www.quant.ku.edu

KUANT Guides

Guide No.
KUANT

SEM with Lavaan 0.5-9

A guide introducing the R package lavaan for structural equation modeling.

Created: Patrick Miller (2011)

Updated: Terry Jorgensen and Sunthud Pornprasertmanit (2012)

Introduction

R is an open source software environment for statistical computing, and one of its primary benefits is the vast number of available packages developed for advanced statistics and data analysis. Within the R programming environment there are several packages designed for latent variable analysis (including `OpenMx`, `ltm`, and `sem`), but in this guide we will be looking at a relatively new and user-friendly package called lavaan, throughout which I assume a basic knowledge of R.

While nothing beats consulting the lavaan documentation itself, in this guide I will point out the most salient features of the software. The authors of lavaan also provide a helpful, readable user's guide as well as the more technical official software documentation (see references).

First Steps

If you have not yet installed R, follow this link: http://cran.r-project.org/

Using installation defaults should generally be sufficient for most users.

To install lavaan, open R and type the following command at the prompt:

```
> install.packages("lavaan")
```

Select a CRAN mirror when prompted (any in the US should be sufficient). Load the package by typing the following:

```
>  library(lavaan)
```

You should see a warning telling you that it is beta software, which you should heed. It is reliable, but still beta.

Getting Help with lavaan

- Examples are provided on the web site: http://lavaan.ugent.be/
- Examples are also available in the creator's paper about lavaan, published in the *Journal of Statistical Software* (please cite this paper if you publish results analyzed using lavaan):

  http://www.jstatsoft.org/v48/i02

- Find help files in R by typing on the command line: `help(package = lavaan)`

A Simple Example

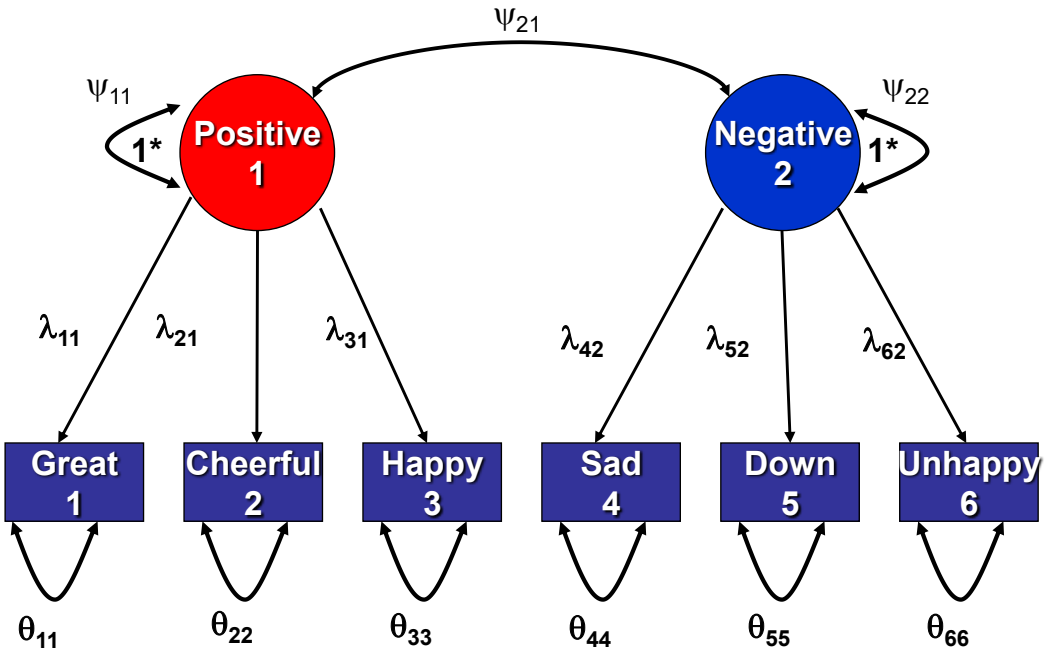
First, we need to read the data into R. In this case, the data file ("posneg.dat") is a full data file with 6 variables.

Assuming that the data file is in the same working directory as R, we can use the following commands to store the data matrix in the object "dat" as a data frame, name the variables, and then verify the results.

```
> dat <- read.table("posneg.dat", header = FALSE)
> names(dat) <- c("great", "cheerful", "happy", "sad", "down", "unhappy")
> head(dat)

   great cheerful  happy    sad down unhappy
1 3.5000   4.0000 4.0000 4.0000  4.0       4
2 2.5000   3.1667 3.0000 3.2123  2.0       3
3 1.8333   2.0000 1.5000 3.0000  3.0       2
4 2.7714   3.0602 2.3639 3.1337  4.0       3
5 3.1667   3.3333 2.8333 3.5000  4.0       4
6 2.3333   2.8333 2.3333 3.0000  2.5       3
```

As in other syntax guides, we will use the following model as an example. Positive and negative affect are each latent constructs and are each indicated by three latent variables.



We make the lavaan model statement and store it in the vector *PosModel*. Note that the variable names in the model statement correspond to the variable names in our data frame.

```
> PosModel <- "Positive =~ great + cheerful + happy
               Negative =~ sad + down + unhappy   "
```

The "=~" means "is loaded by". To find the fit of this CFA model, call the lavaan function `cfa()` and store the results in a vector. In this example, we standardize the latent variances to equal 1 in order to identify the model (as opposed to the default, which is to fix the first loading to equal 1). This is specified in the `cfa()` argument "`std.lv = TRUE`".

```
> Results <- cfa(PosModel, data = dat, std.lv = TRUE)
```

To examine the results, we will use the function `summary()`

```
> summary(Results)

lavaan (0.5-9) converged normally after  43 iterations

  Number of observations                           380

  Estimator                                         ML
  Minimum Function Chi-square                   10.708
  Degrees of freedom                                 8
  P-value                                        0.219

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  Positive =~
    great             0.465    0.021   22.474    0.000
    cheerful          0.492    0.021   23.768    0.000
    happy             0.498    0.022   22.870    0.000
  Negative =~
    sad               0.529    0.039   13.477    0.000
    down              0.579    0.044   13.314    0.000
    unhappy           0.626    0.039   16.232    0.000

Covariances:
  Positive ~~
    Negative          0.507    0.047   10.829    0.000

Variances:
    great             0.049    0.005
    cheerful          0.036    0.005
    happy             0.050    0.005
    sad               0.311    0.030
    down              0.388    0.037
    unhappy           0.200    0.030
    Positive          1.000
    Negative          1.000
```

To include fit measures and the standardized solution, use the `fit` and `standardized` arguments:

```
> summary(Results, fit = TRUE, standardized = TRUE)

lavaan (0.5-9) converged normally after  43 iterations

  Number of observations                           380

  Estimator                                         ML
  Minimum Function Chi-square                   10.708
  Degrees of freedom                                 8
  P-value                                        0.219

Chi-square test baseline model:

  Minimum Function Chi-square                 1416.263
  Degrees of freedom                                15
  P-value                                        0.000

Full model versus baseline model:

  Comparative Fit Index (CFI)                    0.998
  Tucker-Lewis Index (TLI)                       0.996

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)              -1545.377
  Loglikelihood unrestricted model (H1)      -1540.023

  Number of free parameters                         13
  Akaike (AIC)                                3116.754
  Bayesian (BIC)                              3167.976
  Sample-size adjusted Bayesian (BIC)         3126.730

Root Mean Square Error of Approximation:

  RMSEA                                          0.030
  90 Percent Confidence Interval          0.000  0.071
  P-value RMSEA <= 0.05                          0.745

Standardized Root Mean Square Residual:

  SRMR                                           0.018
(continued...)
```

In summary, the entire model can be run from raw data in the following four commands:

```
> data <- read.table("posneg.dat", header = TRUE)
> PosModel <- 'Positive =~ X1 + X2 + X3
               Negative =~ X7 + X8 + X9'
> Results <- cfa(PosModel, data = dat, std.lv = TRUE)
> summary(Results)
```

Lavaan Model Syntax

Main Operators

With the above principles of the R programming language established, using lavaan is straightforward. Structural models are specified with regression like syntax:

```
Y1 ~ X1 + X2 + F1 + F2
```

The "~" operator represents "is regressed on". The left hand side represents dependent variable. The right hand side represents independent variables.

Latent variables are defined as:

```
F1 =~  X4 + X5 + X6
F2 =~  X7 + X8 + X9
```

The "=~" operator represents "is loaded by". The left hand side represents latent variable. The right hand side represents manifest variables.

To express the correlation between two residual variances or two factors:

```
X4 ~~ X5
```

The "~~" operator represents "is correlated with".

Intercepts:

```
X1 ~ 1
F1 ~ 1
```

To build the model in its entirety, simply put enclose all of these statements in quotes " " and assign it to a vector. There are four other operators: ":=" (is defined by), "==" (is equal to), "<" (is smaller than), and ">" (is greater than).

Free or Fixed Parameters

lavaan provides some defaults option that, sometimes, users may be not aware of it. For example, the first manifest variable loading is fixed as 1 by default. Therefore, users need to know how to fix or free parameters. To fix a parameter, users simply put a number with an asterisk in front of a target variable:

```
> PosModel <- 'Positive =~ 1*V1 + V2 + V3
              Negative =~ 1*V4 + V5 + V6'
```

To free a parameter, users simply put an NA with an asterisk in front of a target variable:

```
> PosModel <- 'Positive =~ 1*V1 + NA*V2 + NA*V3
              Negative =~ 1*V4 + NA*V5 + NA*V6'
```

For example, a fixed-factor-approach syntax can be specified:

```
> PosModel <- 'Positive =~ NA*V1 + NA*V2 + NA*V3
              Negative =~ NA*V4 + NA*V5 + NA*V6
              Positive ~~ 1*Positive
              Negative ~~ 1*Negative   '
```

lavaan actually provides some functions to automatically set the default as a fixed-factor approach. The syntax approach is the most flexible approach to specify models.

Equality Constraints

The equality constraints can be done in many ways. The easiest way is to specify a label on each parameter estimates. The label can be attached to a target parameter by adding a label with an asterisk in front of a variable name of target parameter in the syntax:

```
> PosModel <- 'Positive =~ V1 + eq1*V2 + eq1*V3
              Negative =~ V4 + V5 + V6 '
```

The variable with the same labels are equally constrained. In this case, the factor loading from the "Positive" factor on Variables 2 and 3 are equally constrained. Users may fix/free and label parameters simultaneously by specify the target parameter one more time:

```
> PosModel <- 'Positive =~ V1 + eq1*V2 + 1*V2 + eq1*V3
              Negative =~ V4 + V5 + V6 '
```

Fitting Model

The `cfa()` function has been introduced to fit a confirmatory factor analysis. There are two more useful functions to fit models: `growth()` for latent curve model and `sem()` for structural equation modeling. Here are some useful arguments of these functions:

- `model` The lavaan syntax
- `data` The target data to be used
- `sample.mean` The vector of sample means (if the data argument is not used)
- `sample.cov` The covariance matrix (if the data argument is not used)
- `sample.nobs` The number of observations
- `group` The variable name of grouping variable
- `group.equal` The set of parameters that are equally constrained. For example,
  - `"loadings"` For factor loading
  - `"intercepts"` For measurement intercepts
- estimator The method of estimation. `"ML"` is the default, which is a maximum likelihood. `"MLR"` or `"MLM"` is the scaled statistics for nonnormal data adjustment. Ex: `estimator="MLR"`
- `std.lv` If `TRUE`, the fixed-factor method is used for scale identification. If `FALSE`, the manifest variable method is used for scale identification. Ex: `std.lv = FALSE`
- `ordered` The name of categorical variables: Ex: `ordered = c("V1", "V2")`
- `missing` The method to handle missing data. **IMPORTANTLY**, the lavaan default is listwise deletion. The `"fiml"` estimator should be used for full-information maximum likelihood. Ex: `missing = "fiml"`

Multiple Groups

In multiple-group models, researchers may use the concatenate function, `c()`, to specify fixed/free parameters or labels in different groups:

```
> PosModel <- 'Positive =~ V1 + c(eq1, eq1)*V2 + c(eq1, eq1)*V3
               Negative =~ c(1, 1)*V4 + c(0.8, 1.5)*V5 + V6 '
```

If the labels are the same across groups, the parameters are constrained to be equal across groups.

Extract Outputs

The output is saved in a lavaan object. In the previous example, the `Results` is a lavaan object. As shown above, the `summary()` function can be used to summarize the results. Here are the useful options of the `summary()` function:

- `standardized` If `TRUE`, standardized estimates are provided. Ex: `summary(Results, standardized = TRUE)`
- `fit` If `TRUE`, fit indices are provided. Ex: `summary(Results, fit = TRUE)`
- `modindices` If `TRUE`, modification indices are provided. Ex: `summary(Results, modindices = TRUE)`
- `rsquare` If `TRUE`, R-squared are provided. Ex: `summary(Results, fit = TRUE, rsquare = TRUE)`

Other functions that can be used for the lavaan object:

- `inspect` To inspect the lavaan object. Ex: `inspect(Results, "coef")`. This code is used to inspect parameter estimates. Here are other useful parts to be inspected:
  - `"free"` Free parameters
  - `"sampstat"` The statistics of observed variables
  - `"coef"` Parameter estimates
  - `"se"` Standard errors
  - `"fit"` Fit indices
  - `"std"` Standardized coefficients
  - `"r2"` R-squared
  - `"mi"` Modification indices
- `predict` To provide the factor scores. Ex: `predict(Results)`
- `anova` To provide a model comparison statistics between two nested models. Ex: `anova(Results, Results2)`
- `parameterEstimates` Another way to provide the summary of the model. The fraction missing information is provided here. Ex: `parameterEstimates(Results)`

Notable Limitations

Lavaan is still beta software, and does not yet support everything other dedicated software packages for SEM do. This includes support for Bayesian analyses, discrete latent variables, and hierarchical/multilevel data sets.

References

R Development Core Team. (2012). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org

Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software, 48*(2), 1–36. Retrieved from http://www.jstatsoft.org/v48/i02/

User's Guide: http://users.ugent.be/~yrosseel/lavaan/lavaanIntroduction.pdf
Official Reference: http://cran.r-project.org/web/packages/lavaan/lavaan.pdf