SEMinR: domain-specific language for building, estimating, and visualizing structural equation models in R

Soumya Ray, National Tsing Hua University
Nicholas P. Danks, Trinity College Dublin
André Calero Valdez, RWTH Aachen University

8/06/2021

Summary

SEMinR allows researchers to easily create, estimate, and visualize structural equation models (SEMs) for multiple estimation methods. SEMs are popular modeling techniques in social sciences and the life sciences, and can estimate relationships between concepts that need to be measured by multiple items. SEMinR can estimate SEMs using either covariance-based SEM (CBSEM, such as found in LISREL and Lavaan), or Partial Least Squares Path Modeling (PLS-PM, such as found in SmartPLS, semPLS, plspm, and csem). Moreover, SEMinR implements several advances in SEM methodologies not found elsewhere. And it also allows for visualization of all kinds of SEM models. SEMinR's model description syntax is plain-old-R-functions (PORF!), which allows users to extend and compose syntax in novel ways. Thus, SEMinR is a one-stop-shop for both SEM practitioners seeking to analyze empirical models and SEM methodologists seeking to automate and extend SEM methods. SEMinR is increasingly being used in universities, for both research and teaching needs, and companies world-wide.

Statement of Need

SEMinR seeks to bring the latest state-of-the-art advances in SEM methods to the R ecosystem. This package also seeks to make describing and analyzing SEMs easier for practitioners.

There have been several recent advances in the various branches of SEM that are often not reflected in existing R packages. For example, the PLS-PM approach requires adjustment in how models with interaction terms are estimated. PLS-PM methods have recently incorporated predictive methods such as plsPredict. Meanwhile, CB-SEM approach can avail ten Berge factor-score extraction that obtains construct scores with the same correlation patterns as the latent factors themselves. CB-SEM researchers should also consider VIF scores in their regression models. SEMinR incorporates these and other advancements.

Estimating an SEM using CB-SEM and PLS-PM requires different packages for the two estimation methods, which often requires researchers to wholly redescribe their models in different syntax. SEMinR allows researchers to describe their model once in a common syntax, and estimate the model using different estimation methods. SEMinr includes its own implementation of PLS-PM estimation that is tested against leading commercial applications to ensure comparable results. For CB-SEM estimation, SEMinR delegates the estimation to the popular Lavaan package. Regardless of which estimation method one uses, the results are structured in a similar way for reporting and visualization.

R packages for SEM often use a custom syntax that does not correspond to any programming language; nor does the syntax not reflect the terminology of SEM with which practitioners are familiar. SEMinR offers researchers a domain-specific language for modeling SEMs that uses function names that evoke major SEM

components: constructs, relationships, paths, reflective, composite, etc. As SEMinR's syntax is built using R functions, researchers can inject their own custom functions to extend the behavior of SEMinR.

SEMinR is the first package that allows researchers applying PLS-PM to visualize their graphical models and measurement qualities. Visualization of CB-SEM models is delegated to the semplot package. Moreover, SEMinR allows researchers to visualize models either before or after estimation.

Introduction

SEMinR brings a friendly syntax to creating and estimating structural equation models (SEM). The syntax allows applied practitioners of SEM to use terminology that is very close to their familiar modeling terms (e.g., reflective, composite, interactions) instead of specifying underlying matrices and covariances. SEM models can be estimated either using Partial Least Squares Path Modeling (PLS-PM) as popularized by SmartPLS, or using Covariance Based Structural Equation Modeling (CBSEM) as popularized by LISREL and AMOS. Confirmatory Factor Analysis (CFA) of reflective measurements models is also supported. Both CBSEM and CFA estimation use the Lavaan package.

- A *natural* feeling, *domain-specific* language to build and estimate structural equation models in R
- Can use *variance-based PLS estimation* and *covariance-based SEM estimation* to model *composite* and *common-factor* constructs
- *High-level functions* to quickly specify *interactions*, *higher order* constructs, and *structural paths*

SEMinR uses its own PLS-PM estimation engine and integrates with the Lavaan package for CBSEM/CFA estimation. It also brings a few methodological advancements not found in other packages or software, and encourages best practices wherever possible.

PLS-PM advances and best-practices in SEMinR:

- Implements PLS path modeling algorithm (Wold, 1985)
- Automatically *adjusts PLS estimates to ensure consistency (PLSc)* wherever common-factors are involved (Dijkstra & Henseler, 2015)
- Adjusts for known biases in interaction terms in PLS models (Henseler & Fassot, 2006)
- Continuously tested against leading popular PLS-PM software to ensure parity of outcomes: SmartPLS (Ringle et al., 2015) and ADANCO (Henseler and Dijkstra, 2015), semPLS (Monecke and Leisch, 2012) and matrixpls (Rönkkö, 2016)
- Incorporates *high performance, multi-core* bootstrapping function (Hair et al., 2017)

CBSEM/CFA advances and best-practices in SEMinR:

- Implements covariance-based structural equation modeling (Joreskog, 1973)
- Extracts *ten Berge factor scores* that have the same correlation pattern as the latent constructs (ten Berge et al. 1999; Logan et al. 2019)
- Creates product-indicator *interactions*, or two-stage interactions using ten Berge scores from a CFA (Lodder et al, 2019)
- Defaults to *robust maximum-likelihood* (MLR) estimation to account for potential non-normality (Yuan et al. 2000)

Briefly, there are three steps to specifying and estimating a structural equation model using SEMinR. The following example is generic to either PLS-PM or CBSEM/CFA.

1. Describe measurement model for each construct and its items, specifying interaction terms and other measurement features:

```
# Distinguish and mix composite measurement (used in PLS-PM)
# or reflective (common-factor) measurement (used in CBSEM, CFA, and PLSc)
# - We will first use composites in PLS-PM analysis
# - Later we will convert the omposites into reflectives for CFA/CBSEM (step 3)
```

```
measurements <- 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)),
  interaction_term(iv = "Image", moderator = "Expectation")
)
```

2. Describe the structural model of causal relationships between constructs (and interaction terms):

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

3. Put the above elements together to estimate the model using PLS-PM, CBSEM, or a CFA:

```
# Estimate using PLS-PM from model parts defined earlier
pls_model <- estimate_pls(data = mobi,
                          measurement_model = measurements,
                          structural_model = structure)
summary(pls_model)
# note: PLS requires seperate bootstrapping for PLS path estimates
# SEMinR uses multi-core parallel processing to speed up bootstrapping
boot_estimates <- bootstrap_model(pls_model, nboot = 1000)
summary(boot_estimates)
# Alternatively, we could estimate our model using CBSEM, which uses the Lavaan package
# We often wish to conduct a CFA of our measurement model prior to CBSEM
# note: we must convert composites in our measurement model into reflective constructs for CFA/CBSEM
cfa_model <- estimate_cfa(data = mobi, as.reflective(measurements))
summary(cfa_model)
cbsem_model <- estimate_cbsem(data = mobi, as.reflective(measurements), structure)
summary(cbsem_model)
# note: the Lavaan syntax and Lavaan fitted model can be extracted for your own specific needs
cbsem_model$lavaan_syntax
cbsem_model$lavaan_model
```

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 three 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 using asterixes '*' in your column names (these are reserved for interaction terms).

```
survey_data <- read.csv("mobi_survey_data.csv")
```

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 IMAG3
#> 1     7     7     6     6     4     7     7     6     5     6     7     5     5
#> 2    10    10     9    10    10     8    10    10     2    10    10     9    10
#> 3     7     7     7     8     7     7     6     6     2     7     8     7     6
#> 4     7    10     5    10    10    10     5    10     4    10    10    10     5
#> 5     8     7    10    10     8     8     5    10     3     8    10    10     5
#> 6    10     9     7     8     7     7     8    10     3    10     8     9    10
#>   IMAG4 IMAG5 PERQ1 PERQ2 PERQ3 PERQ4 PERQ5 PERQ6 PERQ7 PERV1 PERV2
#> 1     5     4     7     6     4     7     6     5     5     2     3
#> 2    10     9    10     9    10    10     9    10    10    10    10
#> 3     4     7     7     8     5     7     8     7     7     7     7
#> 4     5    10     8    10    10     8     4     5     8     5     5
#> 5     8     9    10     9     8    10     9     9     8     6     6
#> 6     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
- `interaction_term()` specifies interactions and `higher_composite()` specifies higher order 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.

Specifying measurement models with constructs

`constructs()` compiles the measurement model specification list from the user specified construct descriptions described in the parameters. You must supply it with any number of individual *composite*, *reflective*, *interaction_term*, or *higher_composite* constructs. Note that we currenly only support higher-order constructs for PLS-PM estimation (i.e., composites).

```
measurements <- 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")),
  higher_composite("HOC", c("Value", "Satisfaction"), orthogonal, mode_A),
  interaction_term(iv = "Image", moderator = "Expectation", method =  orthogonal, weights = mode_A),
  reflective("Loyalty",      multi_items("CUSL", 1:3))
)
```

We are storing the measurement model in the `measurements` object 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 `measurements` across different datasets and structural models.

Describe individual constructs as composite or reflective

`composite()` or `reflective()` describe the measurement of a construct by its items.

For example, we can use `composite()` for PLS models 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()` for CBSEM/CFA/PLSc 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))
```

Converting composite models into reflective models

For covariance-based SEM and CFA, you will want constructs to be *reflective* common factors. If you already have composite constructs or measurement models, you may use them for CBSEM/CFA after converting them to reflective versions. The `as.reflective()` function can convert either a single construct or an entire measurement model into reflective forms.

```
# Coerce a composite into reflective form
img_composite <- composite("Image", multi_items("IMAG", 1:5))
img_reflective <- as.reflective(img_composite)
# Coerce all constructs of a measurement model into composite form
mobi_composites <- constructs(
  composite("Image",         multi_items("IMAG", 1:5)),
  composite("Expectation",   multi_items("CUEX", 1:3)),
  reflective("Complaints",   single_item("CUSCO"))
```

```
)
mobi_reflective <- as.reflective(mobi_composites)
```

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")
```

If your constructs are not numbered perfectly sequentially, then you will combine your items using the `c()` function:

```
multi_items("IMAG", c(1, 3:5))
# which is equivalent to the R vector:
c("IMAG1", "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"
```

Note that 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.

Item associations (CBSEM only)

Covariance-based SEM models generally constrain all item errors to be unrelated. However, researchers might sometimes wish to free up covariances between item errors for estimation.

```
# The following specifies that items PERQ1 and PERQ2 covary with each other, both covary with IMAG1
mobi_am <- associations(
  item_errors("PERQ1", "PERQ2"),
  item_errors(c("PERQ1", "PERQ2"), "IMAG1")
)
```

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.

Interaction terms are described in the measurement model function constructs() using the following methods:

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

- `two_stage` describes a single-item interaction composite that uses a product of the IV and moderator construct scores. For PLS-PM, the first stage uses PLS-PM described by Henseler and Chin (2010) whereas for CBSEM, the first stage uses a CFA and extracts ten Berge factor scores.
- `orthogonal` describes a single interaction composite generated by the orthogonalization method of Henseler and Chin (2010). It is more typical to use for composites, to help interpret multicollinearity between product terms

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

For example, we can describe the following interactions between Image and Expectation constructs:

```
# By default, interaction terms are computed using two stage procedures
interaction_term(iv = "Image", moderator = "Expectation")
# You can also explicitly specify how to create the interaction term
interaction_term(iv = "Image", moderator = "Expectation", method =  two_stage)
interaction_term(iv = "Image", moderator = "Expectation", method =  product_indicator)
interaction_term(iv = "Image", moderator = "Expectation", method =  orthogonal)
```

Note that these functions themselves return functions (closures) that are not resolved until processed in the `estimate_pls()` or `estimate_cbsem()` functions for SEM estimation. *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.

Important Note: SEMinR syntax uses an asterisk "*" 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 an asterisk "*" 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

Specify structural model of relationships between constructs

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

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

```
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.

Specify structural paths with

`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:

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

Or paths from a single antecedent to multiple outcomes:

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

Or paths from multiple antecedents to a single outcome:

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

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

```
# "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 and modify.

Model Estimation

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

- `estimate_pls()` estimates the parameters of a PLS-SEM model
- `estimate_cfa()` estimates the parameters of a CFA model using the Lavaan package
- `estimate_cbsem()` estimates the parameters of a CBSEM model using the Lavaan package

The above functions take some combination of the following parameters:

- `data`: the dataset containing the measurement model items specified in `constructs()`
- `measurement_model`: the measurement model described by the `constructs()` function
- `structural_model` (PLS-PM and CBSEM only): the structural model described by the `paths()` function
- `inner_weights` (PLS-PM only): the weighting scheme for path estimation - either `path_weighting` for path weighting (default) or `path_factorial` for factor weighting (Lohmöller 1989).

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

```
# 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)),
  interaction_term(iv = "Image", moderator = "Expectation"),
```

```
  interaction_term(iv = "Image", moderator = "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,
  structural_model = mobi_sm,
  inner_weights = path_weighting
)
#> Generating the seminr model
#> All 250 observations are valid.
mobi_cfa <- estimate_cfa(
  data = mobi,
  measurement_model = as.reflective(mobi_mm)
)
#> Generating the seminr model for CFA
mobi_cbsem <- estimate_cbsem(
  data = mobi,
  measurement_model = as.reflective(mobi_mm),
  structural_model = mobi_sm
)
#> Generating the seminr model for CBSEM
```

Consistent PLS (PLSc) estimation for common factors

Dijkstra and Henseler (2015) offer an adjustment to generate consistent weight and path estimates of common factors estimated using PLS-PM. When estimating PLS-PM models using `estimate_pls()`, SEMinR automatically adjusts to produce consistent estimates of coefficients for common-factors defined using `reflective()`.

Note: SEMinR also uses PLSc on PLS models with interactions involving reflective constructs. PLS models with interactions can be estimated as PLS consistent, but are subject to some bias as per Becker et al. (2018). It is not uncommon for bootstrapping PLSc models to result in errors due the calculation of the adjustment.

Bootstrapping PLS models 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...
#> SEMinR Model successfully bootstrapped
```

1. `bootstrap_model()` returns an object of class `boot_seminr_model` which contains the original model estimation elements as well as the following accessible bootstrap elements:
   - `boot_seminr_model$boot_paths` an array of the `nboot` estimated bootstrap sample path coefficient matrices
   - `boot_seminr_model$boot_loadings` an array of the `nboot` estimated bootstrap sample item loadings matrices
   - `boot_seminr_model$boot_weights` an array of the `nboot` estimated bootstrap sample item weights matrices
   - `boot_seminr_model$boot_HTMT` an array of the `nboot` estimated bootstrap sample model HTMT matrices
   - `boot_seminr_model$boot_total_paths` an array of the `nboot` estimated bootstrap sample model total paths matrices
   - `boot_seminr_model$paths_descriptives` a matrix of the bootstrap path coefficients and standard deviations
   - `boot_seminr_model$loadings_descriptives` a matrix of the bootstrap item loadings and standard deviations
   - `boot_seminr_model$weights_descriptives` a matrix of the bootstrap item weights and standard deviations
   - `boot_seminr_model$HTMT_descriptives` a matrix of the bootstrap model HTMT and standard deviations
   - `boot_seminr_model$total_paths_descriptives` a matrix of the bootstrap model total paths and standard deviations

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 model estimation results

Reporting the estimated 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, also known as composite reliability (Dillon and Goldstein 1987), AVE (Fornell and Larcker 1981), and rho_A (Dijkstra and Henseler 2015).

```
summary(mobi_pls)
#>
#> Results from  package seminr (2.1.0)
#>
#> Path Coefficients:
```

```
#>                    Satisfaction
#> R^2                       0.614
#> AdjR^2                    0.606
#> Image                     0.470
#> Expectation               0.132
#> Value                     0.320
#> Image*Expectation        -0.140
#> Image*Value               0.023
#>
#> Reliability:
#>                    alpha  rhoC   AVE  rhoA
#> Image              0.723 0.818 0.478 0.745
#> Expectation        0.452 0.733 0.481 0.462
#> Value              0.824 0.918 0.848 0.858
#> Image*Expectation 0.802 0.833 0.291 1.000
#> Image*Value        0.810 0.918 0.574 1.000
#> Satisfaction       0.779 0.871 0.693 0.784
#>
#> Alpha, rhoC, and rhoA should exceed 0.7 while AVE should exceed 0.5
```

2. `model_summary <- summary(seminr_model)` returns an object of class `summary.seminr_model` which contains the following accessible objects (might vary depending on CBSEM or PLS model):
   - `meta` reports the metadata about the estimation technique and version
   - `model_summary$iterations` (PLS only) reports the number of iterations to converge on a stable model
   - `model_summary$paths` reports the matrix of path coefficients, R^2, and adjusted R^2
   - `total_effects` reports the total effects of the structural model
   - `total_indirect_effects` reports the total indirect effects of the structural model
   - `model_summary$loadings` reports the estimated loadings of the measurement model
   - `model_summary$weights` reports the estimated weights of the measurement model
   - `model_summary$validity$vif_items` reports the Variance Inflation Factor (VIF) for the measurement model
   - `model_summary$validity$htmt` reports the HTMT for the constructs
   - `model_summary$validity$fl_criteria` reports the fornell larcker criteria for the constructs
   - `model_summary$validity$cross_loadings` (PLS only) reports all possible loadings between contructs and items
   - `model_summary$reliability` reports composite reliability (rho_C), cronbachs alpha, average variance extracted (AVE), and rho_A
   - `model_summary$composite_scores` reports the construct scores of composites
   - `model_summary$vif_antecedents` report the Variance Inflation Factor (VIF) for the structural model
   - `model_summary$fSquare` reports the effect sizes (f^2) for the structural model

   - `model_summary$descriptives` reports the descriptive statistics and correlations for both items and constructs
   - `model_summary$fSquare` reports the effect sizes (f^2) for the structural model
   - `it_criteria` reports the AIC and BIC for the outcome constructs

Please note that common-factor scores are indeterminable and therefore construct scores for reflecive common factors are extracted using a ten Berge procedure.

Reporting results of a bootstrapped PLS

As with the estimated model, there are multiple ways of reporting the bootstrapping of a PLS 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

Get information about bootstrapped PLS models using the `summary()` function on the bootstrapped model object.

```
summary(boot_mobi_pls)
```

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$nboot` reports the number of bootstraps performed
   - `model_summary$bootstrapped_paths` reports a matrix of direct paths and their standard deviation, t_values, and confidence intervals.
   - `model_summary$bootstrapped_weights` reports a matrix of measurement model weights and their standard deviation, t_values, and confidence intervals.
   - `model_summary$bootstrapped_loadings` reports a matrix of measurement model loadings and their standard deviation, t_values, and confidence intervals.
   - `model_summary$bootstrapped_HTMT` reports a matrix of HTMT values and their standard deviation, t_values, and confidence intervals.
   - `model_summary$bootstrapped_total_paths` reports a matrix of total paths and their standard deviation, t_values, and confidence intervals.

Reporting confidence intervals for direct and mediated bootstrapped structural paths

The `summary(boot_seminr_model)` function will return the mean estimate, standard deviation, t_value and confidence intervals for direct structural paths in PLS models. However, the `specific_effect_significance()` function can be used to evaluate the mean estimate, standard deviation, t_value and confidence intervals for specific paths - direct and mediated (Zhao et al., 2010) - in a `boot_seminr_model` object returned by the `bootstrap_model()` function.

This function takes the following parameters:

- `boot_seminr_model`: a bootstrapped SEMinR model returned by `bootstrap_model()`
- `from`: the antecedent construct for the structural path
- `to`: the outcome construct for the structural path
- `through`: the mediator construct, if the path is mediated (default is NULL). This parameter can also take a vector for muliple mediation.
- `alpha` the required level of alpha (default is 0.05)

and returns a specific confidence interval using the percentile method as per Henseler et al. (2014).

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

```
# Creating structural model
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")
)
# Estimating the model
mobi_pls <- estimate_pls(data = mobi,
                         measurement_model = mobi_mm,
                         structural_model = mobi_sm)
#> Generating the seminr model
#> All 250 observations are valid.
# Load data, assemble model, and bootstrap
boot_seminr_model <- bootstrap_model(seminr_model = mobi_pls,
                                     nboot = 50, cores = 2, seed = NULL)
#> Bootstrapping model using seminr...
#> SEMinR Model successfully bootstrapped
# Calculate the 5% confidence interval for mediated path Image -> Expectation -> Satisfaction
specific_effect_significance(boot_seminr_model = boot_seminr_model,
                             from = "Image",
                             through = c("Expectation", "Satisfaction"),
                             to = "Complaints",
                             alpha = 0.05)
#>  Original Est. Bootstrap Mean   Bootstrap SD        T Stat.        2.5% CI
#>    0.016670348    0.015155263    0.015073592    1.105930707   -0.007940149
#>       97.5% CI
#>    0.041018710
# Calculate the 10% confidence interval for direct path Image -> Satisfaction
specific_effect_significance(boot_seminr_model = boot_seminr_model,
                             from = "Image",
                             to = "Satisfaction",
                             alpha = 0.10)
#>  Original Est. Bootstrap Mean   Bootstrap SD        T Stat.          5% CI
#>     0.17873950     0.19166305     0.05674327     3.14996841     0.10683343
#>         95% CI
#>     0.28639050
```

Reporting data descriptive statistics and construct descriptive statistics

The `summary(seminr_model)` function will return four matrices: `model_summary <- summary(seminr_model)` returns an object of class `summary.seminr_model` which contains the following four descriptive statistics matrices:

+ `model_summary$descriptives$statistics$items` reports the descriptive statistics for items
+ `model_summary$descriptives$correlations$items` reports the correlation matrix for items
+ `model_summary$descriptives$statistics$constructs` reports the descriptive statistics for constructs
+ `model_summary$descriptives$correlations$constructs` reports the correlation matrix for constructs

```
model_summary <- summary(mobi_pls)
model_summary$descriptives$statistics$items
```

```
model_summary$descriptives$correlations$items
model_summary$descriptives$statistics$constructs
model_summary$descriptives$correlations$constructs
```

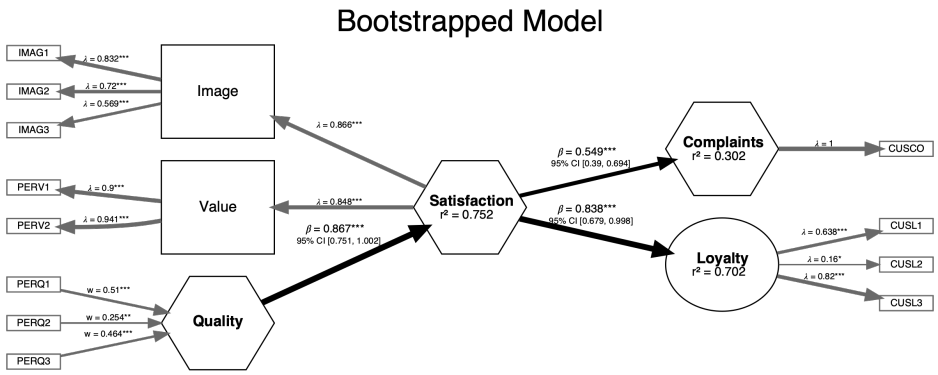
Plotting models

SEMinR can plot all supported models using the dot language and the graphViz.js widget from the `DiagrammeR` package.

```
# generate a small model for creating the plot
mobi_mm <- constructs(
  composite("Image",        multi_items("IMAG", 1:3)),
  composite("Value",        multi_items("PERV", 1:2)),
  higher_composite("Satisfaction", dimensions = c("Image","Value"), method = two_stage),
  composite("Quality",      multi_items("PERQ", 1:3), weights = mode_B),
  composite("Complaints",   single_item("CUSCO")),
  reflective("Loyalty",      multi_items("CUSL", 1:3))
)
mobi_sm <- relationships(
  paths(from = c("Quality"),  to = "Satisfaction"),
  paths(from = "Satisfaction", to = c("Complaints", "Loyalty"))
)
pls_model <- estimate_pls(
  data = mobi,
  measurement_model = mobi_mm,
  structural_model = mobi_sm
)
#> Generating the seminr model
#> Generating the seminr model
#> All 250 observations are valid.
#> All 250 observations are valid.
boot_estimates <- bootstrap_model(pls_model, nboot = 1000)
#> Bootstrapping model using seminr...
#> SEMinR Model successfully bootstrapped
```

When we have a model, we can plot it and save the plot to a file.

```
plot(boot_estimates, title = "Bootstrapped Model")
save_plot("myfigure.png")
```



References

- Becker et al. (2018). Estimating Moderating Effects in PLS-SEM and PLSc-SEM: Interaction Term Generation*Data Treatment
- Cohen, J. (2013). Statistical power analysis for the behavioral sciences. Routledge.
- 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).
- Jöreskog, K. G. (1973). A general method for estimating a linear structural equation system In: Goldberger AS, Duncan OD, editors. Structural Equation Models in the Social Sciences. New York: Seminar Press.
- Lodder, P., Denollet, J., Emons, W. H., Nefs, G., Pouwer, F., Speight, J., & Wicherts, J. M. (2019). Modeling interactions between latent variables in research on Type D personality: A Monte Carlo simulation and clinical study of depression and anxiety. Multivariate behavioral research, 54(5), 637-665.
- Logan, J., Jiang, H., Helsabeck, N., & Yeomans-Maldonado, G. (2019). Should I Allow my Confirmatory Factors to Correlate During Factor Extraction? Implications for the Applied Researcher.
- 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 https://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
- Ten Berge, J. M., Krijnen, W. P., Wansbeek, T., & Shapiro, A. (1999). Some new results on correlation-preserving factor scores prediction methods. Linear Algebra and its Applications, 289(1-3), 311-318.
- 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
- Wold, H. (1985). In S. Kotz & NL Johnson (Eds.), Encyclopedia of statistical sciences. Partial least squares (Vol. 6, pp. 581591).
- Yuan, K. H., & Bentler, P. M. (2000). Three likelihood-based methods for mean and covariance structure analysis with nonnormal missing data. Sociological methodology, 30(1), 165-200.
- Zhao, X., Lynch Jr, J. G., & Chen, Q. (2010). Reconsidering Baron and Kenny: Myths and truths

about mediation analysis. Journal of consumer research, 37(2), 197-206.