SEMinR

Soumya Ray & Nicholas Danks

January 3, 2018

Introduction

SEMinR brings many advancements to creating and estimating structural equation models (SEM) using Partial Least Squares Path Modeling (PLS-PM):

- A *natural* feeling, *domain-specific* language to build and estimate structural equation models in R
- Uses *variance-based PLS estimation* to model both *composite* and *common-factor* constructs
- *High-level functions* to quickly specify interactions and complicated structural models

SEMinR follows the latest best-practices in methodological literature:

- Automatically *adjusts PLS estimates to ensure consistency (PLSc)* wherever common-factors are involved
- Ajusts for known biases in interaction terms in PLS models
- Continuously tested against leading PLSPM software to ensure parity of outcomes: SmartPLS (Ringle et al., 2015) and ADANCO (Henseler and Dijkstra, 2015), as well as other R packages such as semPLS (Monecke and Leisch, 2012) and matrixpls (Rönkkö, 2016)
- *High performance, multi-core* bootstrapping function

Briefly, there are four steps to specifying and estimating a structural equation model using SEMinR:

1. Describe measurement model for each construct and its items:

```r
# Distinguish and mix composite or reflective (common-factor) measurement models
measurements <- constructs(
  composite("Image",       multi_items("IMAG", 1:5), weights = mode_B),
  composite("Expectation", multi_items("CUEX", 1:3), weights = mode_A),
  reflective("Loyalty",    multi_items("CUSL", 1:3))
)
```

2. Specify any interactions between constructs:

```r
# Easily create orthogonalized or scaled interactions between constructs
intxns <- interactions(
  interaction_ortho("Image", "Expectation")
)
```

3. Describe the structural model of causal relationships between constructs (and interactions):

```r
# Quickly create multiple paths "from" and "to" sets of constructs
structure <- relationships(
  paths(from = c("Image", "Expectation", "Image.Expectation"),
        to = "Loyalty")
)
```

4. Put the above elements together to estimate and bootstrap the model:

```
# Dynamically compose SEM models from individual parts
pls_model <- estimate_pls(data = mobi, measurements, intxns, structure)
summary(pls_model)

# Use multi-core parallel processing to speed up bootstraps
boot_estimates <- bootstrap_model(pls_model, nboot = 1000, cores = 2)
summary(boot_estimates)
```

SEMinR seeks to combine ease-of-use, flexible model construction, and high-performance. Below, we will cover the details and options of each of the four parts of model construction and estimation demonstrated above.

Setup

You must install the SEMinR library once on your local machine:

```
install.packages("seminr")
```

And then load it in every session you want to use it:

```
library(seminr)
```

Data

You must load your data into a dataframe from any source you wish (CSV, etc.). Column names must be names of your measurement items.

Important: Avoid dots '.' in your column names (these are reserved for interaction terms).

For demonstration purposes, we will start with a dataset bundled with the seminr package - the `mobi` data frame (also found in the `semPLS` R package). This dataset comes from a measurement instrument for the European Customer Satisfaction Index (ECSI) adapted to the mobile phone market (Tenenhaus et al. 2005).

You can see a description and sample of what is in `mobi`:

```
dim(mobi)
#> [1] 250  24
head(mobi)
#>   CUEX1 CUEX2 CUEX3 CUSA1 CUSA2 CUSA3 CUSCO CUSL1 CUSL2 CUSL3 IMAG1 IMAG2
#> 1     7     7     6     6     4     7     7     6     5     6     7     5
#> 2    10    10     9    10    10     8    10    10     2    10    10     9
#> 3     7     7     7     8     7     7     6     6     2     7     8     7
#> 4     7    10     5    10    10    10     5    10     4    10    10    10
#> 5     8     7    10    10     8     8     5    10     3     8    10    10
#> 6    10     9     7     8     7     7     8    10     3    10     8     9
#>   IMAG3 IMAG4 IMAG5 PERQ1 PERQ2 PERQ3 PERQ4 PERQ5 PERQ6 PERQ7 PERV1 PERV2
#> 1     5     5     4     7     6     4     7     6     5     5     2     3
#> 2    10    10     9    10     9    10    10     9    10    10    10    10
#> 3     6     4     7     7     8     5     7     8     7     7     7     7
#> 4     5     5    10     8    10    10     8     4     5     8     5     5
#> 5     5     8     9    10     9     8    10     9     9     8     6     6
#> 6    10     8     9     9    10     9    10     8     9     9    10    10
```

Measurement model description

SEMinR uses the following functions to describe measurement models:

- `constructs()` gathers all the construct measurement models
- `composite()` or `reflective()` define the measurement mode of individual constructs
- `multi_items()` or `single_item()` define the items of a construct

These functions should be natural to SEM practitioners and encourages them to explicitly specify their core nature of their measurement models: composite or common-factor (See Sarstedt et al., 2016, and Henseler et al., 2013, for clear definitions).

Let's take a closer look at the individual functions.

Create measurement model matrix with `constructs()`

`constructs()` compiles the measurement model source-target matrix from the user specified construct descriptions described in the parameters. You must supply it with any number of individual *composite* or *reflective* constructs:

```
mobi_mm <- constructs(
  composite("Image",         multi_items("IMAG", 1:5), weights = mode_B),
  composite("Expectation",   multi_items("CUEX", 1:3), weights = regression_weights),
  composite("Quality",       multi_items("PERQ", 1:7), weights = mode_A),
  composite("Value",         multi_items("PERV", 1:2), weights = correlation_weights),
  reflective("Satisfaction", multi_items("CUSA", 1:3)),
  reflective("Complaints",   single_item("CUSCO")),
  reflective("Loyalty",      multi_items("CUSL", 1:3))
)
```

We are storing the measurement model in the `mobi_mm` variable for later use.

Note that neither a dataset nor a structural model is specified in the measurement model stage, so we can reuse the measurement model object `mobi_mm` across different datasets and structural models.

Describe individual constructs with `composite()` or `reflective()`

`composite()` or `reflective()` describe a construct and its items.

For example, we can use `composite()` to describe mode A (correlation weights) for the "Expectation" construct with manifest variables CUEX1, CUEX2, and CUEX3:

```
composite("Expectation", multi_items("CUEX", 1:3), weights = mode_A)
# is equivalent to:
composite("Expectation", multi_items("CUEX", 1:3), weights = correlation_weights)
```

We can describe composite "Image" using mode B (regression weights) with manifest variables IMAG1, IMAG2, IMAG3, IMAG4 and IMAG5:

```
composite("Image", multi_items("IMAG", 1:5), weights = mode_B)
# is equivalent to:
composite("Image", multi_items("IMAG", 1:5), weights = regression_weights)
```

Alternatively, we can use `reflective()` to describe the reflective, common-factor measurement of the "Satisfaction" construct with manifest variables CUSA1, CUSA2, and CUSA3:

```
reflective("Satisfaction", multi_items("CUSA", 1:3))
```

Specifying construct measurement items

SEMinR strives to make specification of measurement items shorter and cleaner using `multi_items()` or `single_item()`

- `multi_items()` creates a vector of multiple measurement items with similar names
- `single_item()` describe a single measurement item

We can describe the manifest variables: IMAG1, IMAG2, IMAG3, IMAG4 and IMAG5:

```
multi_items("IMAG", 1:5)
# which is equivalent to the R vector:
c("IMAG1", "IMAG2", "IMAG3", "IMAG4", "IMAG5")
```

`multi_items()` is used in conjunction with `composite()` or `reflective()` to describe a composite and common-factor construct respectively.

We can describe a single manifest variable CUSCO:

```
single_item("CUSCO")
# which is equivalent to the R character string:
"CUSCO"
```

It is important to note that a single-item constructs can be defined as either composite mode A or reflective common-factor, but single-item constructs are essentially composites whose construct scores are determined.

Interaction terms

Creating interaction terms by hand can be a time-consuming and error-prone. SEMinR provides high-level functions for simply creating interactions between constructs.

- `interactions()` gathers all pairs of interacting constructs in the model
- `interaction_ortho()` or `interaction_scaled()` specify individual pairs of interactions

Note that recent studies show PLS models must adjust the standard deviation of the interaction term because: *"In general, the product of two standardized variables does not equal the standardized product of these variables"* (Henseler and Chin 2010). SEMinR automatically adjusts for this providing highly accurate model estimations.

Describe all model interactions with `interactions()`

`interactions()` describes all pairs of interactions in a model.

```
mobi_xm <- interactions(
  interaction_ortho("Image", "Expectation"),
  interaction_ortho("Image", "Value")
)
```

The object held in `mobi_xm` is:

```
mobi_xm
#> function(data, mm, all_intxns=list(...)) {
#>     create_interaction <- function(intxn_function) { intxn_function(data, mm) }
#>     intxns_list <- lapply(all_intxns, create_interaction)
#>     return(intxns_list)
#>   }
#> <environment: 0x10cb36df8>
```

Not that these functions themselves return functions that are not resolved until passed as a parameter to the `estimate_pls()` function for model estimation.

Describe an interaction with `interaction_ortho()` or `interaction_scaled()`

`interaction_ortho()` describes a single interaction composite generated by the orthogonalization method of Henseler and Chin (2010).

`interaction_scaled()` describes a single interaction composite as generated by the scale product-indicator method as described by Henseler and Chin (2010).

For both these methods the standard deviation of the interaction term is adjusted as noted above.

For example, we can describe the following interactions between antecedent and moderator: * "Image" + "Value"

```r
# Orgthogonalized interaction between "Image" x "Expectation"
interaction_ortho("Image", "Expectation")

# Scaled (mean-centered, standardized) interaction between "Image" x "Value"
interaction_scaled("Image", "Value")
```

These functions themselves return functions that are not resolved but passed as parameters to the `interactions()` function discussed above.

Important Note: SEMinR syntax uses a dot "." as a naming convention for the interaction construct. Thus, the "Image" + "Expectation" interaction is called "Image.Expectation" in the structural model below. Please refrain from using a dot "." in the naming of non-interaction constructs.

Structural model description

SEMinR makes for human-readable and explicit structural model specification using these functions:

- `relationships()` gather all the structural relationships between all constructs
- `paths()` specifies relationships between sets of antecedents and outcomes

Create structural model with `relationships()`

`relationships()` compiles the structural model source-target matrix from the user specified structural path descriptions described in the parameters.

For example, we can describe a structural model for the `mobi` data:

```r
mobi_sm <- relationships(
  paths(from = "Image",        to = c("Expectation", "Satisfaction", "Loyalty")),
  paths(from = "Expectation",  to = c("Quality", "Value", "Satisfaction")),
  paths(from = "Quality",      to = c("Value", "Satisfaction")),
  paths(from = "Value",        to = c("Satisfaction")),
  paths(from = "Satisfaction", to = c("Complaints", "Loyalty")),
  paths(from = "Complaints",   to = "Loyalty")
)
```

Note that neither a dataset nor a measurement model is specified in the structural model stage, so we can reuse the structural model object `mobi_sm` across different datasets and measurement models.

Describe structural paths with `paths()`

`paths()` describe single or multiple structural paths between sets of constructs.

For example, we can define paths from a single antecedent construct to a single outcome construct:

```r
# "Image" -> "Expectation"
paths(from = "Image", to = "Expectation")
```

Or paths from a single antecedent to multiple outcomes:

```r
# "Image" -> "Expectation"
# "Image" -> "Satisfaction"
paths(from = "Image", to = c("Expectation", "Satisfaction"))
```

Or paths from multiple antecedents to a single outcome:

```r
# "Image" -> "Satisfaction"
# "Expectation" -> "Satisfaction"
paths(from = c("Image", "Expectation"), to = "Satisfaction")
```

Or paths from multiple antecedents to a common set of outcomes:

```r
# "Expectation" -> "Value"
# "Expectation" -> "Satisfaction"
# "Quality" -> "Value"
# "Quality" -> "Satisfaction"
paths(from = c("Expectation", "Quality"), to = c("Value", "Satisfaction"))
```

Even the most complicated structural models become quick and easy to specify or modify.

PLS SEM Model Estimation

SEMinR can estimate a full SEM model described by the measurement and structural models above:

- `estimate_pls()` estimates the inner and outer parameters of a SEM model using PLSPM algorithms

This function takes the following parameters:

- `data`: the dataset containing the measurement model items specified in `constructs()`
- `measurement_model`: the measurement model described by the `constructs()` function
- `interactions` (optional): the interactions described by the `interactions()` function (default is NULL)
- `structural_model`: the structural model described by the `paths()` function
- `inner_weights`: the weighting scheme for path estimation - either `path_weighting` for path weighting (default) or `path_factorial` for factor weighting (Lohmöller 1989)
- `max_iterations` (optional): the maximum number of iterations (default is 300)
- `stop_criterion` (optional): the number of places of precision to stop at (default is 7)

For example, we can estimate a simple PLS SEM model adapted from the structural and measurement model with interactions described thus far:

```r
# define measurement model
mobi_mm <- constructs(
  composite("Image",        multi_items("IMAG", 1:5)),
  composite("Expectation",  multi_items("CUEX", 1:3)),
  composite("Value",        multi_items("PERV", 1:2)),
  composite("Satisfaction", multi_items("CUSA", 1:3))
)
```

```
# specify interactions among constructs
mobi_xm <- interactions(
  interaction_ortho("Image", "Expectation"),
  interaction_ortho("Image", "Value")
)

# define structural model
# note: interactions cobnstruct should be named by its main constructs joined by a '.'
mobi_sm <- relationships(
  paths(to = "Satisfaction",
        from = c("Image", "Expectation", "Value",
                 "Image.Expectation", "Image.Value"))
)

mobi_pls <- estimate_pls(data = mobi,
                         measurement_model = mobi_mm,
                         interactions = mobi_xm,
                         structural_model = mobi_sm,
                         inner_weights = path_weighting)
#> Generating the seminr model
#> All 250 observations are valid.
```

Consistent PLS (PLSc) for common-factors

Dijkstra and Henseler (2015) offer an adjustment to generate consistent weight and path estimates of common factors estimated using PLSPM. SEMinR automatically adjusts for consistent estimates of coefficients for common-factors defined using `reflective()`.

Note: At this point, SEMinR does not adjust for PLSc on models with interactions involving common-factors. Thus in models including common-factors and interactions, the coefficient of common-factors will be subject to bias.

Bootstrapping the model for significance

SEMinR can conduct high performance bootstrapping.

- `bootstrap_model()` bootstraps a SEMinR model previously estimated using `estimate_pls()`

This function takes the following parameters:

- `seminr_model`: a SEM model provided by `estimate_pls()`
- `nboot`: the number of bootstrap subsamples to generate
- `cores`: If your pc supports multi-core processing, the number of cores to utilize for parallel processing (default is NULL, wherein SEMinR will automatically detect and utilize all available cores)

For example, we can bootstrap the model described above:

```
# use 1000 bootstraps and utilize 2 parallel cores
boot_mobi_pls <- bootstrap_model(seminr_model = mobi_pls,
                                 nboot = 1000,
                                 cores = 2)
#> Bootstrapping model using seminr...
```

Notably, bootstrapping can also be meaningfully applied to models containing interaction terms and readjusts the interaction term (Henseler and Chin 2010) for every sub-sample. This leads to slightly increased processing times, but provides accurate estimations.

Reporting the PLS SEM Model

Reporting the estimated `seminr_model`

There are multiple ways of reporting the estimated model. The `estimate_pls()` function returns an object of class `seminr_model`. This can be passed directly to the base R function `summary()`. This can be used in two primary ways:

1. `summary(seminr_model)` to report R^2, adjusted R^2, path coefficients for the structural model, and the construct reliability metrics rho_C (Dillon and Goldstein 1987), AVE (Fornell and Larcker 1981), and rho_A (Dijkstra and Henseler 2015)

```
summary(mobi_pls)
#>
#>  Total Iterations: 8
#> Path Coefficients:
#>                  Satisfaction
#> R^2                     0.627
#> AdjR^2                  0.619
#> Image                   0.455
#> Expectation             0.165
#> Value                   0.309
#> Image.Expectation      -0.143
#> Image.Value            -0.058
#>
#> Reliability:
#>                    rhoC   AVE  rhoA
#> Image            0.8183 0.478 0.745
#> Expectation      0.7332 0.481 0.462
#> Value            0.9179 0.848 0.858
#> Image.Expectation 0.8006 0.255 0.553
#> Image.Value      0.0439 0.226 0.339
#> Satisfaction     0.8715 0.693 0.783
```

2. `model_summary <- summary(seminr_model)` returns an object of class `summary.seminr_model` which contains the following accessible objects:
   - `model_summary$iterations` reports the number of iterations to converge on a stable model
   - `model_summary$paths` reports the matrix of path coefficients, R^2, and adjusted R^2
   - `model_summary$reliability` reports composite reliability (rho_C), average variance extracted (AVE), and rho_A
   - `model_summary$cross_loadings` reports all possible loadings between contructs and items
   - `model_summary$loadings` reports the estimated loadings of the measurement model
   - `model_summary$weights` reports the estimated weights of the measurement model
   - `model_summary$composite_scores` reports the construct scores of composites

Please note that common-factor scores are indeterminable and therefore construct scores for common factors are not reported (Hair et al.,2011).

Reporting the bootstrapped `boot_seminr_model`

As with the estimated model, there are multiple ways of reporting the bootstrapped model. The `bootstrap_model()` function returns an object of class `boot_seminr_model`. This can be passed directly to the base R function `summary()`. This can be used in two primary ways:

1. `summary(boot_seminr_model)` to report t-values and p-values for the structural paths

```
summary(boot_mobi_pls)
#>
#>  Bootstrapped resamples: 1000
#>
#> Structural Path t-values:
#>                      Satisfaction
#> Image                       8.281
#> Expectation                 2.740
#> Value                       4.363
#> Image.Expectation           1.080
#> Image.Value                 0.033
#>
#> Structural Path p-values:
#>                      Satisfaction
#> Image                       0.000
#> Expectation                 0.006
#> Value                       0.000
#> Image.Expectation           0.280
#> Image.Value                 0.974
```

2. `boot_model_summary <- summary(boot_seminr_model)` returns an object of class `summary.boot_seminr_model` which contains the following accessible objects:
    - `boot_model_summary$nboot` reports the number of bootstraps performed
    - `model_summary$t_values` reports the matrix of t_values for path co-efficients
    - `model_summary$p_values` reports the matrix of p_values for path co-efficients

References

- Dijkstra, T. K., & Henseler, J. (2015). Consistent Partial Least Squares Path Modeling, MIS Quarterly Vol. 39(X).
- Dillon, W. R, and M. Goldstein. 1987. Multivariate Analysis: Methods, and Applications. Biometrical Journal 29 (6): 750–756.
- Fornell, C. and D. F. Larcker (February 1981). Evaluating structural equation models with unobservable variables and measurement error, Journal of Marketing Research, 18, pp. 39-5)
- Hair, J. F., Hult, G. T. M., Ringle, C. M., and Sarstedt, M. (2017). A Primer on Partial Least Squares Structural Equation Modeling (PLS-SEM), 2nd Ed., Sage: Thousand Oaks.
- Hair, J. F., Ringle, C. M., & Sarstedt, M. (2011). PLS-SEM: Indeed a silver bullet. Journal of Marketing theory and Practice, 19(2), 139-152.
- Henseler, J., & Fassot, G. (2006). Testing moderating effects in PLS path models. In: Esposito Vinzi, V., Chin,W.W., Henseler, J., & Wang, H. (Eds.), Handbook PLS and Marketing. Berlin, Heidelberg, New York: Springer.
- Henseler, J., & Chin, W. W. (2010), A comparison of approaches for the analysis of interaction effects between latent variables using partial least squares path modeling. Structural Equation Modeling, 17(1), 82–109. https://doi.org/10.1080/10705510903439003
- Henseler, J., Dijkstra, T. K., Sarstedt, M., Ringle, C. M., Diamantopoulos, A., Straub, D. W., … Calantone, R. J. (2014). Common Beliefs and Reality About PLS. Organizational Research Methods, 17(2), 182–209. https://doi.org/10.1177/1094428114526928
- Henseler, J. and Dijkstra, T.K. (2015), "ADANCO 2.0", Composite Modeling, Kleve, available at: www.compositemodeling.com (accessed December 14, 2015).
- Lohmöller, J.-B. (1989). Latent variables path modeling with partial least squares. Heidelberg, Germany:

Physica- Verlag. Marsh,

- Monecke, A., & Leisch, F. (2012). semPLS: structural equation modeling using partial least squares. Journal of Statistical Software, 48(3), 1–32.
- Ringle, C. M., Wende, S., & Becker, J-M. (2015). SmartPLS 3. Bönningstedt: SmartPLS. Retrieved from http://www.smartpls.com
- Rönkkö, M. (2016). R package matrixpls: Matrix-based partial least squares estimation (version 0.7.0). https://CRAN.R-project.org/package=matrixpls
- Sarstedt, M., Hair, J. F., Ringle, C. M., Thiele, K. O., & Gudergan, S. P. (2016). Estimation issues with PLS and CBSEM: Where the bias lies! Journal of Business Research, 69(10), 3998–4010. https://doi.org/10.1016/j.jbusres.2016.06.007
- Tenenhaus, M., Vinzi, V. E., Chatelin, Y. M., & Lauro, C. (2005). PLS path modeling. Computational Statistics and Data Analysis, 48(1), 159–205. https://doi.org/10.1016/j.csda.2004.03.005