A Tutorial for Bayesian Integrative Factor Models

Mavis Liang

2024-10-10

Table of contents

Preface

This is the tutorial to guide statisticians to use Baysian integrative factor models.

```r
1 + 1
```

```
[1] 2
```

1 Preliminary

1.1 Factor models and multi-study setting

This chapter introduce the theory behind factor models

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

2 Quick start

Step 1: Prepare the package and data

A small simulated data can be downloaded here:

RDS format

Step 2: Run BMSFA

Step 3: Post-processing

Step 4: Visualization

```r
1 + 1
```

```
[1] 2
```

3 Package Installation

I have this separate section to introduce the package installation because installing some packages requires extra efforts.

3.1 Stack FA, Ind FA, and BMSFA

3.2 PFA

PFA does not provide any downloadable R packages and we need to download the R scripts from their GitHub repository, put them in the same directory as the main script, and source them for use.

We only need the three files: `FBPFA-PFA.R`, `FBPFA-PFA with fixed latent dim.R`, and `PFA.cpp`, which can be found in https://github.com/royarkaprava/Perturbed-factor-models. The `FBPFA-PFA.R` file contains the full Bayesian inference algorithm for the PFA model, directly set the latent dimensions equal to the dimensions or the original data. The `FBPFA-PFA with fixed latent dim.R` file contains the same algorithm that requires to set numbers of common factors K. We also notice that two version of the models are both `PFA()`, and some functions in the `FBPFA-PFA with fixed latent dim.R` file depends on the `FBPFA-PFA.R` file. Therefore, since we want to run the dimension reduction version of the model, we must source the `FBPFA-PFA.R` file first, and then source the `FBPFA-PFA with fixed latent dim.R` file.

```r
# Suppose the files are in the same directory as the main script
source("FBPFA-PFA.R")
```

```
Warning: package 'expm' was built under R version 4.4.2
```

```
Loading required package: Matrix
```

```
Attaching package: 'Matrix'
```

```
The following objects are masked from 'package:pracma':

    expm, lu, tril, triu


Attaching package: 'expm'

The following object is masked from 'package:Matrix':

    expm

The following objects are masked from 'package:pracma':

    expm, logm, sqrtm

Warning: package 'RcppArmadillo' was built under R version 4.4.2
```

```r
source("FBPFA-PFA with fixed latent dim.R")
```

3.3 MOM-SS

3.4 SUFA

3.5 Tetris

4 Case study: nutrition data

4.1 Loading and previewing the data

The data used in this section is from... This data is not publicly available. Please contact the authors of the original study for access.

```r
load("./Data/dataLAT_projale2.rda")
```

The resulting object is a list of 6 data frames, each corresponding to a different study. Each data frame contains information about the nutritional intake of individuals, and the columns represent different nutrients. From Study 1 to Study 6, the number of individuals (N_s) are 1364, 1517, 2210, 5184, 2478, and 959, respectively, and the number of nutrients (P) are all 42.

```r
# Check how many studies in the list
length(X_s2)
```

```
[1] 6
```

```r
# Dimension of each study
lapply(X_s2, dim)
```

```
[[1]]
[1] 1364   42

[[2]]
[1] 1517   42

[[3]]
[1] 2210   42

[[4]]
[1] 5184   42
```

```
[[5]]
[1] 2478   42

[[6]]
[1] 959  42
```

Let's take a look at the first few rows of the first data frame to get an idea of the data structure.

```r
X_s2[[1]][1:5, 1:5]
```

```
  Animal Protein (g) Vegetable Protein (g) Cholesterol (mg)  SCSFA MCSFA
1            28.9560               14.7440          256.761 0.2665 0.939
2            33.6675                8.9710          104.217 0.2180 0.520
3            70.0000               31.0635          207.902 0.9845 1.692
4            20.6700               13.8240          148.921 0.0625 0.239
5            15.4250               10.5550           65.060 0.0090 0.033
```

We note that the data we have available is different from the original data (cite). The original data is a collection of 12 studies, and there are known covariates for each individuals, like the one we simulated in the previous section. However, for the purpose of this case study, the data we used are collapsed into 6 studies, and only the nutritional intake data are available.

4.2 Data preprocessing

Some individuals have missing values for all nutrients, and we will remove these individuals from the data. We will also replace negative values with 0, since nutrition intake cannot be less than 0. Then apply a log transformation to the data.

We first count how many NA values and negative values are in each study.

```r
count_na_and_negatives <- function(df) {
  # Count NA values
  na_count <- sum(is.na(df))
  # Count negative values
  negative_count <- sum(df < 0, na.rm = TRUE)

  # Print counts
  cat("Number of NAs:", na_count, "\n")
  cat("Number of negative values:", negative_count, "\n")
}
invisible(lapply(X_s2, count_na_and_negatives))
```

```
Number of NAs: 1344
Number of negative values: 0
Number of NAs: 1344
Number of negative values: 1
Number of NAs: 1344
Number of negative values: 0
Number of NAs: 1344
Number of negative values: 2
Number of NAs: 1344
Number of negative values: 1
Number of NAs: 1344
Number of negative values: 0
```

We will define a function to process the data, which removes rows where all values are NA. We also define a function that replaces negative values with 0, and applies a log transformation to the data.

```r
process_study_data <- function(df) {
  # Remove rows where all values are NA
  cleaned_df <- df[!apply(df, 1, function(row) all(is.na(row))), , drop = FALSE]
  # Count remaining rows
  remaining_rows <- nrow(cleaned_df)
  # Print results for the study
  cat("Remaining rows:", remaining_rows, "\n")
  return(cleaned_df)
}

Y_list <- lapply(X_s2, process_study_data)
```

```
Remaining rows: 1332
Remaining rows: 1485
Remaining rows: 2178
Remaining rows: 5152
Remaining rows: 2446
Remaining rows: 927
```

```r
# Replace negative values with 0, then log(x+0.01) + 0.01
replace_negatives <- function(df) {
  # Replace negative values with 0
  df[df < 0] <- 0
  # Apply log transformation. Add 0.01 to avoid log(0).
  transformed_df <- log(df + 0.01)
  return(transformed_df)
}

Y_list <- lapply(Y_list, replace_negatives)
```

The numbers of individuals in each study left for analysis (N_s) are 1332, 1485, 2178, 5152, 2446, and 927, respectively.

```r
# Check the processed data
invisible(lapply(Y_list, count_na_and_negatives))
```

```
Number of NAs: 0
Number of negative values: 11910
Number of NAs: 0
Number of negative values: 11222
Number of NAs: 0
Number of negative values: 15006
Number of NAs: 0
Number of negative values: 36230
Number of NAs: 0
Number of negative values: 19349
Number of NAs: 0
Number of negative values: 6707
```

Now we don't have any NA values or negative values in the data.

For some models, we will need to standardize the data. We will standardize the data for each study separately.

4.3 Model fitting

4.4 Post processing

4.5 Visualization

4.6 Mean squared error (MSE)

5 Summary

In summary, this book has no content whatsoever.

```r
1 + 1
```

```
[1] 2
```

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.