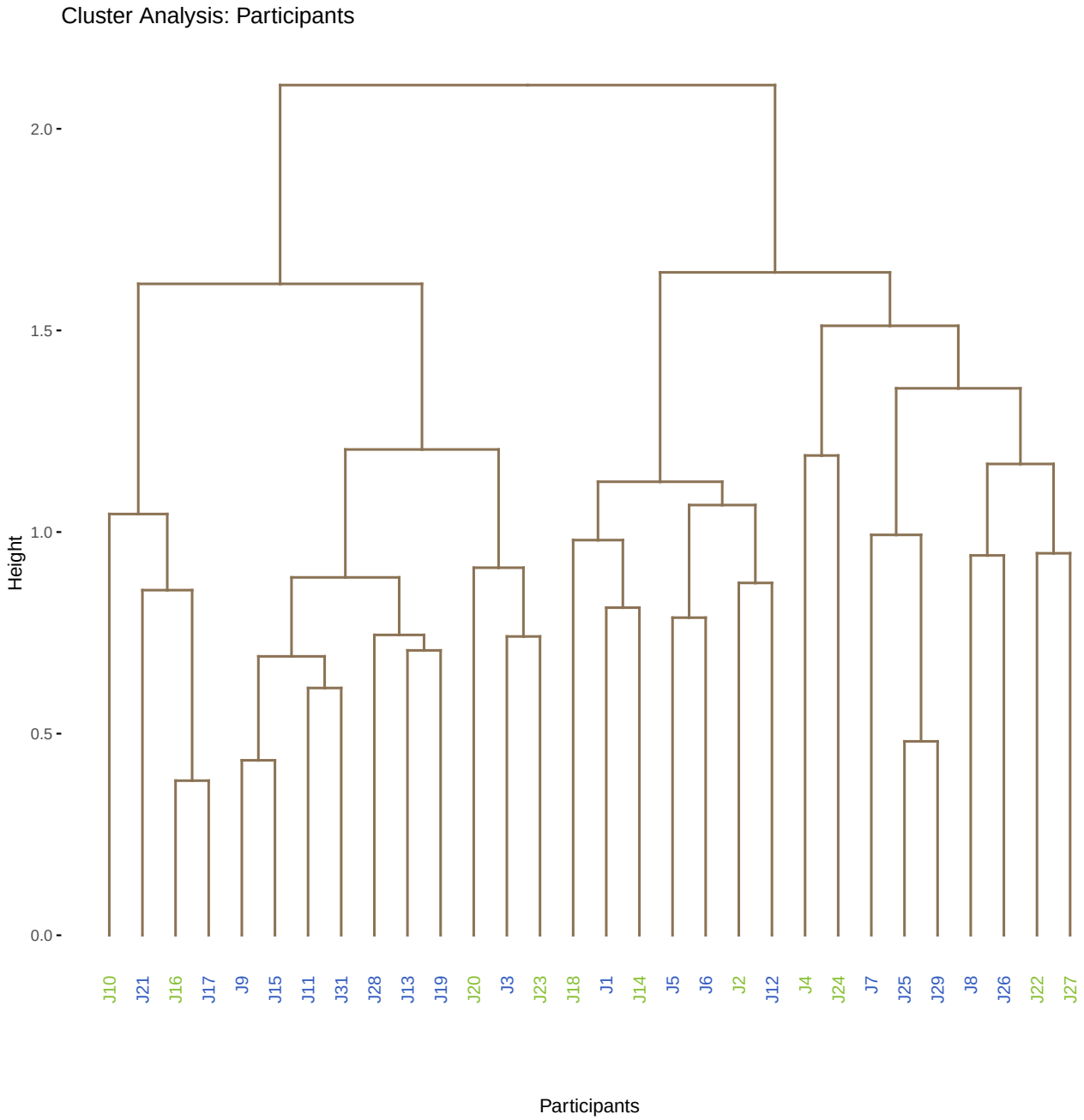
8.2 Post-Hoc Analysis: Exploring the individual participants between participants

8.2.1 Cluster Analysis

Here we conduct a hierarchical cluster analysis on the factor scores of the R_V matrix.

```
D <- dist(resDistatis$res4Cmat$G, method = "euclidean")
fit <- hclust(D, method = "ward.D2")
a05.tree4participants <- fviz_dend(fit,  k = 1,
                       k_colors = 'burlywood4',
                       label_cols = color4Judges.list$oc[fit$order],
                       cex = .7, xlab = 'Participants',
                       main = 'Cluster Analysis: Participants')
```

```
print(a05.tree4participants)
```



The cluster analysis suggests that there are groups of participants, but how many groups maybe more difficult to evaluate.

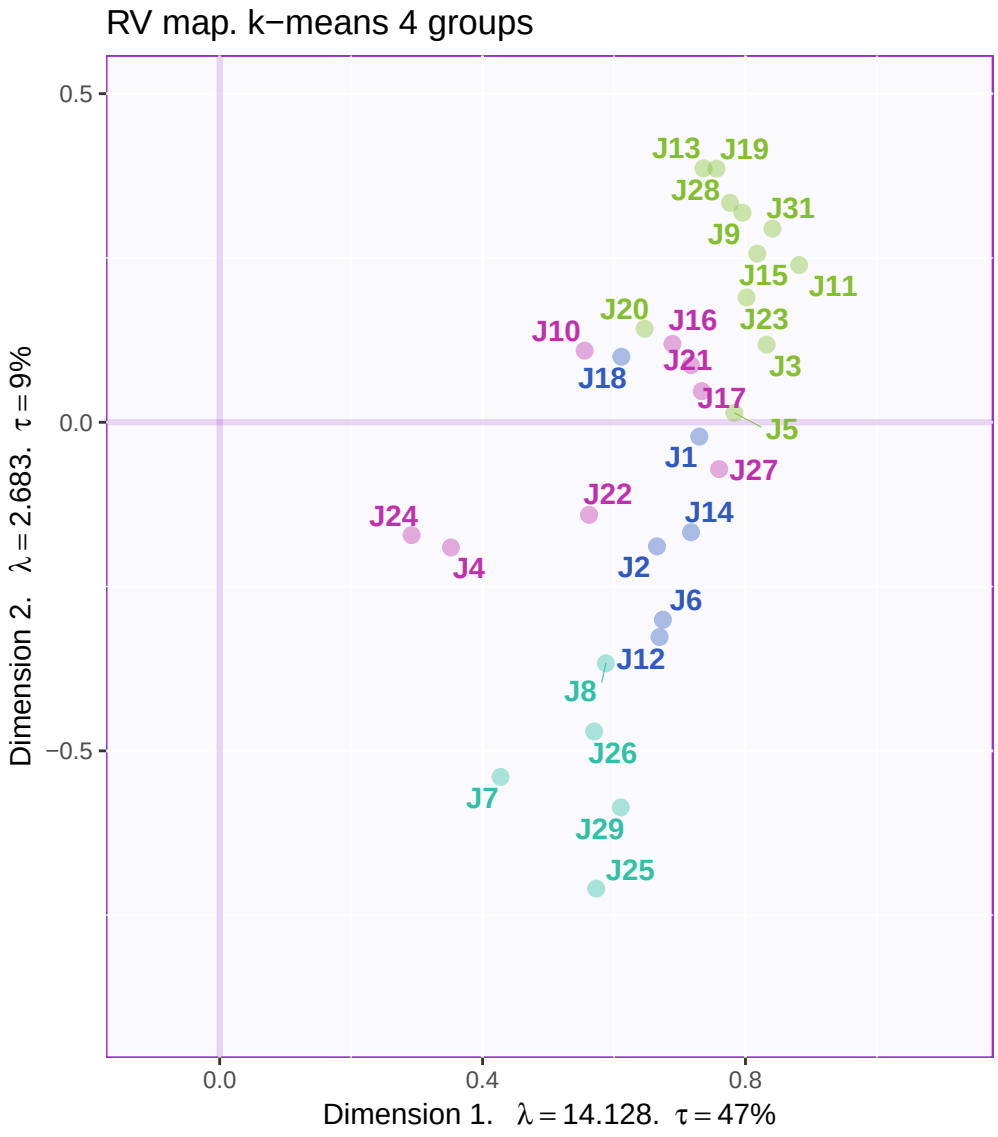
8.2.2 K-means

We try k-means with 4 centers

```
# First plain k-means
set.seed(42)
participants.kMeans <- kmeans(x = G , centers = 4)
#_____________________________________________________________
# Now to get a map by cluster:
col4Clusters  <- createColorVectorsByDesign(
```

```
              makeNominalData(
              as.data.frame(participants.kMeans$cluster)  ))
```

```
baseMap.i.km <- PTCA4CATA::createFactorMap(G,
                              title = "RV map. k-means 4 groups",
                                   col.points = col4Clusters$oc,
                                   col.labels = col4Clusters$oc,
                  constraints =    gg.rv.graph.out$constraints,
                                   alpha.points =  .4)
a06.aggMap.i.km <- baseMap.i.km$zeMap_background +
  baseMap.i.km$zeMap_dots + baseMap.i.km$zeMap_text +  labels4RV
print(a06.aggMap.i.km)
```

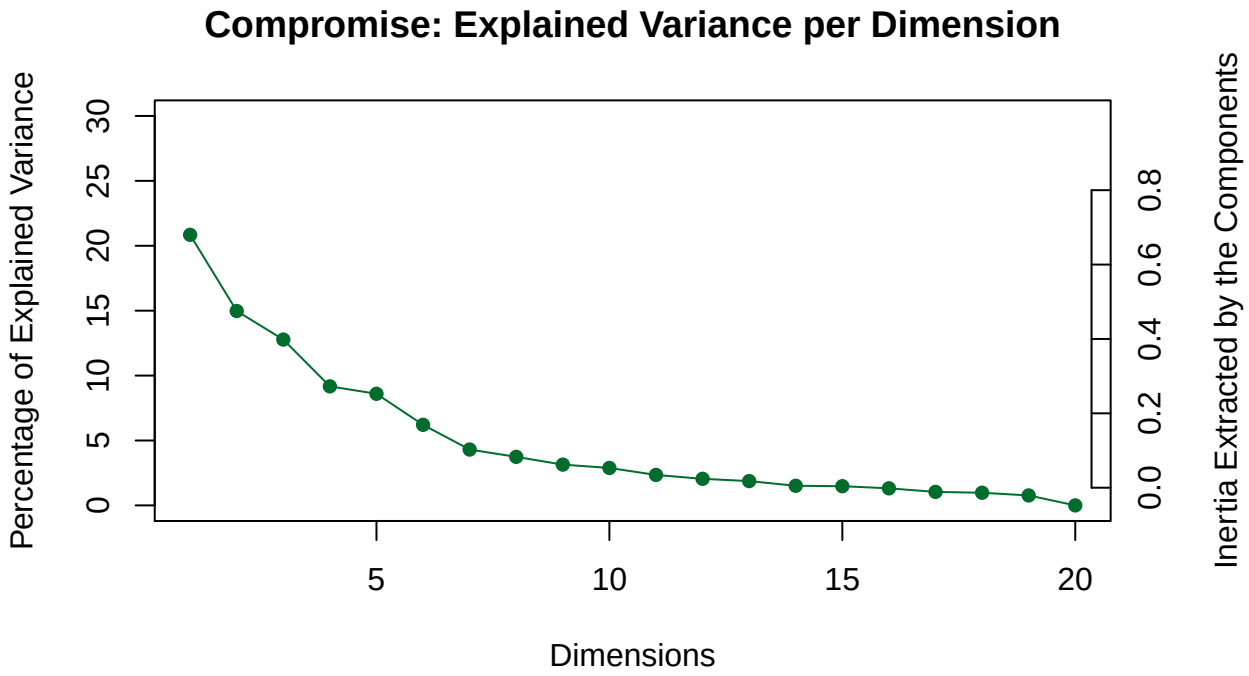


8.3 The compromise

8.3.1 Scree for the compromise

First a scree plot of the compromise

```
#------------------------------------------------------------------------
# A scree plot for the Compromise.
scree.S.out <- PlotScree(
              ev = resDistatis$res4Splus$eigValues,
              title = "Compromise: Explained Variance per Dimension")
```



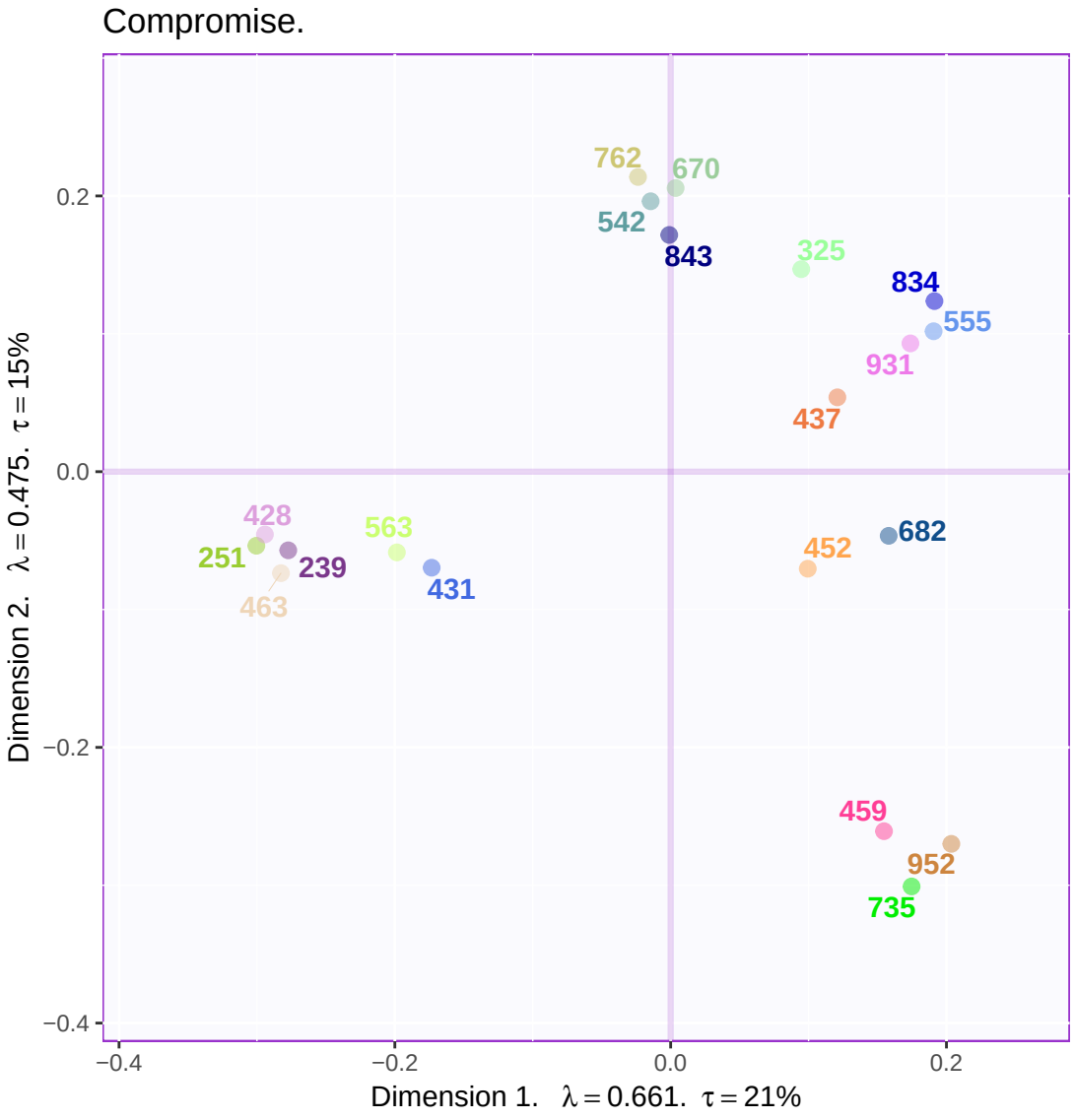

```
b1.Scree.S <- recordPlot()
#------------------------------------------------------------------------
```

8.3.2 The compromise

```
# 4.1 Get the bootstrap factor scores (with default 1000 iterations)
BootF <- BootFactorScores(resDistatis$res4Splus$PartialF)
# 5.2 a compromise plot
# General title for the compromise factor plots:
genTitle4Compromise = 'Compromise.'
# To get graphs with axes 1 and 2:
h_axis = 1
v_axis = 2
# To get graphs with say 2 and 3
# change the values of v_axis and h_axis
color4Products <- #  Create color for the Products from prettyGraph
 prettyGraphsColorSelection(n.colors = nrow(resDistatis$res4Splus$F))
gg.compromise.graph.out <- createFactorMap(resDistatis$res4Splus$F,
                                    axis1 = h_axis,
                                    axis2 = v_axis,
                                    title = genTitle4Compromise,
                                    col.points = color4Products ,
                                    col.labels = color4Products)
# NB for the lines below You need DISTATIS version > 1.0.0
#  to get the eigen values and tau for the compromise
label4S <- createxyLabels.gen(
```

```
            x_axis   = h_axis, y_axis = v_axis,
            lambda   = resDistatis$res4Splus$eigValues ,
            tau      = resDistatis$res4Splus$tau,
            axisName = "Dimension ")
b2.gg.Smap <-  gg.compromise.graph.out$zeMap + label4S
#
# 5.4 a bootstrap confidence interval plot
# 5.3  create the ellipses
gg.boot.graph.out.elli <- MakeCIEllipses(
                              data = BootF[,c(h_axis,v_axis),],
                              names.of.factors =
                                c(paste0('Factor ',h_axis),
                                  paste0('Factor ',v_axis)),
                              col = color4Products,
)
# Add ellipses to compromise graph
b3.gg.map.elli <- gg.compromise.graph.out$zeMap + gg.boot.graph.out.elli + label4S
#
```

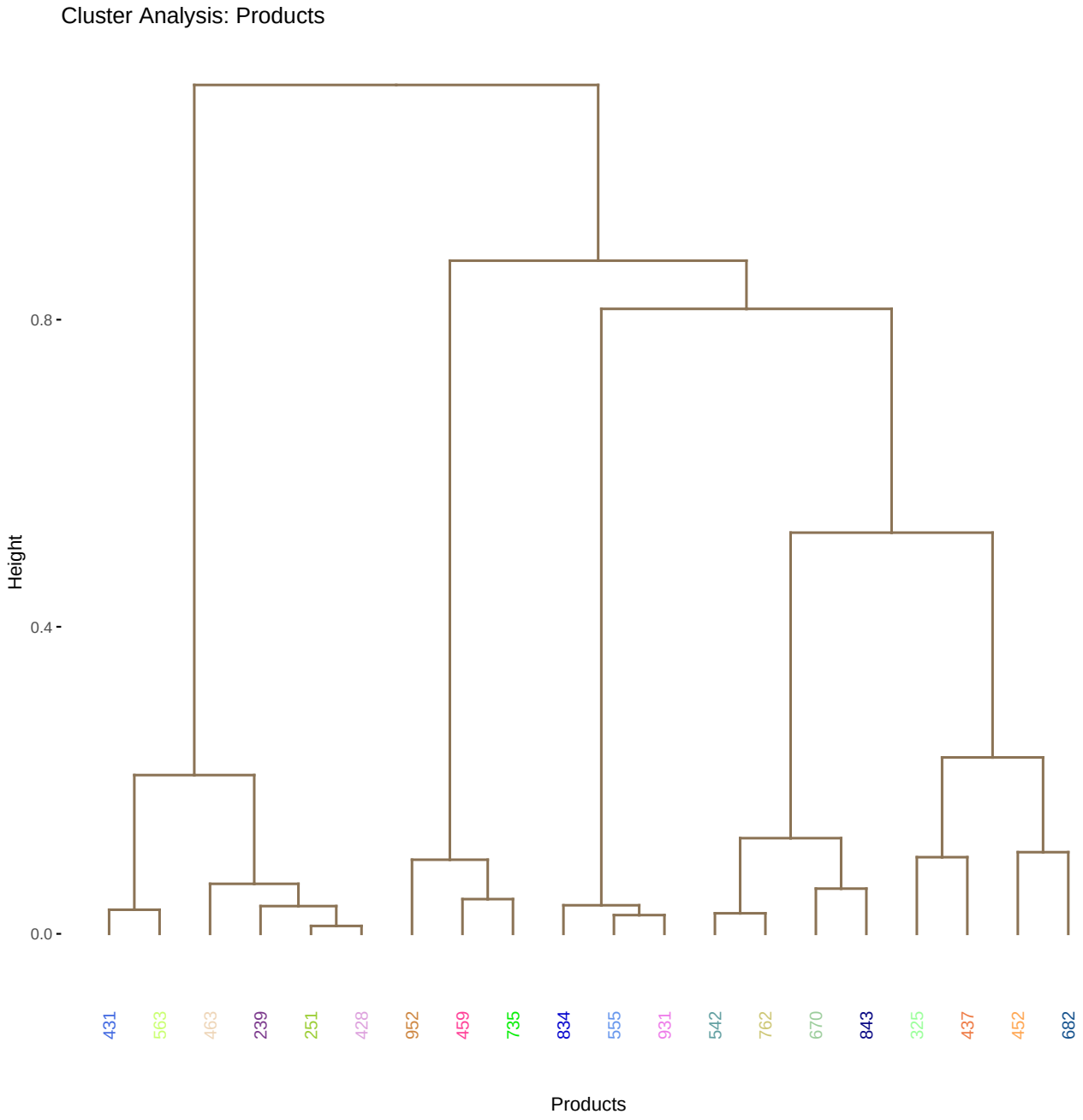
```
print(b2.gg.Smap)
```



A tree on top

```
nFac4Prod = 3
D4Prod <- dist(resDistatis$res4Splus$F[,1:nFac4Prod], method = "euclidean")
 fit4Prod <- hclust(D4Prod, method = "ward.D2")
 b3.tree4Product <- fviz_dend(fit4Prod,  k = 1,
                         k_colors = 'burlywood4',
                         label_cols = color4Products[fit4Prod$order],
                         cex = .7, xlab = 'Products',
                         main = 'Cluster Analysis: Products')
```

```
 print(b3.tree4Product)
```



8.4 Map of compromise with partial factor scores

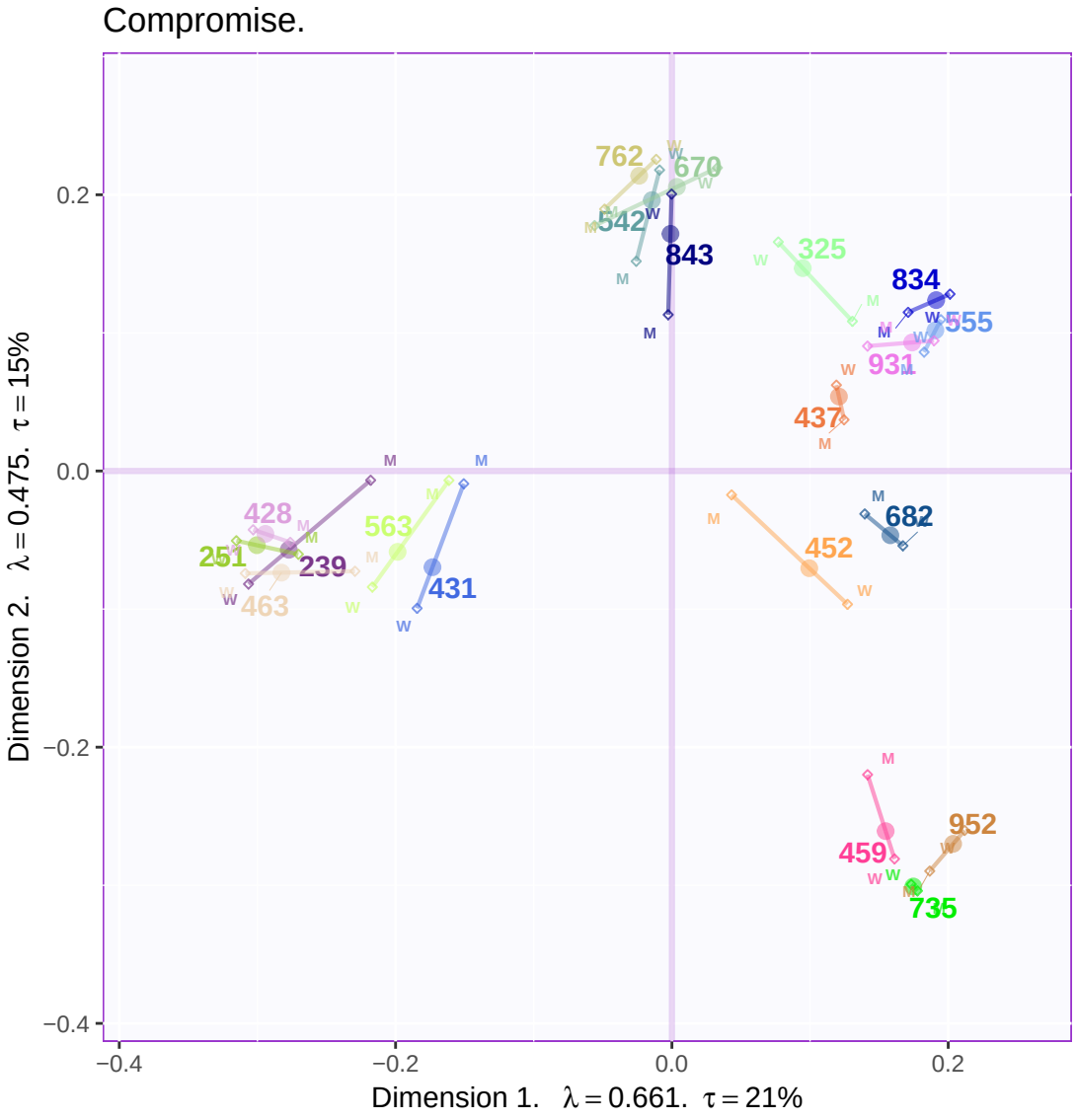
```
# get the partial map
map4PFS <- createPartialFactorScoresMap(
                        factorScores = resDistatis$res4Splus$F,
                         partialFactorScores = F_k,
                         axis1 = 1, axis2 = 2,
                         colors4Items = as.vector(color4Products),
                         names4Partial = dimnames(F_k)[[3]], #
                         font.labels = 'bold')

d1.partialFS.map.byProducts <- gg.compromise.graph.out$zeMap +
```

```
                                    map4PFS$mapColByItems + label4S
d2.partialFS.map.byCategories  <- gg.compromise.graph.out$zeMap +
                                    map4PFS$mapColByBlocks + label4S
```

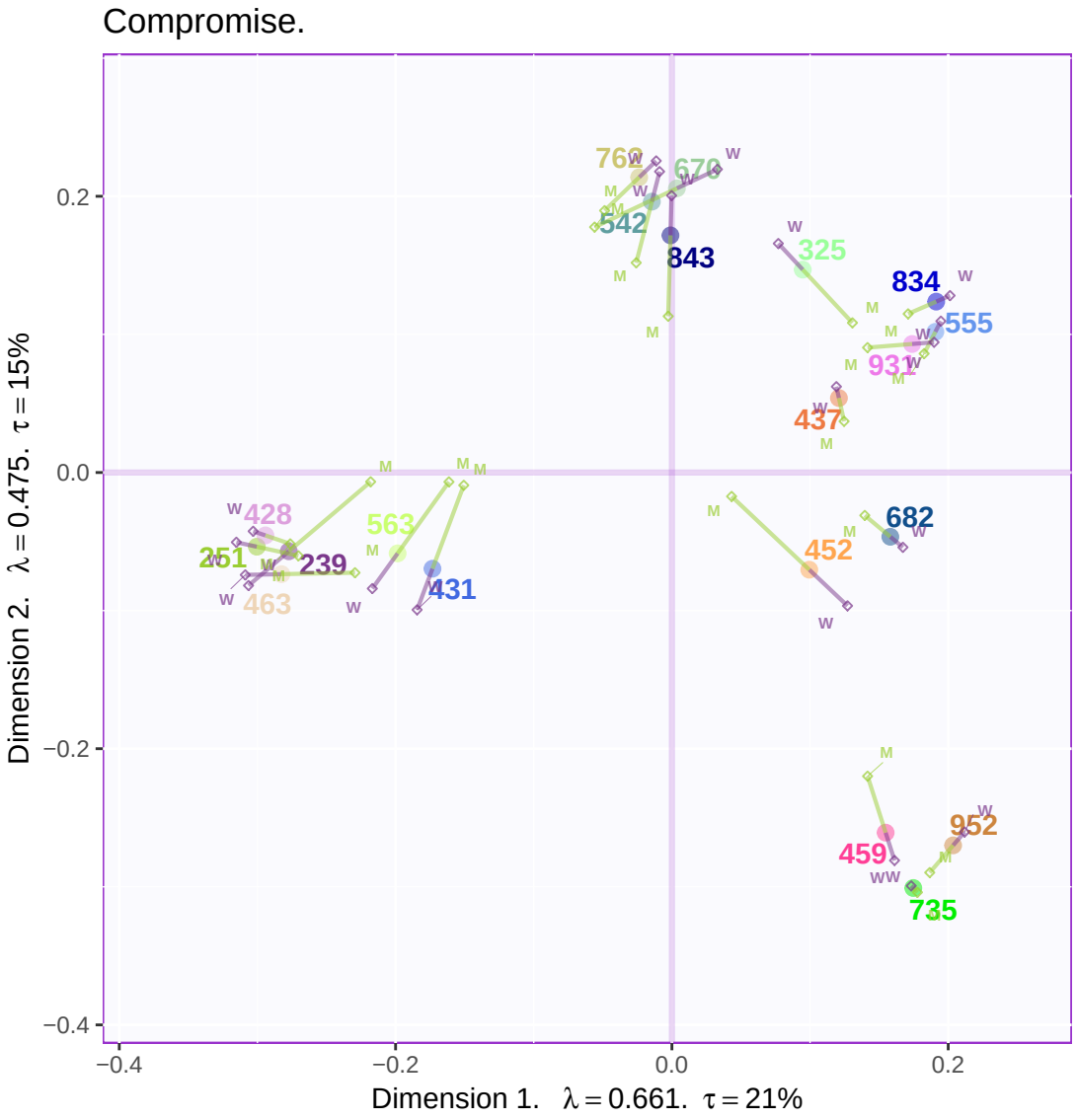
8.4.1 Map of Compromise with partial factor scores: colored by products

```
print(d1.partialFS.map.byProducts )
```



8.4.2 Map of Compromise with partial factor scores: colored by Judges' groups

```
print(d2.partialFS.map.byCategories)
```



8.5 Create vocabulary graphs

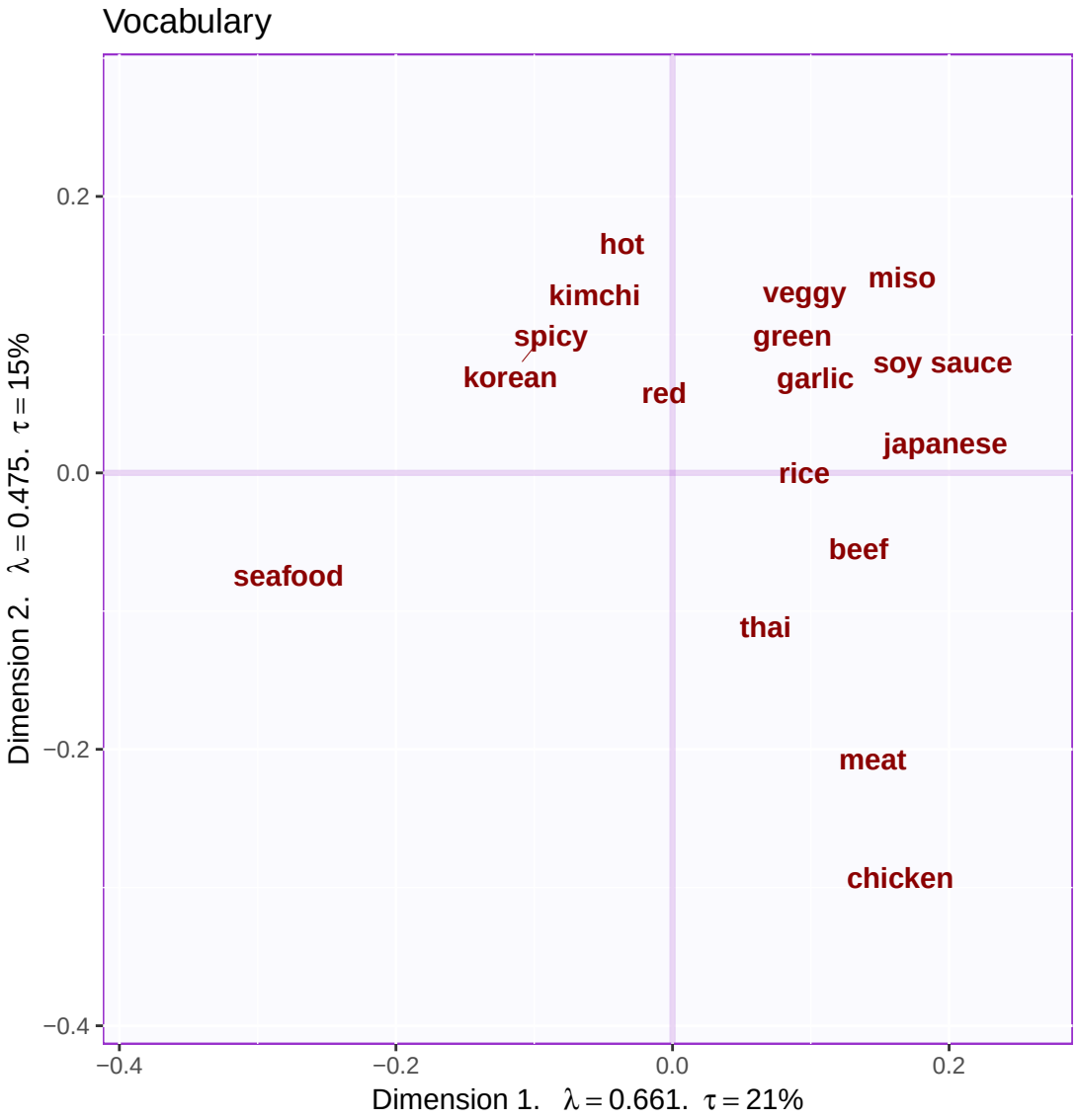
```
# 5.5. Vocabulary
# 5.5.2 CA-like Barycentric (same Inertia as products)
gg.voc.bary <- createFactorMap(F4Voc$Fvoca.bary,
                    title = 'Vocabulary',
                    col.points = 'red4',
                    col.labels = 'red4',
                    display.points = FALSE,
                    constraints = gg.compromise.graph.out$constraints)
#
e1.gg.voc.bary.gr <- gg.voc.bary$zeMap + label4S

#print(e1.gg.voc.bary.gr)
b5.gg.voc.bary.dots.gr <- gg.compromise.graph.out$zeMap_background +
                          gg.compromise.graph.out$zeMap_dots +
                          gg.voc.bary$zeMap_text + label4S
```

```
#print(gg.voc.bary.dots.gr)
```

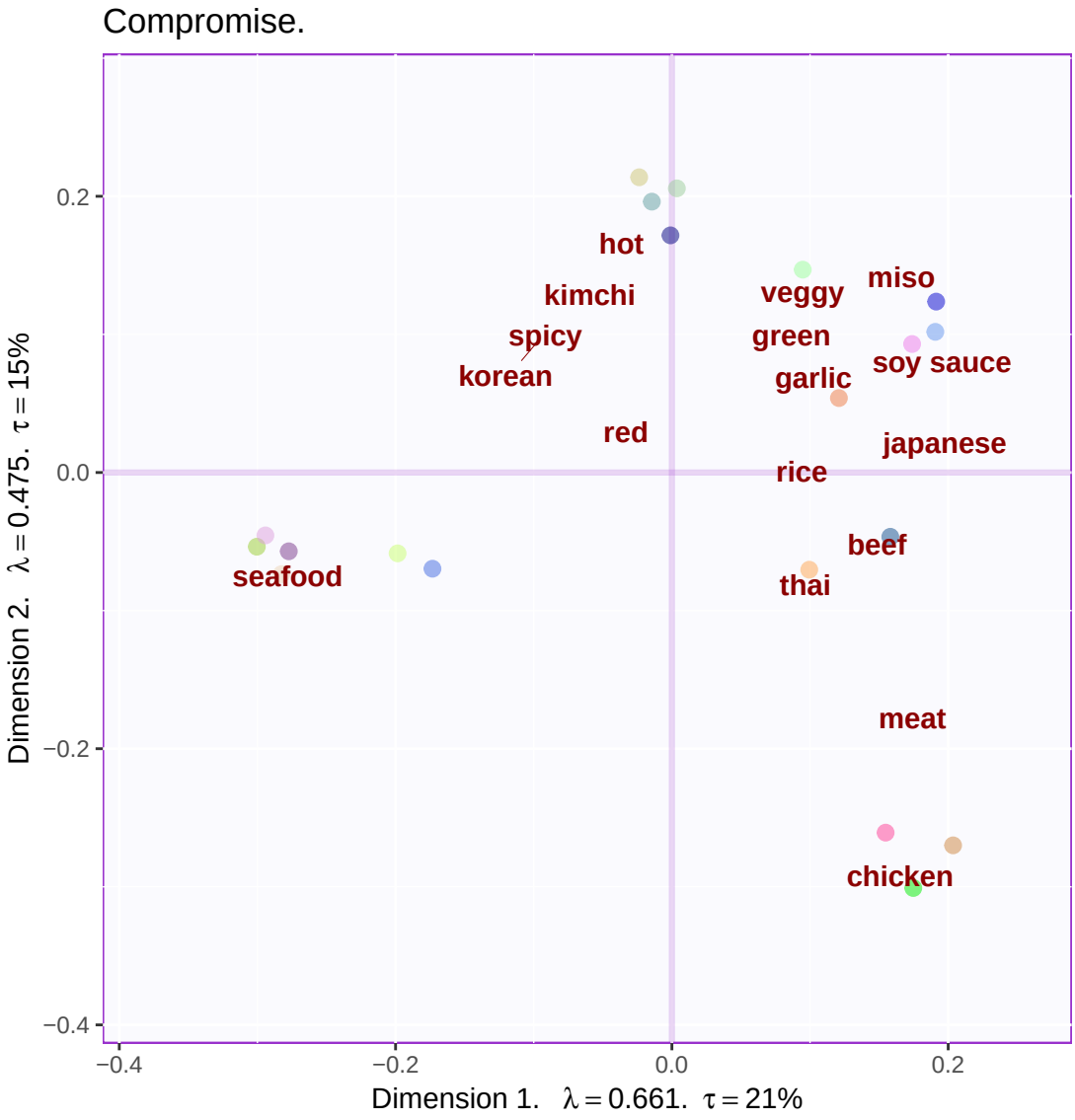
8.5.1 Print the graph vocabulary (without the products)

```
print(e1.gg.voc.bary.gr)
```



8.6 Print the graph vocabulary (with the products dots)

```
print(b5.gg.voc.bary.dots.gr)
```



9 Save the graphics as a PowerPoint

The graphics are saved as a PowerPoint with the following command

```
toto <- PTCA4CATA::saveGraph2pptx(file2Save.pptx = name4Graphs,
                  title = '30 (mostly Asian) participants sort pictures of 20 Ramen noodles',
                  addGraphNames = TRUE)
```

Note that we could also have created a PowerPoint with `Rmarkdown` by using the following options in the preamble:

```
output:
      powerpoint_presentation:
           slide_level: 4
```

instead of (for example):

```
output:
       rmarkdown::html_vignette:
```

```
        toc: true
        number_sections: true
```