Stat 243: Final Project, Due Friday Dec-02

October 31, 2016

Abstract

The final project involves creating an R package that implements a statistical multivariate technique called Multiple Factor Analysis. You will have to form groups of four members, and work together to complete the project.

An R package for Multiple Factor Analysis

The goal of the project is to create an R package that implements Multiple Factor Analysis (MFA). The package should have unit tests, as well as vignette(s). To provide a companion interactive visualization tool for the package you should also create a shiny app. In addition, create html slides that you would use to give a presentation (e.g. a tutorial or seminar) about MFA, and your package.

To work on this project please read the paper *Multiple Factor Analysis: principal component analysis for multitable and multiblock data sets*, by Herve Abdi, Lynne Williams, and Dominique Valentin.

https://www.utdallas.edu/~herve/abdi-WiresCS-mfa-2013.pdf

Additionally, you can also take a look at *Multiple Factor Analysis: main features and application to sensory data* by Jerome Pages.

http://factominer.free.fr/docs/PagesAFM.pdf

Data set

The working example consists of a (fictitious) wine tasting experiment. The data concerns 12 wines made from Sauvignon Blanc grapes coming from three wine regions (4 wines from each region): New Zealand, France, and Canada. Ten expert assessors were asked to evaluate these wines. The assessors were asked:

1. to evaluate the wines on 9-point rating scales, using four variables considered as standard for the evaluation of these wines: cat-pee, passion-fruit, green pepper, and mineral.
2. if they felt the need, to add some variables of their own: some assessors choose none, some choose one or two more variables.

The raw data is available in the file `wines.csv` inside the `final-project/data/` folder:

https://github.com/ucb-stat243/stat243-fall-2016/tree/master/problem-sets/final-project

Function to compute MFA

The package should provide a main function, e.g. `mfa()`—or you can give it a different name—with the following usage:

```
mfa(data, sets, ncomps = NULL, center = TRUE, scale = TRUE)
```

where the arguments are:

- `data`: data set (matrix or data frame).
- `sets`: list of vectors indicating the sets of variables (i.e. the blocks).
- `ncomps`: integer indicating how many number of components (i.e. factors) are to be extracted.
- `center`: either a logical value or a numeric vector of length equal to the number of *active* variables in the analysis
- `scale`: either a logical value or a numeric vector of length equal to the number of *active* variables in the analysis

with the following details:

- You should define your function in such a way that the user is able to pass data in the form of a data.frame or a matrix.
- The argument `sets` should handle a list of character vectors with the names of the active variables, or a list of numeric vectors with the position of the active variables in the data table.
- By default, `ncomps = NULL` indicate that all possible components should be extracted. But the user can specify a number e.g. `ncomps = 2`.
- To handle the centering-and-scaling, I recommend you to use the function `scale()` so that you can pass it the arguments `center` and `scale`.
- The value of `center` determines how column centering is performed. If `center` is a numeric vector with length equal to the number of active variables, then each variable has the corresponding value from `center` subtracted from it. If `center = TRUE` then centering is done by subtracting the column means (omitting NAs), and if `center = FALSE`, no centering is done.
- The value of `scale` determines how the scaling of active variables is performed (after centering). If `scale` is a numeric vector, then each active variable is divided by the corresponding value from `scale`. If `scale = TRUE` then scaling is done by dividing the (centered) variables by their standard deviations if `center = TRUE`, and the root mean square otherwise. If `scale = FALSE`, no scaling is done.

The function should return an object of class `"mfa"` with the following elements:

- vector containing the eigenvalues: these are denoted as λ in the paper, computed from the GSVD of $\mathbf{X} = \mathbf{P}\boldsymbol{\Delta}\mathbf{Q}^\mathsf{T}$.
- matrix of common factor scores—a.k.a. compromise factor scores. An example of this output is in the matrix $\mathbf{F}$ of eq. 65 (page 15).
- matrix of partial factors scores. An example of this output is in the matrix $\mathbf{F}_{[\mathbf{1}]}$ of eq. 66 (page 15).
- matrix of loadings (a.k.a. factor loadings). This is the matrix $\mathbf{Q}$ in $\mathbf{X} = \mathbf{P}\boldsymbol{\Delta}\mathbf{Q}^\mathsf{T}$, i.e. the right singular values. See table 3 (page 16).

Generic Methods

Write a `print()` method for `"mfa"` objects. The printing method should display basic information. Avoid printing the entire list containing the vector of eigenvalues, and the three matrices with the scores and loadings.

Write plotting auxiliary functions that, given two dimensions (e.g. 1 and 2), a graphic is displayed of:

- the compromise of the tables: see figure 2a) (page 18)
- partial factor scores
- variable loadings

Related Methods and Functions

To provide more functionality to your package, you will have to create a set of complementary functions and methods to compute:

- summaries of eigenvalues
- contributions
- R_V coefficient

Eigenvalues

One of the outputs of MFA involves summarizing information about the obtained eigenvalues (often referred to as explained inertia). An example of this type of table is in Table 2 (see page 15).

Write a function that takes the `"mfa"` object and returns a table (like Table 2) with the singular values (i.e. square root of eigenvalues), the eigenvalues, cumulative, percentage of intertia, cumulative percentage of inertia, for all the extracted components.

Contributions

Another derived output from MFA is a series of descriptive statistics called *contributions*. These values help us interpret how 1) the observations, 2) the variables, and 3) the tables, contribute to the variability of the extracted dimensions:

1. Contribution of an observation to a dimension. See the formula in equation 25 (page 7).
2. Contribution of a variable to a dimension. See the formula in equation 27 (page 7). An example of these contributions is inside the table 3 (page 16).
3. Contribution of a table to a dimension. See the formula in equation 28 (page 7). An example of this type of contributions is in the matrix $ctr_{k,l}$ of eq. 67 (see page 15).

Write functions to compute each type of contribution. These functions should take an object of class `"mfa"` as the main argument, and return a matrix with the corresponding contribution values.

Coefficients to study the Between-Table Structure

- To evaluate the similarity between two tables we use the R_V coefficient. See equation 29 (page 8). You should create a function that computes this coefficient for two tables. The usage of such function could be something like this:

```
RV(table1, table2)
```

- In addition, write another function that takes a data set, and a list with `sets` of variables, and which returns a matrix of R_V coefficients. For example, if you provide a data that comprises three sets or blocks, then the output should be a 3×3 matrix with the R_V coefficients between each pair of tables. The usage of this function could be something like:

```
RV_table(dataset, sets = list(1:3, 4:5, 6:10))
```

In this case the first table would be formed by columns 1, 2 and 3. The second table would be formed by columns 4 and 5. And the third table would be formed by columns 6 to 10. The output should be a symmetric matrix with 3 rows and 3 columns containing the RV coefficients.

Extra-credit: L_g Coefficient

Optionally, you can try to write a function that computes the L_g coefficient between two tables. See equation 30 (page 8). And also a function to compute a table of L_g coefficients, similar to the table of R_V coefficients.

Extra-credit: Bootstrap

Optionally, you can try to write a function that allows the user to perform bootstrapping in order to estimate the stability of the compromise factor scores. See the bootstrap approach described in page 11 of the paper, as well as in pages 18-21.

Shiny App

As mentioned in the introduction, you should also design a shiny app to provide an interactive visualization tool for (some of) the results from MFA.

You can create a shiny app to make plots like those shown in figure 2, without the glass of wine icons. And/or plots like those shown in figure 3, again without the glass of wine icons and the men/women icons.

The main idea is to use the plotting functions and use them in your app. You can have a widget to select what to plot:

- maybe a bar-chart for the eigenvalues
- a scatterplot of the common factor scores
- a scatterplot of the partial factors scores.
- a scatterplot of theloadings

You can add as many widgets as you want, and you can also expriment with the User Interface layouts. Get inspired by the examples in the RStudio Shiny gallery:

http://shiny.rstudio.com/gallery/

Publish and share your shiny app using shinyapps.io

Submission

To submit all the materials of your project (one submission per team) you can use:

- a github repository (for those of you familiar with git and github).
- an Open Science Framework (OSF) project: https://osf.io/.
- a dropbox folder