Package: OptimalFactor (via r-universe)

October 25, 2026

Type Package

Title Item Selection in Factor Analysis by Boosting and Specification Search

Version 1.5.0

Date 2026-10-05

Description Provides an iterative optimization algorithm for exploratory and confirmatory factor analysis that follows a boosting approach inspired by machine learning. It uses a composite fit index whose weights adapt to the degrees of freedom and the sample size, following Kenny and McCoach (2003) <doi:10.1207/S15328007SEM1003_1> and Shi, Lee and Maydeu-Olivares (2019) <doi:10.1177/0013164418783530>. It detects and removes problematic items (Heywood cases, cross-loadings, low loadings) with greedy or global search strategies, checks interfactor correlations, and supports ordinal data with WLSMV estimation and several rotation methods. A heuristic specification search for confirmatory models (MacCallum, 1986 <doi:10.1037/0033-2909.100.1.107>) explores move, drop and covariance operations across seed configurations. It also evaluates the resampling stability of the item selection, the recovery of a known or theoretical structure, and offers an optional conceptual analysis of removed items through a large language model service.

License GPL (>= 3)

URL https://github.com/jventural/OptimalFactor, https://jventural.github.io/OptimalFactor/

BugReports https://github.com/jventural/OptimalFactor/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.0.0)

Imports lavaan, psych, dplyr, tidyr, ggplot2, stringr, parallel, stats, utils

Suggests testthat (>= 3.0.0), httr, jsonlite, R.utils, knitr, rmarkdown, semTools

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev

Repository https://jventural.r-universe.dev

Date/Publication 2026-10-05 13:08:24 UTC

RemoteUrl https://github.com/jventural/optimalfactor

RemoteRef HEAD

RemoteSha cc69b9449dfc4e8fdc57683ff91c9484e98fcaea

Contents

algorithm_stability . . . 3
bifactor_indices . . . 5
cfa_boosting . . . 7
cross_validate_cfa . . . 9
Data_Expectativas . . . 11
Data_Personality . . . 11
discriminant_boosting . . . 12
efa_boosting . . . 14
efa_optimizer . . . 20
export_cfa_boosting . . . 24
export_conceptual_analysis . . . 25
imprimir_items_eliminados . . . 27
item_stability . . . 28
local_fit_search . . . 30
optimal_cfa_with_ai . . . 32
optimal_efa_with_ai . . . 35
plot.simulate_cfa_recovery . . . 39
plot.simulate_recovery . . . 40
print.discriminant_boosting . . . 41
print.local_fit_search . . . 41
print.specification_search . . . 42
print_cfa_boosting . . . 42
print_cfa_results . . . 44
print_conceptual_analysis . . . 45
print_efa_results . . . 47
redundancy_short_form . . . 48
report_cfa_results . . . 50
report_efa_results . . . 51
simulate_cfa_recovery . . . 53
simulate_recovery . . . 55
specification_search_theory . . . 58

theory_recovery 61

Index **63**

`algorithm_stability` *Split-Half Stability of Any Item-Selection Algorithm*

Description

Answers the question that decides between purification algorithms: does the algorithm make the same decision on a different sample, and does that decision hold on data it has never seen? The data are split at random into two halves many times. On each split the algorithm runs on the derivation half, and the structure it returns is fitted as a CFA on the validation half. Unlike `item_stability` (tied to `efa_boosting()`) and `cross_validate_cfa` (one factor only), the algorithm here is any function, so every routine in the package can be compared on the same footing.

Usage

```
algorithm_stability(
  data,
  algorithm,
  n_splits = 30,
  theory = NULL,
  reference = NULL,
  estimator = "WLSMV",
  ordered = TRUE,
  targets = c(cfi = 0.95, rmsea = 0.08, srmr = 0.08),
  phi_max = 0.9,
  seed = NULL,
  n_cores = 1,
  timeout = 300,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with the item responses.
`algorithm`	A function of one argument, the data of the derivation half, returning the structure it selects: a named list factor -> items, a loading matrix, or the object returned by `efa_boosting()`, `cfa_boosting()`, `discriminant_boosting()`, `specification_search_theory()` or `local_fit_search()`. The residual covariances these routines free are kept in the validation CFA. Functions from other packages must be called with `pkg::fun` inside it when `n_cores > 1`.
`n_splits`	Number of random splits. Default 30.
`theory`	Optional named list with the theoretical key, as in `theory_recovery`.
`reference`	Optional structure used as reference for `jaccard`. Default `NULL` runs `algorithm` once on the complete sample.

`estimator`, `ordered`
: Estimation of the validation CFA. Defaults `"WLSMV"` and `TRUE` (items treated as ordinal).

`targets`
: Named vector with the fit targets used for `meets`. Default `c(cfi = .95, rmsea = .08, srmr = .08)`.

`phi_max`
: Interfactor correlation at or above which two factors are considered indistinguishable; part of `meets`. Default .90.

`seed`
: Optional integer seed for reproducibility; NULL (default) leaves the RNG untouched.

`n_cores`
: Number of cores. Values above 1 run the splits on a PSOCK cluster (works on Windows). Default 1.

`timeout`
: Seconds allowed for the algorithm on one split; a split that exceeds it is counted as failed instead of stalling the whole run. Small derivation halves can send an iterative search into a loop that never ends. Requires the R.utils package; `NULL` or `Inf` disables it. Default 300.

`verbose`
: Print progress. Default `TRUE`.

Details

An algorithm that purifies items on the same data used to judge them capitalises on chance (MacCallum, Roznowski & Necowitz, 1992): the fit it reports is optimistic, and a different sample may have led it elsewhere. Two kinds of evidence are therefore recorded on every split:

Replication of the decision `jaccard`, the overlap between the items retained on the derivation half and those retained by the reference solution (the algorithm run once on the complete sample), and `ari_reference`, the agreement of the two partitions on their common items. `item_retention` gives, for each item, the proportion of splits in which it survived: an item dropped in 29 of 30 splits is a defensible removal, one dropped in 16 is a coin flip.

Out-of-sample adequacy CFI, TLI, RMSEA, SRMR and the largest interfactor correlation of the derived structure fitted on the validation half, and `meets`, whether all the targets are met there.

When `theory` is given, `theory_recovery` is applied to every derived structure, so the stability of the recovery of the theory is reported as well.

The halves are drawn on the master, so results are reproducible with `seed` and identical for any `n_cores`.

Value

An object of class `algorithm_stability`: a list with `summary` (means over successful splits, plus `meets_rate` and `success_rate`), `splits` (one row per split), `item_retention`, `reference` (the reference partition) and `call`.

References

MacCallum, R. C., Roznowski, M., & Necowitz, L. B. (1992). Model modifications in covariance structure analysis: The problem of capitalization on chance. *Psychological Bulletin, 111*(3), 490-504. doi:10.1037/00332909.111.3.490

See Also

theory_recovery, item_stability.

Examples

```
data(Data_Expectativas)
items  <- paste0("EAF", 1:10)
theory <- list(F1 = paste0("EAF", 1:5), F2 = paste0("EAF", 6:10))
model  <- paste(sapply(names(theory), function(f)
            paste(f, "=~", paste(theory[[f]], collapse = " + "))), collapse = "\n")

st <- algorithm_stability(
  Data_Expectativas[, items],
  algorithm = function(d) cfa_boosting(d, model = model, verbose = FALSE,
                model_config = list(estimator = "MLR", ordered = FALSE)),
  n_splits = 3, theory = theory, estimator = "MLR", ordered = FALSE,
  seed = 1)
st
```

---

bifactor_indices *Bifactor Statistical Indices*

Description

Computes the auxiliary bifactor statistical indices from a fitted bifactor `lavaan` model: ECV (explained common variance), PUC (percent of uncontaminated correlations), omega, omega hierarchical (ω_H), omega hierarchical subscale (ω_{HS}), the H construct-replicability index, and the item-level ECV (I-ECV). These indices help decide whether a multidimensional scale can be treated as essentially unidimensional and scored with a single total score (Rodriguez, Reise & Haviland, 2016).

Usage

```
bifactor_indices(fit, general = NULL, verbose = TRUE)
```

Arguments

`fit`	A fitted bifactor `lavaan` object (general factor + orthogonal specific factors).
`general`	Name of the general factor. If `NULL` (default) it is auto-detected as the latent variable that loads on all items.
`verbose`	Logical. If `TRUE` (default), prints the rounded indices to the console.

Details

The model must be a bifactor structure with one general factor loading on all items plus orthogonal specific factors. Standardized loadings are used. With general loadings g_i, specific loadings s_i and errors $e_i = 1 - g_i^2 - s_i^2$:

- `ECV` = $\sum g_i^2 / \sum(g_i^2 + s_i^2)$.
- `omega_H` = $(\sum g_i)^2 / Var_{total}$, and `omega` adds the grouped specific variance to the numerator.
- `omega_HS` (per specific factor) = $(\sum s_i)^2 / Var_{sub}$.
- `H` = $1/(1 + 1/\sum(\lambda_i^2/(1-\lambda_i^2)))$.
- `PUC` = proportion of item correlations that are between items of different specific factors.
- `IECV_i` = $g_i^2/(g_i^2 + s_i^2)$ (item purity toward G).

Value

A list with: `overall` (a one-row data frame with ECV, PUC, omega, omega_H and H_general), `by_factor` (per specific factor: ECV, omega_S, omega_HS, H) and `by_item` (general/specific loadings and I-ECV). Printed rounded to three decimals.

References

Rodriguez, A., Reise, S. P., & Haviland, M. G. (2016). Evaluating bifactor models: Calculating and interpreting statistical indices. *Psychological Methods, 21*(2), 137–150.

Examples

```
# Simulate data from a bifactor population (G + three specific factors)
pop <- '
G  =~ 0.6*x1 + 0.6*x2 + 0.6*x3 + 0.6*x4 + 0.6*x5 + 0.6*x6 +
      0.6*x7 + 0.6*x8 + 0.6*x9
S1 =~ 0.5*x1 + 0.5*x2 + 0.5*x3
S2 =~ 0.5*x4 + 0.5*x5 + 0.5*x6
S3 =~ 0.5*x7 + 0.5*x8 + 0.5*x9
'
set.seed(1)
dat <- lavaan::simulateData(pop, sample.nobs = 500, orthogonal = TRUE)

mod <- '
G  =~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
S1 =~ x1 + x2 + x3
S2 =~ x4 + x5 + x6
S3 =~ x7 + x8 + x9
'
fit <- lavaan::cfa(mod, data = dat, orthogonal = TRUE, std.lv = TRUE)
bi <- bifactor_indices(fit)
bi$by_item
```

cfa_boosting *CFA-Boosting Optimization*

Description

Performs iterative optimization of Confirmatory Factor Analysis (CFA) models using modification indices to improve model fit. The algorithm automatically adds error covariances based on modification indices while respecting configurable constraints.

Usage

```
cfa_boosting(
  data,
  model,
  n_sample = NULL,
  thresholds = list(loading = 0.3, min_items_per_factor = 3,
    rmsea_target = 0.08, cfi_target = 0.95, srmr_target = 0.08,
    enforce_loading = TRUE, cross_loading = 0.3,
    enforce_simple_structure = TRUE),
  model_config = list(estimator = "WLSMV", ordered = TRUE),
  mod_indices_config = list(max_covs_to_add = 10, only_within_factor = TRUE,
    delta = 0.1, power_threshold = 0.75, alpha = 0.05),
  performance = list(max_iterations = 30, timeout_cfa = 60),
  verbose = TRUE
)
```

Arguments

`data` Data frame containing the observed variables.

`model` Character string specifying the CFA model in lavaan syntax.

`n_sample` Sample size. If NULL, auto-detected from data.

`thresholds` List of fit thresholds:

- `loading`: Minimum acceptable loading (default 0.30)
- `min_items_per_factor`: Minimum items per factor (default 3)
- `rmsea_target`: Target RMSEA value (default 0.08)
- `cfi_target`: Target CFI value (default 0.95)
- `srmr_target`: Target SRMR value (default 0.08)
- `enforce_loading`: Treat `loading` as an admissibility floor rather than a hint (default `TRUE`). An item whose standardized loading falls below it is removed even when the fit targets are already met, unless removing it would breach `min_items_per_factor`. `FALSE` restores the behaviour of versions up to 1.3.0, where the loop stopped as soon as the fit targets were satisfied and the loadings were never inspected. Global fit does not reveal a weak item: in simulation an item loading .20 in the population sat comfortably inside a model with RMSEA = .03. Enforcing the floor can leave global fit

slightly worse while making the retained set defensible, which is the trade this option makes explicit.

- `cross_loading`: Standardized magnitude above which an item is declared to load on a foreign factor (default 0.30).
- `enforce_simple_structure`: Remove items that load on a factor other than their own (default `TRUE`). The loading floor catches the item that measures nothing; this catches the one that measures two things, which the floor cannot see because a cross-loading item still loads acceptably on its own factor, and which global fit does not reveal either: an omitted cross-loading is absorbed by the interfactor correlation, so in simulation six items cross-loading at .60 left RMSEA at .053 and CFI at .997. Detection uses the modification index of the absent loading, but never significance alone: the standardized expected parameter change must also reach `cross_loading`, since a significant index with a trivial EPC is precisely the capitalization on chance this is meant to avoid.

`model_config` Model configuration:

- `estimator`: Estimation method (default "WLSMV")
- `ordered`: Whether variables are ordered (default TRUE)

`mod_indices_config`
Modification indices configuration using the Saris, Satorra & van der Veld (2009) framework (MI + EPC + Power):

- `max_covs_to_add`: Maximum covariances to evaluate per iteration (default 10)
- `only_within_factor`: Only consider within-factor covariances (default TRUE)
- `delta`: Minimum misspecification size to detect (default 0.10)
- `power_threshold`: Threshold for high/low power classification (default 0.75)
- `alpha`: Significance level for the MI test (default 0.05)

`performance` Performance settings:

- `max_iterations`: Maximum optimization iterations (default 30)
- `timeout_cfa`: Timeout per CFA run in seconds (default 60)

`verbose` Logical. Print progress information.

Value

A list containing:

- `final_model`: The optimized lavaan CFA model object
- `fit_indices`: Final fit indices
- `added_covariances`: List of error covariances added
- `iterations`: Number of iterations performed
- `history`: History of fit indices across iterations

See Also

efa_boosting, print_cfa_boosting

Examples

```
data(Data_Personality)
model <- '
F1 =~ PPTQ1 + PPTQ2 + PPTQ3 + PPTQ4 + PPTQ5
F2 =~ PPTQ6 + PPTQ7 + PPTQ8 + PPTQ9 + PPTQ10
F3 =~ PPTQ11 + PPTQ12 + PPTQ13 + PPTQ14 + PPTQ15
'
result <- cfa_boosting(
  data    = Data_Personality,
  model   = model,
  verbose = TRUE
)
result$removed_items
result$added_covariances
```

---

cross_validate_cfa *Split-Half Cross-Validation of a Factor Model*

Description

Assesses the out-of-sample stability of a one-factor scale by repeated split-half cross-validation. The sample is randomly split many times; on each split the one-factor model is fitted and its fit and reliability are recorded on independent subsamples. Two modes are available: (a) confirm a **fixed** item set (default), and (b) test the **procedure** by re-deriving a short form with redundancy_short_form in the calibration half and confirming it in the validation half (set derive_k). Mode (b) also returns how often each item is selected, exposing whether a data-driven short form capitalizes on chance.

Usage

```
cross_validate_cfa(
  data,
  items,
  n_splits = 200,
  derive_k = NULL,
  groups = NULL,
  min_per_group = 3,
  estimator = "WLSMV",
  targets = c(cfi = 0.95, rmsea = 0.08),
  seed = NULL,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with the item responses.
`items`	Character vector with the item names of the scale to validate (fixed mode) or the candidate pool (derivation mode).
`n_splits`	Number of random splits. Default 200.
`derive_k`	If `NULL` (default), the fixed `items` set is confirmed across holdout subsamples. If an integer, a `derive_k`-item short form is re-derived in each calibration half and confirmed in the validation half.
`groups, min_per_group`	Passed to `redundancy_short_form` when `derive_k` is set (to preserve content coverage). Default `NULL`, 3.
`estimator`	Estimator passed to `lavaan::cfa`. Default `"WLSMV"`.
`targets`	Named numeric vector of pass thresholds for the fraction of subsamples meeting fit. Default `c(cfi = 0.95, rmsea = 0.08)`.
`seed`	Optional integer seed for reproducibility; NULL (default) leaves the RNG untouched.
`verbose`	Logical. If `TRUE` (default), prints the summary tables to the console.

Value

A list. In fixed mode: `summary` (P10/P50/P90 of cfi, rmsea, srmr, omega across holdout subsamples) and `pct_meeting` (fraction meeting all `targets`). In derivation mode it adds `selection_freq` (per-item selection frequency) and the confirmation summary of the derived forms.

References

MacCallum, R. C., Roznowski, M., & Necowitz, L. B. (1992). Model modifications in covariance structure analysis: The problem of capitalization on chance. *Psychological Bulletin, 111*(3), 490–504.

See Also

`redundancy_short_form`

Examples

```
data(Data_Expectativas)
items <- paste0("EAF", 1:10)
# confirm a fixed item set out of sample
cross_validate_cfa(Data_Expectativas, items, n_splits = 3, seed = 1)
# test the derivation procedure itself
cross_validate_cfa(Data_Expectativas, items, derive_k = 6, n_splits = 3,
                   groups = list(A = paste0("EAF", 1:5),
                                 B = paste0("EAF", 6:10)),
                   seed = 1)
```

Data_Expectativas *Expectativas Dataset*

Description

A dataset containing responses to an expectations scale. This dataset is provided as an example for demonstrating the EFA optimization functions in the OptimalFactor package.

Usage

```
data(Data_Expectativas)
```

Format

A data frame with observations and item responses for the expectations scale.

Source

Simulated data for demonstration purposes.

Examples

```
data(Data_Expectativas)
head(Data_Expectativas)
```

Data_Personality *Personality Dataset*

Description

A dataset containing responses to a personality assessment scale. This dataset is provided as an example for demonstrating the EFA optimization functions in the OptimalFactor package.

Usage

```
data(Data_Personality)
```

Format

A data frame with observations and item responses for personality traits.

Source

Simulated data for demonstration purposes.

Examples

```
data(Data_Personality)
head(Data_Personality)
```

discriminant_boosting *Discriminant-Boosting: Rescue the Discriminant Validity of a Scale*

Description

Reespecifies a multidimensional scale whose interfactor correlations are so high that the factors cannot be told apart — or so high that the solution is inadmissible, with a correlation above 1 — using the smallest possible departure from the theoretical structure.

Usage

```
discriminant_boosting(
  data,
  theory,
  reverse_items = NULL,
  phi_max = 0.9,
  cfi_target = 0.95,
  rmsea_target = 0.08,
  omega_min = 0.7,
  min_loading = 0.35,
  min_items_per_factor = 4,
  exclude_reverse = c("auto", "always", "never"),
  try_method_factor = TRUE,
  try_bifactor = TRUE,
  estimator = "WLSMV",
  ordered = TRUE,
  n_cores = 1,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with the item responses.
`theory`	Named list encoding the theoretical structure, of the form `list(FactorA = c("it1","it2",...), FactorB = c(...))`.
`reverse_items`	Character vector of reverse-worded items, used to test whether they form a method factor. Default `NULL`.
`phi_max`	Maximum acceptable interfactor correlation. Default `0.90`.
`cfi_target, rmsea_target`	Fit targets. Defaults `0.95` and `0.08`.
`omega_min`	Reliability floor per factor. Default `0.70`.
`min_loading`	Loading floor. Default `0.35`.
`min_items_per_factor`	Structural protection rule. Default `4`.

`exclude_reverse`
: One of `"auto"`, `"always"` or `"never"`. With `"auto"` (default) a method factor over `reverse_items` is fitted and, if it raises CFI by more than .02, those items are dropped before the search. Reverse wording introduces shared variance that does not belong to the construct (Marsh, 1996; Podsakoff et al., 2003).

`try_method_factor`, `try_bifactor`
: Whether to include the method-factor and the bifactor specifications in the ladder. Default `TRUE`.

`estimator`, `ordered`
: Passed to `cfa`. Defaults `"WLSMV"` and `TRUE`.

`n_cores`
: Cores for the candidate evaluations within each greedy iteration. Default `1`. Values above 1 use a PSOCK cluster.

`verbose`
: Logical; print progress. Default `TRUE`.

Details

A model can fit well and still be uninterpretable. A four-factor solution whose factors correlate .97 reproduces the covariance matrix as well as a one-factor solution does, so CFI and RMSEA say nothing about whether the factors exist. This routine targets the correlation itself.

Three findings shape the algorithm:

- **Pruning by fit does not lower the interfactor correlation.** Removing the item that most improves CFI can leave ϕ untouched or even raise it. The greedy search must be guided by ϕ.
- **Letting the data assign items surfaces method variance.** When item-to-factor assignment is taken from an exploratory solution, what separates is often the wording polarity of the items, not their content, and the algorithm reports a method factor as if it were a dimension. The assignment therefore stays *theoretical*; only the grouping of whole factors and the set of retained items are searched.
- **Optimising fit and ϕ at once lets fit dominate.** They are optimised in sequence: first ϕ, then CFI.

The search proceeds in four phases.

Phase 0 — diagnosis. The theoretical model is fitted and its interfactor correlations inspected. A correlation above 1 is reported as an inadmissible solution, not as a large correlation: the latent covariance matrix is no longer positive definite and the model describes something that cannot exist.

Phase 1 — ladder of structures. Candidates are generated ordered by distance from the original: the theoretical model, the model plus a method factor, the bifactor model, and every partition of the factors into $k-1, k-2, \ldots, 2$ blocks. Item-to-factor assignment never changes; only whole factors are merged.

Phases 2–4 — pruning. For each level of the ladder the best candidate is pruned by ϕ, then by CFI while holding ϕ below the threshold, and finally cleaned of items below the loading floor.

The winner is the model that meets every criterion *with the largest number of factors*, not the one with the best fit: the aim is to depart as little as possible from the intended structure. If none qualifies, the admissible model with the lowest ϕ is returned and `reason` says so.

Reliability floors are *adaptive*: a model already below the floor is not frozen, its effective bar becomes its current value, so removals that do not worsen reliability remain available. A floor set above the starting value would reject every candidate and stall the search on the first iteration.

Warning. Like any specification search this is an EXPLORATORY device that capitalises on chance (MacCallum, Roznowski & Necowitz, 1992). Cross-validate the chosen model on an independent sample; see `cross_validate_cfa`.

Value

An object of class `discriminant_boosting`, a list with `best` (the retained model and its indices), `best_model`, `reason`, `baseline`, `ladder` (one row per candidate), `refined`, `logs`, `all`, `reverse_excluded` and `config`.

References

Marsh, H. W. (1996). Positive and negative global self-esteem: A substantively meaningful distinction or artifactors? *Journal of Personality and Social Psychology, 70*(4), 810-819.

MacCallum, R. C., Roznowski, M., & Necowitz, L. B. (1992). Model modifications in covariance structure analysis: The problem of capitalization on chance. *Psychological Bulletin, 111*(3), 490-504.

Podsakoff, P. M., MacKenzie, S. B., Lee, J.-Y., & Podsakoff, N. P. (2003). Common method biases in behavioral research. *Journal of Applied Psychology, 88*(5), 879-903.

See Also

`local_fit_search`, `specification_search_theory`, `cross_validate_cfa`

Examples

```
if (requireNamespace("semTools", quietly = TRUE)) {
  data(Data_Personality)
  theory <- list(F1 = paste0("PPTQ", 1:5),
                 F2 = paste0("PPTQ", 6:10),
                 F3 = paste0("PPTQ", 11:15))
  res <- discriminant_boosting(Data_Personality, theory,
           estimator = "MLR", ordered = FALSE, n_cores = 1)
  res
  res$ladder
}
```

`efa_boosting` *EFA-Boosting Optimization*

Description

Performs an iterative, machine-learning-inspired optimization of Exploratory Factor Analysis (EFA).

The algorithm integrates: (1) greedy item-by-item elimination, (2) optional global subset search (multi-item removal), (3) structural rule enforcement (loadings, Heywood / near-Heywood, cross-loadings, minimum items per factor), (4) adaptive minimum interfactor correlation checks, and (5) a composite fit index whose weights adapt dynamically to degrees of freedom *and* sample size (df x N), following Kenny and McCoach (2003) and Shi, Lee and Maydeu-Olivares (2019).

The function handles ordinal data (WLSMV) and includes robust corrections, adaptive weighting rules, and an optional GPT-based conceptual analysis of removed (and optionally retained) items.

When `verbose = TRUE`, the function prints a complete *EFA-Boosting Optimization Summary Report* including:

(a) iteration diagnostics (RMSEA, SRMR, CFI, df, composite loss), (b) identification and elimination of structural problems, (c) global-search progress bars and decisions, (d) thresholded final structure, (e) interfactor-correlation compliance check, and (f) AI-based conceptual interpretations.

Returns a complete list of results invisibly.

Usage

```
efa_boosting(
  data,
  name_items,
  item_range = NULL,
  n_factors = 3,
  n_sample = NULL,
  exclude_items = NULL,
  thresholds = list(loading = 0.3, min_items_per_factor = 3,
    heywood_tol = 1e-06, near_heywood = 0.015,
    min_interfactor_correlation = 0.32, min_omega = NULL),
  model_config = list(estimator = "WLSMV", rotation = "oblimin"),
  performance = list(max_candidates_eval = 12, smart_pruning = TRUE,
    timeout_efa = 30, timeout_optimization = 120, use_timeouts = FALSE,
    emit_progress = TRUE, fit_target_only = TRUE),
  use_global = FALSE,
  global_opt = list(max_drop = 2, max_global_combinations = 5000,
    verbose = TRUE, progress_bar = TRUE),
  fit_config = list(targets = list(rmsea = 0.08, srmr = 0.06, cfi = 0.95),
    margins = list(rmsea = 0.03, srmr = 0.03, cfi = 0.03),
    base_weights = list(rmsea = 0.5, srmr = 0.25, cfi = 0.25),
    small_df_cut = 5,
    small_df_weights = list(rmsea = 0.15, srmr = 0.45, cfi = 0.4),
    wlsmv_boost = list(rmsea = 0.8, srmr = 1.2, cfi = 1.1),
    use_pclose_if_available = TRUE, pclose_bonus = 0.1),
  use_ai_analysis = FALSE,
  ai_config = list(api_key = NULL, generate_names = FALSE,
    only_removed = TRUE, gpt_model = "gpt-3.5-turbo", language = "english",
    analysis_detail = "detailed", domain_name = "Default Domain",
    scale_title = "Default Scale Title", construct_definition = "",
```

```
    model_name = "EFA Model", item_definitions = NULL),
  verbose = TRUE,
  ...
)
```

Arguments

`data` Data frame of item responses.

`name_items` Item prefix (e.g., `"IT"` produces IT1, IT2, ...). Used for auto-detection.

`item_range` Integer vector `c(start, end)`. If `NULL`, items are auto-detected.

`n_factors` Number of factors.

`n_sample` Sample size. If `NULL`, auto-detected as `nrow(data)`. Used for df x N adaptive weighting of the composite fit index.

`exclude_items` Items excluded before starting optimization.

`thresholds` Rules governing structural decisions:

- `loading`: Minimum acceptable loading.
- `min_items_per_factor`: Structural protection rule.
- `heywood_tol`, `near_heywood`: Detection thresholds.
- `min_interfactor_correlation`: Minimum acceptable factor correlation.
- `min_omega`: Reliability floor. When set (e.g. `0.80`), an item is removed only if every factor keeps McDonald's omega at or above this value, so the algorithm cannot buy fit at the cost of internal consistency. Omega is a guard rail here, never a term in the loss: reliability is only interpretable once the model fits, so it vetoes individual removals instead of competing with the fit indices. A factor that already sits below the floor is not frozen - its effective bar becomes its current omega, so removals that do not reduce reliability are still allowed. Heywood cases are exempt, since an inadmissible solution must be fixed regardless. `NULL` (default) disables the check and reproduces the behaviour of versions prior to 1.3.0 exactly.

`model_config` EFA estimation configuration:

- `estimator = "WLSMV"`
- `rotation = "oblimin"`

`performance` Performance and timeout settings:

- `max_candidates_eval`: If set, only this many candidates are evaluated in greedy mode.
- `timeout_efa`: Timeout per EFA run (requires `R.utils`).
- `timeout_optimization`: Global timeout.
- `use_timeouts`: Enable/disable timeout protection.
- `smart_pruning`: Rank candidates by smallest maximum loading before applying `max_candidates_eval`, so the items most likely to be removed are the ones evaluated.
- `emit_progress`: Emit a `message()` per iteration and per candidate, which is what lets a Shiny app show live progress.

- `fit_target_only`: Fit only the `n_factors` solution while evaluating a candidate (default `TRUE`). The engine used to fit the 1..k-1 solutions as well and discard them, since candidate evaluation reads only the target one. Results are identical, verified on `Data_Personality` and `Data_Expectativas` across structure, removed items, stop reason, RMSEA, omega and the fit table, with 38 percent fewer lavaan fits where the greedy loop evaluates candidates. `FALSE` restores the previous behaviour.

`use_global` Enable global subset search (multi-item removal).

`global_opt` Configuration for global search:

- `max_drop`: Maximum subset size *k*.
- `max_global_combinations`: Hard cap to avoid explosion.
- `verbose`: Print global-search diagnostics.
- `progress_bar`: Show a visual progress bar.

`fit_config` Configuration of the adaptive composite fit index:

- `targets`: Target values for RMSEA, SRMR, CFI.
- `margins`: Tolerances for each index.
- `base_weights`: Default RMSEA/SRMR/CFI weights.
- `critical_weights`: df < 5 and N < 200 (RMSEA nearly useless).
- `df_low_n_high_weights`: df < 5 and N >= 200.
- `df_mid_n_low_weights`: df 5-19 and N < 200.
- `df_mid_n_high_weights`: df 5-19 and N >= 200.
- `critical_df_cut`, `moderate_df_cut`: df boundaries.
- `small_n_cut`: N-boundary for weighting.
- `wlsmv_boost`: Multiplicative corrections for WLSMV.
- `use_pclose_if_available`: Enables p-close bonus.
- `pclose_bonus`: Amount subtracted from composite loss.

`use_ai_analysis` Enable GPT-based conceptual analysis of removed (and optionally retained) items.

`ai_config` AI analysis configuration:

- `api_key`: OpenAI API key.
- `gpt_model`: Model (e.g., `"gpt-4"`, `"gpt-3.5-turbo"`).
- `language`: "spanish" or "english".
- `analysis_detail`: "brief", "standard", "detailed".
- `domain_name`, `scale_title`, `model_name`: Metadata included in prompts.
- `construct_definition`: Theoretical definition of the latent construct.
- `item_definitions`: Named list with item wording.
- `only_removed`: If FALSE, retained items are also analyzed.

`verbose` If TRUE, prints the full diagnostic report, global-search bars, item maps, interfactor-correlation warnings, and AI progress bars.

`...` Additional arguments passed to internal estimation routines.

Details

1. Optimization strategy.

The algorithm follows a strict hierarchical rule system:

- Remove Heywood items (ψ < -tol or |loading| > 1).
- Remove near-Heywood items (ψ ~ 0).
- Resolve cross-loadings by removing items with smallest ambiguity gap.
- Enforce minimum items per factor.
- If structure is acceptable but RMSEA > target: remove the weakest-loading item.
- If `use_global = TRUE`: evaluate all subsets up to `max_drop` using the composite loss.

2. Adaptive composite loss (df x N).

Weights for RMSEA / SRMR / CFI adapt dynamically according to:

- degrees of freedom (df),
- sample size (N),
- estimator (WLSMV boosters),
- p-close >= .05 (bonus).

This allows stable decisions even when RMSEA is unreliable (df < 5).

3. Interfactor correlation rule.

After each iteration and at finalization, factor correlations are checked:

- If any $|\phi_{ij}|$ < `min_interfactor_correlation`, a warning is printed.

4. Global subset search.

If enabled:

- evaluates all subsets of size 1 ... k,
- uses safe-combination caps,
- shows a progress bar,
- accepts subset removals only if composite loss strictly improves.

5. AI conceptual analysis.

If `use_ai_analysis = TRUE`:

- removed items receive a narrative justification using loadings, ambiguity gaps, h^2, ψ, RMSEA-at-removal, and algorithmic reason,
- retained items can also be evaluated,
- exponential-backoff retry logic ensures robustness,
- a detailed elimination timeline is integrated into the prompt.

Value

A list containing:

- `final_structure`: Thresholded loading matrix.
- `removed_items`: Items removed (in chronological order).
- `steps_log`: Full elimination log with the following columns:
  - `step`: Iteration number
  - `removed_item`: Name of the removed item
  - `reason`: Reason for removal
  - `rmsea`: RMSEA value at the moment of removal (scaled when WLSMV is used)
  - `srmr`: SRMR value at the moment of removal
  - `cfi`: CFI value at the moment of removal (scaled when WLSMV is used)
- `iterations`: Number of optimization iterations.
- `final_rmsea`: Final RMSEA.
- `bondades_original`: Fit indices from the final model.
- `stop_reason`: Why the loop ended (`"all_criteria_met"`, `"min_items_per_factor_protected"`, `"min_omega_protected"`, `"fit_target_reached"`, `"max_iterations"`, `"not_enough_items"`, `"timeout"`, `"efa_convergence_failed"`, `"fit_zero_no_structural_problem"`).
- `inter_factor_correlation`: Final phi matrix.
- `interfactor_check`: Information about violations.
- `omega_final`: McDonald's omega per factor in the final solution, computed from the primary loadings of the pattern matrix. Always reported, whether or not the floor is active.
- `omega_check`: The floor in force, whether it was met, and `blocked_items` (items the floor kept from being removed).
- `last_h2`, `last_psi`: Final communalities and uniquenesses.
- `last_flags`: Heywood and near-Heywood indicators.
- `conceptual_analysis`: GPT-based narrative analyses.
- `config_used`: Full configuration list.

References

Kenny, D. A., & McCoach, D. B. (2003). Effect of the number of variables on measures of fit in structural equation modeling. *Structural Equation Modeling, 10*(3), 333-351. doi:10.1207/S15328007SEM1003_1

Shi, D., Lee, T., & Maydeu-Olivares, A. (2019). Understanding the model size effect on SEM fit indices. *Educational and Psychological Measurement, 79*(2), 310-334. doi:10.1177/0013164418783530

See Also

`print_conceptual_analysis`, `export_conceptual_analysis`

Examples

```
data(Data_Personality)
res <- efa_boosting(
  data       = Data_Personality,
  name_items = "PPTQ",
  n_factors  = 3,
  verbose    = TRUE
)
res$removed_items
res$final_structure
```

---

efa_optimizer *Exploratory Factor Analysis Optimizer with Optional LLM Support*

Description

Automatically refines an Exploratory Factor Analysis (EFA) solution by combining global fit (scaled RMSEA), item-level loading quality, and explicit detection of Heywood/near-Heywood cases. At each iteration the routine re-fits the model, removes the worst offending item (by structural fault or RMSEA improvement), and stops when predefined fit/structure criteria are met or no admissible improvement remains. Optionally, it can call a Large Language Model (LLM) to generate plain-language justifications for removing/keeping items, with language control and detail level.

Usage

```
efa_optimizer(
  data,
  name_items,
  item_range = NULL,
  n_factors = 3,
  exclude_items = NULL,
  # Thresholds (includes Heywood tolerances)
  thresholds = list(
    rmsea = 0.08,
    loading = 0.30,
    min_items_per_factor = 3,
    heywood_tol = 1e-6,
    near_heywood = 0.015
  ),
  # Model configuration
  model_config = list(
    estimator = "WLSMV",
    rotation  = "oblimin"
  ),
  # AI (optional)
  use_ai_analysis = FALSE,
```

```
  ai_config = list(
    api_key = NULL,
    generate_names = FALSE,
    only_removed = TRUE,
    gpt_model = "gpt-3.5-turbo",
    language = "english",
    analysis_detail = "detailed",
    domain_name = "Default Domain",
    scale_title = "Default Scale Title",
    construct_definition = "",
    model_name = "EFA Model",
    item_definitions = NULL
  ),
  verbose = TRUE, ...
)
```

Arguments

`data` A `data.frame` containing observed variables.

`name_items` Common item prefix (e.g., `"DP"` or `"PPTQ"`).

`item_range` Integer vector of length 2 with the first/last item indices; if `NULL`, items are discovered by the prefix followed by a trailing integer.

`n_factors` Number of factors to extract.

`exclude_items` Character vector of items to exclude at the start.

`thresholds` List of decision thresholds:

- **rmsea** Maximum acceptable scaled RMSEA.
- **loading** Minimum salient absolute loading.
- **min_items_per_factor** Minimum items retained per factor (based on primary loading).
- **heywood_tol** Tolerance for detecting negative uniqueness ($\psi < -$`heywood\_tol`) or impossible loadings.
- **near_heywood** Band around zero uniqueness treated as near-Heywood.

`model_config` List of EFA fitting options:

- **estimator** Estimator (e.g., `"WLSMV"`).
- **rotation** Rotation method (e.g., `"oblimin"`).

`use_ai_analysis` Logical; if `TRUE`, requests LLM justifications without affecting the optimization path.

`ai_config` List with LLM settings:

- **api_key** API key (character).
- **generate_names** If `TRUE`, may also request short factor names (if implemented).
- **only_removed** If `TRUE`, justify only removed items; otherwise also justify retained items.
- **gpt_model** LLM model identifier.

language Language for justifications; supports at least `"english"` and `"spanish"`.

analysis_detail Verbosity level for the LLM answer; one of `"brief"`, `"standard"`, `"detailed"`.

`domain_name`, `scale_title`, `construct_definition`, `model_name` Context strings injected into prompts.

item_definitions Named list mapping item IDs to plain-language item stems/descriptions used by the LLM.

`verbose` Logical; if `TRUE`, prints progress, current structure, counts per factor, and a progress bar during LLM analysis.

`...` Additional arguments forwarded to the package's internal EFA engine.

Details

Iteration logic.

1. Fit an EFA using the package's internal EFA engine on the current item set (oblique rotation by default).
2. Extract the loading matrix L and the factor correlation matrix Φ (identity if unavailable).
3. Derive communalities $h^2 = \mathrm{diag}(L\Phi L')$ and uniquenesses $\psi = 1 - h^2$. Flag:
   - Heywood ($\psi < -$`heywood\_tol` or impossible loadings),
   - near-Heywood (ψ within `[-near\_heywood, near\_heywood]`),
   - no salient loading (all $|loading| <$ `loading`),
   - cross-loadings (2+ salient loadings; severity scored by the gap between the two largest absolute loadings).
4. Enforce `min_items_per_factor` using the primary (largest absolute) loading; rows with no salient loading do not count toward any factor.
5. Removal policy (priority order):
   (a) Remove the most severe Heywood item.
   (b) Else remove the worst near-Heywood item.
   (c) Else, if RMSEA exceeds `thresholds$rmsea`, simulate single-item removals (that preserve `min\_items\_per\_factor`) and drop the candidate yielding the best RMSEA improvement.
   (d) Else remove the structurally worst item (by diagnostic score).
6. Stop when RMSEA $\leq$ threshold and all factors meet `min\_items\_per\_factor`, or when no admissible removal improves fit/structure, or when the maximum number of steps is reached.

LLM justifications (optional). If `use_ai_analysis = TRUE` and `item_definitions` are provided, the function can call an LLM via **httr** to draft plain-language justifications. The language (`ai_config$language`) and verbosity (`ai_config$analysis_detail`) control the style and length. The request includes, when available, item-level technical statistics (primary loading, h^2, reason for removal, and RMSEA at the removal step). Transient server errors (e.g., HTTP 502/503/504) and rate limits (HTTP 429) are handled with up to three attempts and exponential backoff. This analysis is purely descriptive and does not influence the optimization path.

Dependencies. If LLM is used, **httr** and **jsonlite** are required; the function attempts to install them if missing.

Value

A list with:

final_structure Data frame with the final, thresholded loadings and item labels.

removed_items Character vector of removed items, in order.

steps_log Data frame with `step`, `removed_item`, `reason`, `rmsea` (value at that step).

iterations Number of iterations executed.

final_rmsea Scaled RMSEA at termination.

bondades_original Fit indices returned by `EFA_modern`.

specifications Model specifications returned by `EFA_modern`.

inter_factor_correlation Estimated Φ (identity if unavailable or `n_factors == 1`).

last_h2 Vector of communalities from the last iteration.

last_psi Vector of uniquenesses from the last iteration.

last_flags List with logical vectors `heywood` and `near` for the last iteration.

conceptual_analysis If LLM used, a list with:

- **removed** Named list of justifications for removed items (or `NULL`).
- **kept** Named list of justifications for retained items if `only_removed = FALSE` (or `NULL`).
- **item_stats** Named list of per-item statistics passed to the LLM (primary loading, h^2, removal reason, RMSEA at removal).

config_used Echo of `thresholds`, `model_config`, `use_ai_analysis`, and `ai_config` (including `language` and `analysis_detail`).

See Also

the package's internal EFA engine

Examples

```
# Run without LLM support
data(Data_Personality)
res <- efa_optimizer(
  data       = Data_Personality,
  name_items = "PPTQ",
  n_factors  = 3,
  thresholds = list(
    rmsea = 0.08, loading = 0.30, min_items_per_factor = 3,
    heywood_tol = 1e-6, near_heywood = 0.015
  ),
  model_config = list(estimator = "WLSMV", rotation = "oblimin"),
  verbose = FALSE
)
res$final_structure
res$steps_log
res$final_rmsea

## Not run:
```

```
# LLM justification (requires a valid API key and item definitions)
res_ai <- efa_optimizer(
  data       = Data_Personality,
  name_items = "PPTQ",
  n_factors  = 3,
  use_ai_analysis = TRUE,
  ai_config = list(
    api_key = Sys.getenv("OPENAI_API_KEY"),
    item_definitions = as.list(stats::setNames(
      paste("Item", 1:15, "content"), paste0("PPTQ", 1:15))),
    language = "spanish",
    analysis_detail = "standard",
    only_removed = TRUE
  ),
  verbose = TRUE
)
str(res_ai$conceptual_analysis, max.level = 1)

## End(Not run)
```

`export_cfa_boosting` *Export CFA Boosting Results to Structured Data Frames*

Description

Export CFA Boosting results to structured data frames suitable for saving as CSV, embedding in a manuscript table, or further analysis.

Usage

```
export_cfa_boosting(result)
```

Arguments

`result`	Output from `cfa_boosting`.

Value

A named list of data frames (`fit_indices`, `standardized_loadings`, `factor_correlations`, `reliability`, `steps_log`).

See Also

`cfa_boosting`, `print_cfa_boosting`

Examples

```
data(Data_Personality, package = "OptimalFactor")
model <- '
F1 =~ PPTQ1 + PPTQ2 + PPTQ3 + PPTQ4 + PPTQ5
F2 =~ PPTQ6 + PPTQ7 + PPTQ8 + PPTQ9 + PPTQ10
F3 =~ PPTQ11 + PPTQ12 + PPTQ13 + PPTQ14 + PPTQ15
'
res <- cfa_boosting(Data_Personality, model,
                    model_config = list(estimator = "MLR", ordered = FALSE),
                    verbose = FALSE)
out <- export_cfa_boosting(res)
names(out)

# Each component is a data.frame ready for write.csv()
head(out$fit_indices)
head(out$loadings)
head(out$steps)

# Save the data frames to a temporary directory.
for (nm in c("fit_indices", "loadings", "reliability"))
  if (!is.null(out[[nm]]))
    utils::write.csv(out[[nm]],
                     file.path(tempdir(), paste0("cfa_boost_", nm, ".csv")),
                     row.names = FALSE)
```

`export_conceptual_analysis`

Export LLM Conceptual Analyses to a UTF-8 Text File

Description

Writes the formatted conceptual analyses produced by `efa_optimizer` to a UTF-8 encoded text file. Internally calls `print_conceptual_analysis` with `width = 100` and `show_stats = TRUE`, capturing its console output via `sink()`. After writing, emits a short confirmation via `message()` and invisibly returns the output file path.

Usage

```
export_conceptual_analysis(resultado, file)
```

Arguments

`resultado`	A result list returned by `efa_optimizer`; must contain `$conceptual_analysis` and `$config_used` (see `print_conceptual_analysis`).
`file`	Character path to the output text file (UTF-8). Required (there is no default, so nothing is written to the working directory unless asked); for a throw-away file use, e.g., `file.path(tempdir(), "conceptual_analysis.txt")`. If it exists, it will be overwritten.

Details

What gets written. The function captures the full console rendering of `print_conceptual_analysis` (headings, section dividers, emojis, warnings, and technical stats) and writes it to `file` using UTF-8 encoding. Because `print_conceptual_analysis` is bilingual, headings will appear in Spanish when `resultado$config_used$ai_config$language` is `"spanish"` (or `"español"`); otherwise, English is used.

Formatting choices. Text is wrapped at 100 characters per line and per-item technical statistics are always included (primary loading, h^2, RMSEA at removal, and removal reason) when available.

Encoding and Unicode. Output is forced to UTF-8. Unicode symbols used by `print_conceptual_analysis` (e.g., `a bar chart symbol`, `a warning sign`, `a cross mark`) will be preserved; ensure your editor is set to UTF-8 to avoid mojibake.

Side effects and limitations. The function uses `sink(file, encoding = "UTF-8")` followed by `sink()` to restore the console. If an error interrupts execution before the second `sink()`, the connection may remain diverted; call `sink()` manually to reset if needed.

Value

Invisibly returns the `file` path (character scalar). A brief confirmation message is also emitted with `message()`, of the form "Análisis exportado a: <file>".

See Also

`print_conceptual_analysis`, `efa_optimizer`

Examples

```
# Example mock object (see print_conceptual_analysis() examples for structure)
resultado <- list(
  conceptual_analysis = list(
    removed = list(DP1 = "Desde el punto de vista psicométrico..."),
    kept = list(DP2 = "Psychometrically, the item shows..."),
    item_stats = list(DP1 = list(loading = 0.22, h2 = 0.11,
                                 rmsea_at_removal = 0.095, reason = "Cross-loading"))
  ),
  config_used = list(
    ai_config = list(
      language = "spanish",
      item_definitions = list(
        DP1 = "La gente generalmente piensa que soy inmoral porque soy LGBT.",
        DP2 = "Me tratan diferente porque no soy heterosexual."
      )
    )
  )
)

out <- export_conceptual_analysis(resultado,
                                  file = file.path(tempdir(), "conceptual_analysis.txt"))
out  # invisible return; prints the path if typed
# Read first lines to verify encoding and content:
head(readLines(out, encoding = "UTF-8"))
```

`imprimir_items_eliminados` *Impresión de Ítems Eliminados*

Description

Imprime en pantalla un mensaje que lista los ítems eliminados. Si el vector de ítems está vacío, indica explícitamente que no hubo eliminaciones.

Usage

```
imprimir_items_eliminados(removed_items)
```

Arguments

`removed_items`	`character` vector. Nombres de los ítems que fueron eliminados.

Details

La función construye un mensaje uniendo los elementos de `removed_items` con comas. - Si `length(removed_items) > 0`, muestra la lista de nombres. - Si `length(removed_items) == 0`, muestra "Ninguno". Finalmente, antepone la etiqueta "Ítems eliminados: " y añade un salto de línea.

Value

`NULL`. La función se usa por su efecto de impresión en consola.

Author(s)

Dr. José Ventura-León

Examples

```
# Con ítems eliminados
imprimir_items_eliminados(c("PPTQ5", "PPTQ10", "PPTQ15"))
#> Ítems eliminados: PPTQ5, PPTQ10, PPTQ15

# Sin ítems eliminados
imprimir_items_eliminados(character(0))
#> Ítems eliminados: Ninguno
```

item_stability *Resampling Stability of the EFA-Boosting Item Selection*

Description

Runs the whole efa_boosting pipeline on many resamples of the data and records how often each item survives, how often it is dropped, and how stable its factor assignment is. This is the empirical answer to the classic objection against data-driven item purification: that it capitalizes on chance (MacCallum, Roznowski & Necowitz, 1992). An item removed in 97 out of 100 resamples is a defensible removal; one removed in 55 is a coin flip dressed as a decision.

Usage

```
item_stability(
  data,
  name_items,
  n_factors = 3,
  R = 100,
  method = c("subsample", "bootstrap"),
  subsample_frac = 0.8,
  reference = NULL,
  seed = NULL,
  n_cores = 1,
  timeout = 120,
  verbose = TRUE,
  ...
)

## S3 method for class 'item_stability'
print(x, digits = 3, ...)
```

Arguments

data	Data frame with the item responses.
name_items	Item name prefix, as in efa_boosting.
n_factors	Number of factors to extract. Default 3.
R	Number of resamples. Default 100. Runtime is roughly R times the cost of a single efa_boosting() call, so start small.
method	Resampling scheme: "subsample" (default) or "bootstrap". Subsampling without replacement is the default because it is what the stability selection literature uses (Meinshausen & Buhlmann, 2010) and because resampling with replacement damages ordinal data specifically: duplicated cases distort the polychoric correlations and thresholds the estimator depends on. On a sample of 100 a bootstrap draw can leave 63 distinct cases, and the near singular matrix that follows sends WLSMV into fits that run for minutes. Measured on

Data_Personality with R = 40 over 8 cores: 5m46s with 3 replications hitting the timeout under bootstrap, against 2m31s with none under subsampling.

The cost is that each replication sees subsample_frac of the data, so the instability reported is mildly conservative: it errs towards calling a decision unstable, never towards endorsing one.

subsample_frac — Fraction of rows drawn when method = "subsample". Default 0.8.

reference — Optional efa_boosting() result on the full sample, used as the alignment reference. If NULL (default) it is computed internally.

seed — Optional integer seed for reproducibility; NULL (default) leaves the RNG untouched.

n_cores — Number of parallel workers. Default 1 (sequential). Values above 1 use a PSOCK cluster and require the package to be installed, not merely loaded with pkgload::load_all().

timeout — Seconds allowed per replication. Default 120; NULL removes the cap. Resampling puts the pipeline on datasets nobody would analyse by hand: a bootstrap draw of N = 100 can repeat rows until only 63 remain distinct, and a WLSMV fit on such a sample can grind for many minutes while every other replication waits. A capped replication returns the best model reached so far, with stop_reason = "timeout", and is counted in the printed summary. Needs the R.utils package.

verbose — Print a progress line per replication. Default TRUE.

... — Ignored.

x — An item_stability object.

digits — Number of digits for the printed rates. Default 3.

Details

Two resampling schemes are available. "bootstrap" draws n cases with replacement, which is the usual choice for assessing sampling variability. "subsample" draws subsample_frac * n cases without replacement, which is more conservative with ordinal data because it cannot duplicate rare response patterns and therefore triggers fewer empty categories under WLSMV.

Because rotation labels factors arbitrarily, the factor assignment of each replication is aligned to a reference solution (the pipeline run once on the complete sample) before agreement is computed. Alignment is greedy: factors are matched by the largest overlap of retained items, most overlapping pair first. factor_agreement is then the proportion of replications, among those where the item was retained, in which the item landed on its modal factor.

Replications that fail (non-convergence, empty response categories after resampling) are counted in n_failed and excluded from the rates, so every proportion is computed over successful replications only.

Value

An object of class item_stability: a list with retention (one row per item: times retained, retention and removal rates, modal factor and factor agreement), stop_reasons (table of the stop reason across replications), n_removed (distribution of how many items each replication dropped), reference (the full-sample result), n_valid, n_failed and call.

References

MacCallum, R. C., Roznowski, M., & Necowitz, L. B. (1992). Model modifications in covariance structure analysis: The problem of capitalization on chance. *Psychological Bulletin, 111*(3), 490–504.

See Also

efa_boosting, cross_validate_cfa, simulate_recovery

Examples

```
data(Data_Expectativas)
st <- item_stability(Data_Expectativas, "EAF", n_factors = 2, R = 3,
                     timeout = 15, seed = 1)
st                      # printed summary
st$retention            # per-item detail
```

---

local_fit_search *Local-Fit Search for a Unidimensional Model*

Description

Searches for a specification of a one-factor model that reaches the fit targets, combining two actions — dropping items and freeing residual covariances — and then cleaning up the result.

Usage

```
local_fit_search(
  data,
  items,
  factor_name = "F",
  item_text = NULL,
  rmsea_target = 0.08,
  cfi_target = 0.95,
  srmr_target = 0.08,
  omega_min = 0.7,
  min_loading = 0.35,
  min_items = 5,
  max_ec = 4,
  mi_min = 5,
  start_ec = character(0),
  estimator = "WLSMV",
  ordered = TRUE,
  verbose = TRUE
)
```

Arguments

`data`	Data frame with the item responses.
`items`	Character vector of candidate item names.
`factor_name`	Name given to the latent variable. Default `"F"`.
`item_text`	Optional data frame with columns `Item` and `Texto` (or `Text`); used only to print the wording of the items involved in each retained covariance, so that its substantive plausibility can be judged.
`rmsea_target`, `cfi_target`, `srmr_target`	Fit targets. Defaults `0.08`, `0.95` and `0.08`.
`omega_min`	Reliability floor. Default `0.70`.
`min_loading`	Loading floor. Default `0.35`.
`min_items`	Minimum number of retained items. Default `5`.
`max_ec`	Maximum number of residual covariances. Default `4`.
`mi_min`	Minimum modification index for a covariance to be considered. Default `5`.
`start_ec`	Character vector of residual covariances to start from, in lavaan syntax (e.g. `"IT1 ~~ IT2"`). Default none.
`estimator`, `ordered`	Passed to `cfa`.
`verbose`	Logical; print progress. Default `TRUE`.

Details

Applies to the scale that is essentially unidimensional but whose one-factor model misfits. Where `redundancy_short_form` attacks a single cause (local dependence between near-duplicate items) by pruning, this routine also considers keeping the pair and modelling its residual covariance, and decides between the two by their effect on fit.

Four phases, three of which encode decisions usually taken by hand.

Phase 0 — sanitation. Items below the loading floor are removed *without* requiring that fit improve. An item that does not load cannot stay, and dropping it often worsens RMSEA temporarily because the degrees of freedom lost outweigh the chi-square gained. A rule that demanded immediate improvement would reject the removal and stall the search before it starts.

Phase 1 — greedy search. At each step both actions are evaluated for every candidate — drop an item, or free one residual covariance — and the one that most reduces RMSEA is taken.

Phase 2 — consolidation. If an item takes part in two or more residual covariances, the item is the problem and not the relations: dropping it is attempted, which typically resolves both at once and buys back a degree of freedom.

Phase 3 — parsimony. Each retained covariance is removed in turn; those that are not needed to keep meeting the targets are discarded. Every freed parameter is a debt that has to be justified.

What the function does *not* do is judge whether a covariance makes substantive sense. It returns its estimate, confidence interval, sign and — when `item_text` is supplied — the wording of both items, so that decision stays with whoever knows the construct. A *negative* residual covariance between items at opposite poles of a construct is often the trace of a dimension that the unidimensional model absorbed, and is worth reading before treating it as noise.

Warning. This is a data-driven search and capitalises on chance (MacCallum, Roznowski & Necowitz, 1992). It establishes whether an admissible specification exists; it does not replace validation on an independent sample.

Value

An object of class `local_fit_search`, a list with `best` (the retained model and its indices), `log` (one row per action, with its phase), `ec_detail` (the retained covariances with sign, interval and wording), `meets` and `config`.

References

MacCallum, R. C., Roznowski, M., & Necowitz, L. B. (1992). Model modifications in covariance structure analysis: The problem of capitalization on chance. *Psychological Bulletin, 111*(3), 490-504.

Saris, W. E., Satorra, A., & van der Veld, W. M. (2009). Testing structural equation models or detection of misspecifications? *Structural Equation Modeling, 16*(4), 561-582.

See Also

`redundancy_short_form`, `discriminant_boosting`, `cross_validate_cfa`

Examples

```
data(Data_Expectativas)
res <- local_fit_search(Data_Expectativas, paste0("EAF", 1:10),
         factor_name = "Expectations", estimator = "MLR", ordered = FALSE)
res
res$log
```

`optimal_cfa_with_ai` *Confirmatory Factor Analysis Optimization with AI Assistance*

Description

Automatically refines a user-specified Confirmatory Factor Analysis (CFA) model by iteratively evaluating fit and modification indices. At each step, the function:

1. Fits the CFA model and computes the scaled RMSEA. 2. If RMSEA <= `rmsea_threshold`, stops. 3. Otherwise, identifies all modification indices (MI) > `mi_threshold`. 4. Selects the largest MI and compares the standardized loadings for the two involved items; removes the item with the smaller loading. 5. Records the modification, RMSEA and MI, updates the model syntax by dropping the removed item, and repeats until RMSEA is acceptable, no MI exceed threshold, or `max_steps` is reached.

Optionally, when `analyze_removed = TRUE`, it uses the OpenAI API to generate concise justifications for exclusion or retention of each item, based on provided `item_definitions` and `factor_definitions`.

Usage

```
optimal_cfa_with_ai(
  initial_model,
  data,
  rmsea_threshold      = 0.08,
  mi_threshold         = 3.84,
  max_steps            = 10,
  verbose              = TRUE,
  debug                = FALSE,
  filter_expr          = NULL,
  exclude_items        = character(0),
  analyze_removed      = FALSE,
  api_key              = NULL,
  item_definitions     = NULL,
  domain_name          = "Dominio por Defecto",
  scale_title          = "Título de la Escala por Defecto",
  construct_definition = "",
  model_name           = "Modelo CFA",
  gpt_model            = "gpt-3.5-turbo",
  factor_definitions   = NULL,
  ...
)
```

Arguments

`initial_model` A character string with the lavaan-style CFA model syntax.

`data` A `data.frame` containing the observed variables.

`rmsea_threshold` Numeric. Maximum acceptable scaled RMSEA to stop refinement.

`mi_threshold` Numeric. Minimum modification index value to consider.

`max_steps` Integer. Maximum number of refinement iterations.

`verbose` Logical. If `TRUE`, prints progress messages.

`debug` Logical. If `TRUE`, prints additional debugging information.

`filter_expr` An expression to subset `data` before fitting (optional).

`exclude_items` Character vector of items to never remove.

`analyze_removed` Logical. If `TRUE`, performs AI-driven conceptual analysis.

`api_key` String. OpenAI API key (required if `analyze_removed = TRUE`).

`item_definitions` Named list mapping each item to its textual content for AI prompts.

`domain_name` Character. Domain or factor label used in AI prompts.

`scale_title` Character. Scale title used in AI prompts.

`construct_definition` Character. Brief construct definition for AI context.

`model_name` Character. Label for the CFA model in AI prompts.

`gpt_model` Character. Name of the ChatGPT model (e.g., `"gpt-3.5-turbo"`).

`factor_definitions` Named list of factor descriptions, indexed by factor name.

`...` Additional arguments passed to `cfa`.

Details

The function proceeds through these phases:

1. **Initialization:** Parses the initial model syntax into factor and item lists.
2. **Filtering:** Optionally subsets `data` via `filter_expr`.
3. **Iterative refinement (up to `max_steps`):**
   (a) Fit the current CFA model.
   (b) Compute scaled RMSEA; if <= `rmsea_threshold`, stop.
   (c) Extract all MI > `mi_threshold`; if none, stop.
   (d) Select the single largest MI, compare the two involved items' standardized loadings, and remove the item with the lower loading.
   (e) Update the model syntax by dropping that item and log the step.
4. **Final fit:** Stores the last fitted `lavaan` object and final RMSEA/CFI.
5. **Conceptual analysis (optional):** For each removed and retained item, calls the OpenAI API to generate justification text.

Value

A list with elements:

`final_model` Character. The refined CFA model syntax.

`final_fit` A `lavaan` object of the last fitted model.

`log` Data frame recording each refinement step (`step`, `modification`, `mi_value`, `rmsea`).

`removed_items` Character vector of items removed.

`alternative_rmsea` Numeric. Final scaled RMSEA.

`final_cfi` Numeric. Final scaled CFI.

`conceptual_analysis` List with sublists `removed` and `kept` containing AI-generated texts, or `NULL` if not performed.

Author(s)

Dr. José Ventura-León

Examples

```
data(Data_Personality)
model <- '
F1 =~ PPTQ1 + PPTQ2 + PPTQ3 + PPTQ4 + PPTQ5
F2 =~ PPTQ6 + PPTQ7 + PPTQ8 + PPTQ9 + PPTQ10
F3 =~ PPTQ11 + PPTQ12 + PPTQ13 + PPTQ14 + PPTQ15
'
# Run the optimization without AI
res_cfa <- optimal_cfa_with_ai(
  initial_model   = model,
  data            = Data_Personality,
  rmsea_threshold = 0.08,
  max_steps       = 5
)
res_cfa$removed_items
res_cfa$log
cat(res_cfa$final_model, "\n")

## Not run:
# Run with AI-driven analysis (requires an OpenAI-compatible API key)
res_cfa_ai <- optimal_cfa_with_ai(
  initial_model      = model,
  data               = Data_Personality,
  analyze_removed    = TRUE,
  api_key            = Sys.getenv("OPENAI_API_KEY"),
  item_definitions   = as.list(stats::setNames(
    paste("Item", 1:15, "content"), paste0("PPTQ", 1:15))),
  factor_definitions = list(F1 = "Factor 1 description",
                            F2 = "Factor 2 description",
                            F3 = "Factor 3 description")
)
res_cfa_ai$conceptual_analysis

## End(Not run)
```

`optimal_efa_with_ai` *Exploratory Factor Analysis Optimization with AI Assistance*

Description

Automatically refines an Exploratory Factor Analysis (EFA) solution by combining model-fit criteria (scaled RMSEA) and item-loading quality, and-when requested- integrates AI-generated conceptual analyses and factor naming. At each iteration it:

1. Estimates the specified EFA model on the current set of items.
2. Computes the scaled RMSEA and checks whether it is <= `threshold_rmsea` and every factor retains at least `min_items_per_factor` items.
3. If fit or structure criteria are not met, identifies and removes the item whose exclusion yields the greatest RMSEA improvement or that exhibits structural issues (cross-loading or no loading).

4. Repeats steps 1-3 until both fit and structure criteria are satisfied or `max_steps` iterations are reached.

Optionally, when `analyze_removed = TRUE` and `item_definitions` are provided, it calls the OpenAI API to generate concise justifications for both exclusion and retention of each item, using the provided definitions and the final loading structure as context. If `generate_factor_names = TRUE`, it also prompts the AI to propose brief (1-2 word) tentative names for each factor, taking into account the specified `domain_name`, `scale_title`, and `construct_definition`.

Usage

```
optimal_efa_with_ai(
  data,
  items                    = NULL,
  n_factors                = 5,
  n_items                  = NULL,
  name_items               = "PPTQ",
  estimator                = "WLSMV",
  rotation                 = "oblimin",
  threshold_rmsea          = 0.08,
  threshold_loading        = 0.30,
  min_items_per_factor     = 2,
  apply_threshold          = TRUE,
  max_steps                = NULL,
  verbose                  = TRUE,
  exclude_items            = character(0),
  analyze_removed          = FALSE,
  api_key                  = NULL,
  item_definitions         = NULL,
  domain_name              = "Dominio por Defecto",
  scale_title              = "Título de la Escala por Defecto",
  construct_definition     = "",
  model_name               = "Modelo EFA",
  gpt_model                = "gpt-3.5-turbo",
  generate_factor_names    = FALSE,
  ...
)
```

Arguments

`data`	A `data.frame` containing the observed variables for the EFA.
`items`	Character vector of item names to include; if `NULL`, names are generated using `name_items` and `n_items`.
`n_factors`	Integer. Number of factors to extract (default `5`).
`n_items`	Integer. Number of items per factor when `items` is `NULL`.
`name_items`	Character. Prefix for item names (default `"PPTQ"`).
`estimator`	Character. Estimator to use (e.g., `"WLSMV"`).
`rotation`	Character. Rotation method (e.g., `"oblimin"`).

`threshold_rmsea`
: Numeric. Maximum allowable scaled RMSEA to stop refinement (default `0.08`).

`threshold_loading`
: Numeric. Minimum absolute loading to consider an item well-loaded (default `0.30`).

`min_items_per_factor`
: Integer. Minimum items required per factor (default `2`).

`apply_threshold`
: Logical. If `TRUE`, zeros out loadings below `threshold_loading` in the final solution.

`max_steps`
: Integer or `NULL`. Maximum number of iterations; if `NULL`, set to `length(items) - 1`.

`verbose`
: Logical. If `TRUE`, prints progress and removal decisions.

`exclude_items`
: Character vector of items to exclude from the start.

`analyze_removed`
: Logical. If `TRUE`, performs conceptual analysis for each removed and conserved item via the OpenAI API.

`api_key`
: String. OpenAI API key (required if `analyze_removed = TRUE` or `generate_factor_names = TRUE`).

`item_definitions`
: Named list mapping each item to its text definition for AI prompts.

`domain_name`
: Character. Domain or factor context used in AI prompts.

`scale_title`
: Character. Scale title used in AI prompts.

`construct_definition`
: Character. Brief definition of the construct used in AI prompts.

`model_name`
: Character. Label for the EFA model in AI prompts.

`gpt_model`
: Character. Name of the ChatGPT model to use (e.g., `"gpt-3.5-turbo"`).

`generate_factor_names`
: Logical. If `TRUE`, triggers factor naming via AI.

`...`
: Additional arguments passed to the package's internal EFA engine.

Details

The function proceeds as follows:

1. Fits the exploratory models with the package's internal EFA engine.
2. Determines the initial set of items from `items` or from `name_items` and `n_items`.
3. Enters an iterative loop:
   (a) Estimates the EFA model with the current items.
   (b) Computes the scaled RMSEA.
   (c) Evaluates the factor-loading structure for cross-loadings or lack of loadings.
   (d) If RMSEA <= `threshold_rmsea` and each factor has >= `min_items_per_factor`, stops.
   (e) Otherwise, removes the item whose exclusion most improves RMSEA or that has the worst structural issue.

4. If analyze_removed = TRUE, for each excluded and conserved item, generates via OpenAI concise justifications based on the final structure.
5. If generate_factor_names = TRUE and api_key provided, proposes tentative names for each factor via AI, considering the specified domain_name, scale_title, and construct_definition.

Value

A list with components:

final_structure
: A data.frame of final item loadings and factor assignments.

removed_items
: Character vector of items removed during refinement.

steps_log
: A data.frame recording each step: step, removed_item, reason, and rmsea.

iterations
: Integer. Total number of iterations performed.

final_rmsea
: Numeric. Final scaled RMSEA after the last iteration.

bondades_original
: Original fit indices and other model information from the package's internal EFA engine.

specifications
: Model specifications returned by the package's internal EFA engine.

conceptual_analysis
: A list with elements removed and kept, each a named list of AI-generated texts per item, or NULL if analyze_removed = FALSE.

factor_names
: Named character vector or list of AI-proposed factor names, or NULL if generate_factor_names = FALSE.

Examples

```
data(Data_Expectativas)
# Run the optimized EFA without AI
res_efa <- optimal_efa_with_ai(
  data       = Data_Expectativas,
  items      = paste0("EAF", 1:10),
  name_items = "EAF",
  n_factors  = 2,
  verbose    = FALSE
)
res_efa$removed_items
res_efa$final_structure

## Not run:
# With AI-assisted conceptual analysis (requires an OpenAI-compatible API key)
construct_def <- paste("Positive consequences of completing a degree, such as",
                       "prestige, income and self-evaluation.")
item_defs <- as.list(stats::setNames(
  paste("Content of item", 1:10), paste0("EAF", 1:10)))
res_efa_ai <- optimal_efa_with_ai(
  data                 = Data_Expectativas,
  items                = paste0("EAF", 1:10),
  name_items           = "EAF",
```

```
  n_factors             = 2,
  analyze_removed       = TRUE,
  api_key               = Sys.getenv("OPENAI_API_KEY"),
  item_definitions      = item_defs,
  construct_definition  = construct_def,
  scale_title           = "EAF Scale",
  model_name            = "EFA Multidimensional",
  generate_factor_names = TRUE,
  domain_name           = "Academic outcome expectations"
)
res_efa_ai$conceptual_analysis
res_efa_ai$factor_names

## End(Not run)
```

---

plot.simulate_cfa_recovery

Plot a CFA Recovery Simulation

Description

Plot a CFA Recovery Simulation

Usage

```
## S3 method for class 'simulate_cfa_recovery'
plot(
  x,
  metric = c("exact_rate", "sensitivity", "specificity", "convergence_rate",
    "mean_retained", "mean_covs_added"),
  reference = 0.9,
  ...
)
```

Arguments

x	A simulate_cfa_recovery object.
metric	Which column of x$summary to draw. Default "exact_rate".
reference	Horizontal reference line for the proportion metrics. Default 0.90; NULL removes it.
...	Ignored.

Value

A ggplot object.

See Also

simulate_cfa_recovery

plot.simulate_recovery

Plot a Recovery Simulation

Description

Draws the recovery curve of a `simulate_recovery` design: the chosen metric against sample size, one line per population loading and one panel per number of items per factor. This is the figure a validation paper reports, so the crossed design is worth running with enough replications for the curve to be stable.

Usage

```
## S3 method for class 'simulate_recovery'
plot(
  x,
  metric = c("recovery_rate", "sensitivity", "specificity", "convergence_rate",
    "mean_retained"),
  reference = 0.9,
  ...
)
```

Arguments

`x`	A `simulate_recovery` object.
`metric`	Which column of `x$summary` to draw: `"recovery_rate"` (default), `"sensitivity"`, `"specificity"`, `"convergence_rate"` or `"mean_retained"`.
`reference`	Horizontal reference line, drawn only for the proportion metrics. Default 0.90; `NULL` removes it.
`...`	Ignored.

Value

A `ggplot` object.

See Also

`simulate_recovery`

Examples

```
sim <- simulate_recovery(n = 200, n_reps = 2, seed = 1)
plot(sim)
plot(sim, metric = "sensitivity")
```

print.discriminant_boosting

Print method for discriminant_boosting

Description

Print method for discriminant_boosting

Usage

```
## S3 method for class 'discriminant_boosting'
print(x, ...)
```

Arguments

x	Output from discriminant_boosting.
...	Ignored.

Value

Invisibly returns x; called for its side effects.

print.local_fit_search

Print method for local_fit_search

Description

Print method for local_fit_search

Usage

```
## S3 method for class 'local_fit_search'
print(x, ...)
```

Arguments

x	Output from local_fit_search.
...	Ignored.

Value

Invisibly returns x; called for its side effects.

print.specification_search *Print method for specification_search*

Description

Print method for specification_search

Usage

```
## S3 method for class 'specification_search'
print(x, top = 10, ...)
```

Arguments

x	An object returned by specification_search.
top	Number of rows to print. Default 10.
...	Ignored.

print_cfa_boosting *Print CFA-Boosting Results*

Description

Formatted console printer for the object returned by cfa_boosting. Displays fit indices, the iteration trail, factor loadings (above a threshold), latent correlations, reliability coefficients and the final lavaan model syntax.

Usage

```
print_cfa_boosting(
  result,
  show_loadings = TRUE,
  show_correlations = TRUE,
  show_reliability = TRUE,
  show_steps = TRUE,
  show_model = TRUE,
  loading_threshold = 0.30,
  digits = 3
)
```

Arguments

`result`	Output from `cfa_boosting`.
`show_loadings`	Logical. Print standardized factor loadings of the final model. Default `TRUE`.
`show_correlations`	Logical. Print latent factor correlations. Default `TRUE`.
`show_reliability`	Logical. Print omega and alpha by factor. Default `TRUE`.
`show_steps`	Logical. Print the per-iteration log of move / drop / cov operations and the resulting fit indices. Default `TRUE`.
`show_model`	Logical. Print the final lavaan model syntax. Default `TRUE`.
`loading_threshold`	Numeric. Loadings below this absolute value are hidden in the printed table. Default `0.30`.
`digits`	Integer. Number of decimal places. Default `3`.

Value

Invisibly returns `result`; called for its side effects on the console.

See Also

`cfa_boosting`, `export_cfa_boosting`

Examples

```
data(Data_Personality, package = "OptimalFactor")
model <- '
F1 =~ PPTQ1 + PPTQ2 + PPTQ3 + PPTQ4 + PPTQ5
F2 =~ PPTQ6 + PPTQ7 + PPTQ8 + PPTQ9 + PPTQ10
F3 =~ PPTQ11 + PPTQ12 + PPTQ13 + PPTQ14 + PPTQ15
'
res <- cfa_boosting(Data_Personality, model,
                    model_config = list(estimator = "MLR", ordered = FALSE),
                    verbose = FALSE)

# Full printout (default).
print_cfa_boosting(res)

# Compact printout - hide steps log, raise the loading threshold, and
# show fewer decimals when reporting in a slide deck.
print_cfa_boosting(res,
  show_steps        = FALSE,
  loading_threshold = 0.40,
  digits            = 2)
```

print_cfa_results — *Print Results from AI-Assisted CFA Refinement*

Description

Prints a structured report of key outputs from `optimal_cfa_with_ai` applied to a Confirmatory Factor Analysis (CFA), including available components, modification log, final model syntax, removed items, fit indices, standardized loadings, and reliability.

Usage

```
print_cfa_results(res)
```

Arguments

`res`	A `list` returned by `optimal_cfa_with_ai` (CFA version), containing at minimum the components: `log`, `final_model`, `removed_items`, `final_rmsea`, `final_cfi`, and `final_fit`.

Details

The function prints the following sections to the console:

1. **Available components**: names of all elements in `res`.
2. **Modification log**: data frame `res\$log` with each removal step.
3. **Final model**: the CFA model syntax stored in `res\$final_model`.
4. **Removed items**: list of items removed during refinement.
5. **Final fit measures**: `res\$final_rmsea` and `res\$final_cfi`.
6. **Additional fit indices**: retrieved via `lavaan::fitMeasures(res$final_fit, c("chisq.scaled", "df.scaled", "srmr", "wrmr", "cfi.scaled", "tli.scaled", "rmsea.scaled"))`, including the scaled chi-square statistic, scaled degrees of freedom, SRMR, WRMR, scaled CFI, scaled TLI, and scaled RMSEA.
7. **Standardized loadings**: extracted from `lavaan::standardizedsolution(res$final_fit)` and filtered to measurement paths (`op == "=~"`), showing the final factor loadings for each indicator.
8. **Reliability**: composite reliability estimates computed with `semTools::compRelSEM(res$final_fit, tau.eq = FALSE, ord.scale = TRUE)`, indicating the internal consistency of each factor under an ordinal measurement model.

Value

Invisibly returns `NULL`. The primary purpose is to print results.

Author(s)

Dr. José Ventura-León

Examples

```
data(Data_Personality)
model <- '
F1 =~ PPTQ1 + PPTQ2 + PPTQ3 + PPTQ4 + PPTQ5
F2 =~ PPTQ6 + PPTQ7 + PPTQ8 + PPTQ9 + PPTQ10
F3 =~ PPTQ11 + PPTQ12 + PPTQ13 + PPTQ14 + PPTQ15
'
res_cfa <- optimal_cfa_with_ai(model, Data_Personality, max_steps = 5,
                               verbose = FALSE)
print_cfa_results(res_cfa)
```

---

print_conceptual_analysis

Pretty-print LLM Conceptual Analyses for Removed/Retained Items

---

Description

Formats and prints the item-level conceptual analyses produced by `efa_optimizer` when `use_ai_analysis = TRUE`. Outputs bilingual (English/Spanish) section headers according to `resultado$config_used$ai_config$language`, wraps lines to the requested console width, optionally shows per-item technical statistics, and warns if the LLM text appears truncated or contains API error messages. The function does not modify objects and returns `invisible(NULL)`.

Usage

```
print_conceptual_analysis(resultado, width = 80, show_stats = FALSE)
```

Arguments

`resultado`	A result list returned by `efa_optimizer` containing `$conceptual_analysis`, `$config_used`, and (optionally) `$conceptual_analysis$item_stats`.
`width`	Integer console width in characters used to wrap paragraphs (passed to `strwrap`).
`show_stats`	Logical; if `TRUE`, prints technical statistics (primary loading, h^2, RMSEA at removal, and removal reason) when available for each item.

Details

What is printed. The function prints two sections, when available:

1. Conceptual analysis of *removed* items.
2. Conceptual analysis of *retained* items (if `ai_config$only_removed = FALSE` in `efa_optimizer`).

For each item, the output shows:

- A header line with the item ID and its plain-language definition from `resultado$config_used$ai_config$item_defi`
- Optional **TECHNICAL INFORMATION / INFORMACIÓN TÉCNICA** (when `show_stats = TRUE`) including primary loading, h^2, RMSEA at removal, and the recorded reason (e.g., "Cross-loading", "Heywood").

- A three-part narrative derived from the LLM text:
  1. *Psychometric Problems/Strengths* (for removed/retained items, respectively),
  2. *Conceptual Misalignment/Alignment*,
  3. *Removal/Retention Benefit*.

Section detection and cleaning. The function attempts to segment the LLM paragraph into the three sections using multilingual keyword heuristics (Spanish/English). If keywords are not detected, it splits the text into three approximately equal blocks (by sentences). Basic cleaning removes Markdown markers (e.g., `**`) and numbering artifacts and ensures sentence-final punctuation if missing.

Truncation and error handling. Heuristics flag potentially truncated analyses (e.g., very short text, missing terminal punctuation, or trailing cut words) and print a warning note. If known API error patterns are present (e.g., "HTTP 503", "Rate limit", "Connection error"), an explicit error block is printed with a suggestion to rerun the analysis for that item.

Language and headings. Headings and status messages are printed in Spanish when `resultado$config_used$ai_config$` is `"spanish"` (or `"español"`); otherwise, English is used.

Unicode. The console output includes simple Unicode characters (e.g., box rules, emojis like `a bar chart symbol`, `a warning sign`, `a cross mark`). If your console does not support UTF-8, some symbols may render as placeholders.

Value

`invisible(NULL)`. The function is used for its side effects (console printing).

See Also

`efa_optimizer`

Examples

```
# Minimal mock object illustrating the expected structure:
resultado <- list(
  conceptual_analysis = list(
    removed = list(
     DP1 = "Desde el punto de vista psicométrico, el ítem muestra carga baja y cruces..."
    ),
    kept = list(
      DP2 = "Psychometrically, the item shows strong primary loading and specificity..."
    ),
    item_stats = list(
    DP1 = list(loading = 0.22, h2 = 0.11, rmsea_at_removal = 0.095, reason = "Cross-loading")
    )
  ),
  config_used = list(
    ai_config = list(
      language = "spanish",
      item_definitions = list(
        DP1 = "La gente generalmente piensa que soy inmoral porque soy LGBT.",
        DP2 = "Me tratan diferente porque no soy heterosexual."
      )
```

```
    )
  )
)

# Print in Spanish with technical stats:
print_conceptual_analysis(resultado, width = 80, show_stats = TRUE)

# Switch to English:
resultado$config_used$ai_config$language <- "english"
print_conceptual_analysis(resultado, width = 70, show_stats = FALSE)
```

`print_efa_results` *Print Results from AI-Assisted EFA Refinement*

Description

Prints a structured summary of the key outputs from `optimal_efa_with_ai`, including the removed items, final RMSEA, iteration count, final factor structure, step-by-step log, and final fit indices.

Usage

```
print_efa_results(res)
```

Arguments

`res`	A `list` returned by `optimal_efa_with_ai`, containing at least the components `removed_items`, `final_rmsea`, `iterations`, `final_structure`, `steps_log`, and `bondades_original`.

Details

The function prints to the console in the following order:

1. A "Summary" block with:
   - `removed_items`: the items removed during refinement.
   - `final_rmsea`: the scaled RMSEA after the last iteration.
   - `iterations`: total number of iterations performed.
2. "Factor Structure" showing the final loadings/data frame.
3. "Step Log" displaying the data frame of each removal step.
4. "Final Fit Indices" printing the `bondades_original` object as returned by the package's internal EFA engine.

Value

invisibly returns `NULL`. Used for its printing side effect.

Author(s)

Dr. José Ventura-León

Examples

```
data(Data_Expectativas)
res_efa <- optimal_efa_with_ai(Data_Expectativas, items = paste0("EAF", 1:10),
                               name_items = "EAF", n_factors = 2,
                               verbose = FALSE)
print_efa_results(res_efa)
```

---

redundancy_short_form *Redundancy-Guided Short Form of a Unidimensional Scale*

Description

Builds a short form of an (essentially) unidimensional scale by iteratively removing the most locally-dependent item. At each step a one-factor model is fitted, the pair of items with the largest residual correlation is located, and the item of that pair with the *lower* loading is dropped (keeping the stronger indicator). Pruning stops when the target length `k` is reached or when no residual correlation exceeds `threshold`. This addresses the common situation where a scale is unidimensional but the one-factor model misfits because of near-duplicate items (local dependence).

Usage

```
redundancy_short_form(
  data,
  items,
  k = NULL,
  groups = NULL,
  min_per_group = 3,
  threshold = 0.15,
  min_omega = NULL,
  estimator = "WLSMV",
  ordered = TRUE
)
```

Arguments

`data`	Data frame with the item responses.
`items`	Character vector with the candidate item names.
`k`	Target number of items. If `NULL`, prunes until the largest residual correlation drops below `threshold`. Default `NULL`.
`groups`	Optional named list mapping content groups to items (e.g. theoretical dimensions), used only to preserve at least `min_per_group` items per group during pruning. Default `NULL` (no constraint).

min_per_group	Minimum items kept per group in groups. Default 3.
threshold	Residual-correlation stopping threshold when k = NULL. Default 0.15.
min_omega	Reliability floor. When set (e.g. 0.80), an item is only dropped if the resulting form keeps McDonald's omega at or above this value, so the short form cannot buy fit at the cost of internal consistency. A scale that already sits below the floor is not frozen: for it the effective bar is its current omega, so removals that do not reduce reliability are still allowed. NULL (default) disables the check.
estimator	Estimator passed to lavaan::cfa. Default "WLSMV".
ordered	Logical; treat items as ordered. Default TRUE.

Details

Removing redundant items is preferred over piling up residual covariances, which capitalizes on chance (MacCallum, Roznowski & Necowitz, 1992). The resulting short form should be cross-validated on an independent sample (see cross_validate_cfa).

Value

A list with items (retained items), trajectory (data frame with n_items, cfi, tli, rmsea, srmr, omega and the item dropped at each step), fit (final one-factor lavaan object), loadings (standardized), omega (McDonald's omega of the final form) and stop_reason ("target_k", "threshold", "min_items", "min_per_group" or "min_omega").

References

MacCallum, R. C., Roznowski, M., & Necowitz, L. B. (1992). Model modifications in covariance structure analysis: The problem of capitalization on chance. *Psychological Bulletin, 111*(3), 490–504.

See Also

cross_validate_cfa

Examples

```
data(Data_Expectativas)
sf <- redundancy_short_form(Data_Expectativas, paste0("EAF", 1:10), k = 6,
        groups = list(A = paste0("EAF", 1:5), B = paste0("EAF", 6:10)))
sf$trajectory; sf$items; sf$omega
```

report_cfa_results *Report the CFA boosting results*

Description

Counterpart of report_efa_results for objects returned by cfa_boosting. Side-effect: writes a human-readable report to the console. Return value: an invisible list (class "cfa_boost_report") with the same information structured for programmatic use, including a text field with the formatted lines so any frontend can re-render exactly what the console showed.

Usage

```
report_cfa_results(res, show_plot = TRUE, print = TRUE)
```

Arguments

res	A list produced by cfa_boosting(). Expected fields include removed_items, added_covariances, fit_indices, targets_met, iterations, standardized_loadings, factor_correlations, reliability, steps_log, final_syntax.
show_plot	Logical. If TRUE (default), include ASCII charts of RMSEA / CFI / SRMR evolution.
print	Logical. If TRUE (default), write the formatted report to the console. Set to FALSE when only the structured list is needed.

Details

Fields of the returned list:

type "cfa_boosting"

summary Single-row list with iterations, n_removed_items, n_added_covariances, targets_all_met

fit_indices Final-model rmsea, cfi, tli, srmr, chisq, df

targets_met Logical flags by index from res$targets_met

removed_items Character vector

added_covariances Character vector with "x ~~ y" strings

standardized_loadings Loadings data.frame

factor_correlations Phi matrix

reliability Reliability table (composite/AVE/etc.)

steps_log Per-iteration log

final_syntax The final lavaan syntax used

text Character vector - same lines that were printed

Value

An invisible list with class "cfa_boost_report" - see Details below for fields.

See Also

cfa_boosting, report_efa_results

Examples

```
data(Data_Personality, package = "OptimalFactor")
# Run CFA boosting first to obtain an object suitable for the reporter.
model <- '
F1 =~ PPTQ1 + PPTQ2 + PPTQ3 + PPTQ4 + PPTQ5
F2 =~ PPTQ6 + PPTQ7 + PPTQ8 + PPTQ9 + PPTQ10
F3 =~ PPTQ11 + PPTQ12 + PPTQ13 + PPTQ14 + PPTQ15
'
res <- cfa_boosting(Data_Personality, model,
                    model_config = list(estimator = "MLR", ordered = FALSE),
                    verbose = FALSE)

# Pretty print to the console (default).
report_cfa_results(res)

# Capture the structured output without printing - useful inside Shiny
# apps or scripts that need the data programmatically.
rep <- report_cfa_results(res, print = FALSE)
str(rep, max.level = 1)
```

---

report_efa_results *Console Report for EFA Optimization Results*

Description

Prints a compact, Spanish-language console report summarizing the iterative optimization performed by efa_boosting. The report includes a process summary table, three independent ASCII evolution plots (RMSEA, SRMR, CFI), a stepwise removal log, the final thresholded loading structure, and common fit indices with rule-of-thumb interpretations. Designed for quick inspection in interactive sessions; returns `invisible(NULL)`.

Usage

```
report_efa_results(res, show_plot = TRUE, print = TRUE)
```

Arguments

`res` A result list as returned by efa_boosting, expected to contain:

- `$final_structure`: data frame of thresholded factor loadings and item labels.
- `$removed_items`: character vector of removed items in order.

- `$steps_log`: data frame with columns `step`, `removed_item`, `reason`, and-if available-`rmsea`, `srmr`, `cfi`.
- `$final_rmsea`: final scaled RMSEA.
- `$iterations`: number of iterations executed.
- `$bondades_original`: fit indices (columns: `chisq.scaled`, `df.scaled`, `rmsea.scaled`, `cfi.scaled`, `tli.scaled`, `srmr`).
- `$inter_factor_correlation`: optional factor correlation matrix.
- `$interfactor_check`: optional list describing correlation-threshold checks.

`show_plot` Logical; if `TRUE`, prints three ASCII evolution bars (RMSEA, SRMR, CFI).

`print` Logical; if `TRUE` (default), the formatted report is printed to the console. If `FALSE`, the structured report object is returned without printing.

Details

Headers and language. All headings and labels are printed in Spanish (e.g., "RESUMEN DEL PROCESO", "EVOLUCIÓN DEL RMSEA", "EVOLUCIÓN DEL SRMR", "EVOLUCIÓN DEL CFI", "ÍNDICES DE AJUSTE DEL MODELO FINAL").

Process summary. The function computes and prints:

- Initial item count (`n_initial`) and final count (`n_final`).
- Percent retained.
- Initial and final RMSEA.
- Percent reduction in RMSEA.
- Iterations executed.

These appear in a compact two-column table.

ASCII evolution plots (RMSEA, SRMR, CFI). When `show_plot = TRUE`, the function prints three separate bar-style ASCII diagnostics:

- *RMSEA evolution:* bar length increases with RMSEA, scaled to the reference range 0.05–0.13.
- *SRMR evolution:* bar length increases with SRMR, scaled to 0.03–0.10.
- *CFI evolution:* bar length increases with CFI *(higher = better)*, scaled linearly between 0.70 and 1.00.

Each block includes an ASCII axis guide:

- RMSEA: ticks at 0.05, 0.08, 0.10, 0.13.
- SRMR: ticks at 0.03, 0.05, 0.08, 0.10.
- CFI: ticks at 0.70, 0.80, 0.90, 0.95, 1.00.

Bars are drawn using `a full block` and capped at a configurable length (default: 20 characters).

Item removal log. The stepwise elimination table mirrors `steps_log`, formatting available fit indices (RMSEA, SRMR, CFI) to three decimals.

Final structure. Prints the thresholded loading matrix produced by `efa_boosting` (loadings < 0.30 appear as 0).

Final fit indices and qualitative labels. Extracts the last row of `bondades_original` and prints:

- Scaled χ^2, scaled df, RMSEA, CFI, TLI, and SRMR.
- Quick qualitative labels:
  - RMSEA: <= .05 = Excelente, <= .08 = Bueno, <= .10 = Mediocre, else Pobre.
  - CFI/TLI: >= .95 = Excelente, >= .90 = Bueno, else Pobre.
  - SRMR: <= .08 = Bueno, else Pobre.

These labels are intended for rapid screening and should be interpreted in context.

Unicode. The report uses UTF-8 box characters (a full block) and accented text. If your console lacks UTF-8 support, glyphs may degrade.

Value

invisible(NULL). This function prints to the console and does not return structured output.

See Also

efa_boosting, print_conceptual_analysis, export_conceptual_analysis

Examples

```
data(Data_Expectativas)
res <- efa_boosting(
  data       = Data_Expectativas,
  name_items = "EAF",
  n_factors  = 2,
  verbose    = FALSE
)
report_efa_results(res, show_plot = TRUE)
rep <- report_efa_results(res, print = FALSE)
names(rep)
```

---

simulate_cfa_recovery *Monte Carlo Recovery of a Known Structure with CFA-Boosting*

Description

The confirmatory counterpart of simulate_recovery. Data are simulated from a known population, a simple structure model over every item is handed to cfa_boosting, and the question is how often the pipeline removes the items that deserve removal and only those.

Usage

```
simulate_cfa_recovery(
  n = 500,
  loading = 0.65,
  items_per_factor = 5,
  n_factors = 3,
```

```
  phi = 0.3,
  n_cross = 1,
  cross_loading = 0.4,
  n_low = 1,
  low_loading = 0.2,
  n_categories = 5,
  skew = c("symmetric", "skewed"),
  n_reps = 100,
  seed = NULL,
  verbose = TRUE,
  n_cores = 1,
  timeout = 120,
  ...
)

## S3 method for class 'simulate_cfa_recovery'
print(x, ...)
```

Arguments

`n`	Sample size(s). Vector values are crossed into conditions. Default 500.
`loading`	Population loading(s) of the good items. Default 0.65.
`items_per_factor`	Number of good items per factor. Default 5.
`n_factors`	Number of factors in the population model. Default 3.
`phi`	Interfactor correlation. Default 0.30.
`n_cross`	Number of cross-loading items added to the pool (they load `cross_loading` on factors 1 and 2). Default 1.
`cross_loading`	Loading of the cross-loading items on both factors. Default 0.40.
`n_low`	Number of weak items added to the pool. Default 1.
`low_loading`	Loading of the weak items. Default 0.20.
`n_categories`	Number of ordered response categories. Default 5.
`skew`	Category threshold shape: `"symmetric"` (default) or `"skewed"`.
`n_reps`	Replications per condition. Default 100.
`seed`	Optional integer seed for reproducibility; NULL (default) leaves the RNG untouched.
`verbose`	Print progress per condition. Default `TRUE`.
`n_cores`	Number of worker processes used to fit the replications of a condition. Default 1 (sequential). A crossed design is hundreds of `efa_boosting()` runs, so this is usually the difference between minutes and hours; `parallel::detectCores() - 1` is a practical choice. The datasets are always generated sequentially from `seed`, so results do not depend on the number of cores.
`timeout`	Seconds allowed per replication. Default 120; `NULL` removes the cap. A single pathological dataset can otherwise grind for many minutes and stall the condition it belongs to. A capped replication is recorded with `stop_reason = "timeout"`. Needs the `R.utils` package.

...	Ignored.
x	A simulate_cfa_recovery object.

Details

The question is not the same one the exploratory simulation asks, and the difference is not cosmetic. In simulate_recovery the structure is discovered, so recovery means every retained item landed on its true factor. Here the structure is imposed by the researcher, so the assignment cannot be wrong by construction and what is at stake is the item selection: the specified model deliberately includes the contaminated items, and a pipeline that works removes them without taking good items with them.

Two further quantities are worth watching. covs_added counts the residual covariances the pipeline introduced: the population has none, so every one of them is a false positive, which is the modification index objection of MacCallum, Roznowski and Necowitz (1992) turned into a number. And exact is deliberately strict, requiring the removed set to equal the contaminated set exactly, because sensitivity and specificity can both look respectable while no single replication got the answer right.

The cross loading item is specified on the first factor and the weak item as well, which is what a researcher would do when writing the model without knowing which items are faulty.

Value

An object of class simulate_cfa_recovery: a list with summary (one row per condition), replications, conditions, population, model (the syntax handed to the pipeline) and call.

See Also

simulate_recovery, cfa_boosting

Examples

```
sim <- simulate_cfa_recovery(n = 300, n_reps = 2, seed = 1)
sim
plot(sim, metric = "sensitivity")
```

---

simulate_recovery *Monte Carlo Recovery of a Known Factor Structure*

Description

Simulates ordinal data from a known population model, runs the efa_boosting pipeline on each replication, and reports how often the algorithm recovers the true structure. Conditions can be crossed over sample size, loading size and number of items per factor, which is what turns this into evidence about *when* the method works rather than a single anecdote.

Usage

```
simulate_recovery(
  n = 500,
  loading = 0.65,
  items_per_factor = 5,
  n_factors = 3,
  phi = 0.3,
  n_cross = 1,
  cross_loading = 0.4,
  n_low = 1,
  low_loading = 0.2,
  n_categories = 5,
  skew = c("symmetric", "skewed"),
  n_reps = 100,
  seed = NULL,
  verbose = TRUE,
  n_cores = 1,
  timeout = 120,
  n_factors_fitted = NULL,
  ...
)

## S3 method for class 'simulate_recovery'
print(x, ...)
```

Arguments

`n`	Sample size(s). Vector values are crossed into conditions. Default 500.
`loading`	Population loading(s) of the good items. Default 0.65.
`items_per_factor`	Number of good items per factor. Default 5.
`n_factors`	Number of factors in the population model. Default 3.
`phi`	Interfactor correlation. Default 0.30.
`n_cross`	Number of cross-loading items added to the pool (they load `cross_loading` on factors 1 and 2). Default 1.
`cross_loading`	Loading of the cross-loading items on both factors. Default 0.40.
`n_low`	Number of weak items added to the pool. Default 1.
`low_loading`	Loading of the weak items. Default 0.20.
`n_categories`	Number of ordered response categories. Default 5.
`skew`	Category threshold shape: `"symmetric"` (default) or `"skewed"`.
`n_reps`	Replications per condition. Default 100.
`seed`	Optional integer seed for reproducibility; NULL (default) leaves the RNG untouched.
`verbose`	Print progress per condition. Default `TRUE`.

`n_cores` Number of worker processes used to fit the replications of a condition. Default 1 (sequential). A crossed design is hundreds of `efa_boosting()` runs, so this is usually the difference between minutes and hours; `parallel::detectCores() - 1` is a practical choice. The datasets are always generated sequentially from `seed`, so results do not depend on the number of cores.

`timeout` Seconds allowed per replication. Default 120; `NULL` removes the cap. A single pathological dataset can otherwise grind for many minutes and stall the condition it belongs to. A capped replication is recorded with `stop_reason = "timeout"`. Needs the `R.utils` package.

`n_factors_fitted` Number of factors the pipeline is told to extract. Default `NULL`, meaning the true `n_factors`. Vector values are crossed into conditions, which is how the question a reviewer will ask gets answered: every other condition here hands the algorithm the true number of factors, and in practice nobody knows it. Fitting fewer factors than the population has cannot recover the structure by definition, so `recovery_rate` goes to zero and the informative columns become `sensitivity` and `specificity`: what the pipeline does to the items when the dimensionality is wrong.

`...` Ignored.

`x` A `simulate_recovery` object.

Details

The population model has `n_factors` correlated factors (correlation `phi`), `items_per_factor` good items with loading `loading`, plus optional contaminated items: `n_cross` items that load `cross_loading` on two factors and `n_low` items whose loading is `low_loading`. Continuous responses are drawn from the implied correlation matrix and discretized into `n_categories` ordered categories, either symmetrically or with a skew.

Four quantities are recorded per replication:

`recovered` TRUE when every retained good item sits on the factor it truly belongs to, after aligning labels (rotation names factors arbitrarily), and no factor was lost.

`sensitivity` proportion of contaminated items correctly removed. `NA` when the condition has no contaminated items.

`specificity` proportion of good items correctly retained.

`n_retained`, `rmsea`, `stop_reason` final size, fit and why the pipeline stopped.

Sensitivity and specificity are the honest way to read the pipeline: an algorithm that removes everything scores a perfect sensitivity and a terrible specificity, so both must be read together.

Value

An object of class `simulate_recovery`: a list with `summary` (one row per condition: recovery rate, mean sensitivity and specificity, mean items retained, convergence rate), `replications` (the raw per-replication data frame), `conditions`, `population` (the loading matrix, factor correlations and item roles of the last condition) and `call`.

See Also

efa_boosting, item_stability

Examples

```
# single condition, quick check (use n_reps >= 100 in real studies)
sim <- simulate_recovery(n = 200, n_reps = 2, seed = 1)
sim$summary
```

---

specification_search_theory

Theory-Guided Specification Search for CFA Models

Description

Theory-aware extension of specification_search. The classic specification search (MacCallum, 1986) is driven purely by fit: its loss only looks at RMSEA/CFI/SRMR, so it "capitalizes on chance" and can drift toward models that fit but break the intended theory (moving items to the wrong factor, or dropping core items of the construct). This function adds a **theory-congruence** term to the loss that penalizes departures from the theoretical item-to-factor assignment and penalizes dropping theoretical items. A single parameter theory_weight grades how much theory counts relative to fit (0 reproduces the fit-only search; larger values make the search progressively more conservative).

Usage

```
specification_search_theory(
  data,
  items,
  theory,
  theory_weight = 0.5,
  tw_move = 0.6,
  tw_drop = 0.4,
  tw_k = 0,
  seeds = NULL,
  max_factors = 4,
  min_items_factor = 2,
  cfi_target = 0.95,
  rmsea_target = 0.08,
  srmr_target = 0.08,
  max_iter_per_config = 40,
  max_covs = 5,
  max_items_removed = 6,
  try_bifactor = TRUE,
  operations = c("move", "drop", "cov"),
  estimator = "WLSMV",
  ordered = TRUE,
```

```
  std.lv = TRUE,
  mi_min = 10,
  mi_top = 3,
  only_within_factor = FALSE,
  loss_weights = c(rmsea = 0.5, cfi = 0.3, srmr = 0.2),
  patience = 8,
  early_stop_after_meet = 3,
  n_cores = 1,
  verbose = TRUE
)
```

Arguments

`data` Data frame with the observed item responses.

`items` Character vector with the item names to search over (must all exist in `data`).

`theory` Named list encoding the theoretical structure, of the form `list(FactorA = c("it1","it2",...), FactorB = c(...))`. Items not listed are treated as having no theoretical factor (neither rewarded nor penalized).

`theory_weight` Non-negative scalar weighting the theory term against the fit term. `0` reproduces the fit-only search; typical values 0.5-1 recover theory-congruent models with equivalent fit. Default 0.5.

`tw_move`, `tw_drop`, `tw_k`
Internal weights of the theory loss: penalty for misassigned items, for dropped theoretical items, and for deviating from the theoretical number of factors. Defaults 0.6, 0.4, 0.0.

`seeds`, `max_factors`, `min_items_factor`, `cfi_target`, `rmsea_target`, `srmr_target`
As in `specification_search`. If `seeds = NULL`, the theoretical structure plus block-partition seeds are generated.

`max_iter_per_config`, `max_covs`, `max_items_removed`, `try_bifactor`, `operations`
As in `specification_search`.

`estimator`, `ordered`, `std.lv`, `mi_min`, `mi_top`, `loss_weights`, `patience`, `early_stop_after_meet`, `verbose`
As in `specification_search`.

`only_within_factor`
Restrict the `cov` operation to pairs of items belonging to the same factor. Default `FALSE`, which reproduces the behaviour of earlier versions.

A residual correlation between items of *different* factors is rarely what it looks like. It is usually the signature of an omitted cross-loading, a misassigned item or a missing factor, and it absorbs covariance that belongs at the factor level: the estimated interfactor correlation drops and discriminant validity looks better than it is. Measured on a two-factor scale, freeing two cross-factor residuals moved the factor correlation from .956 to .919 and pulled the upper bound of its confidence interval from 1.031 down to .999, which is the difference between two factors that cannot be told apart and two that can.

TRUE is the conservative choice. It does not make cross-factor residuals wrong in principle, since item adjacency and shared method can justify them, but it forces those to be added deliberately rather than by modification index.

n_cores Number of CPU cores for parallel evaluation of the candidate models within each greedy iteration (which are independent). Default 1 (sequential). Values > 1 use a PSOCK cluster (works on Windows, where fork is unavailable); results are identical to the sequential run, only faster. A practical choice is parallel::detectCores() - 1.

Details

The total loss minimized for each candidate model is

$$L = L_{fit} + \texttt{theory_weight} \cdot L_{theory}$$

where L_{fit} is the composite fit loss of specification_search

$$L_{fit} = w_{rmsea} \max(0, (RMSEA - t_{rmsea})/0.03) + w_{cfi} \max(0, (t_{cfi} - CFI)/0.03) + w_{srmr} \max(0, (SRMR - t_{srmr})/0.$$

and the theory loss is

$$L_{theory} = \texttt{tw_move} \cdot p_{mis} + \texttt{tw_drop} \cdot p_{drop} + \texttt{tw_k} \cdot |k - k_{theory}|/k_{theory}$$

with p_{mis} the proportion of retained items assigned to a factor that does not match their theoretical dimension, and p_{drop} the proportion of theoretical items dropped from the model. Empirical factors are aligned to theoretical dimensions greedily by maximum item overlap (each factor inherits the most frequent theoretical label among its items), the same criterion used by the project's loading maps. The reported congruence is $1 - p_{mis}$ (1 = the model fully respects the theory). Default seeds include the theoretical structure itself as a starting point.

Warning. Like the fit-only search, this remains an EXPLORATORY device; cross-validate the chosen model on an independent sample.

Value

An object of class specification_search (so print.specification_search applies), a list with table (one row per evaluated configuration, including the extra columns congruence and theory_loss), successful, best (the lowest-total-loss model, with its congruence and theory_loss), results, theory_weight and call.

References

MacCallum, R. C. (1986). Specification searches in covariance structure modeling. *Psychological Bulletin, 100*(1), 107–120.

See Also

specification_search

Examples

```
data(Data_Expectativas)
items  <- paste0("EAF", 1:10)
theory <- list(F1 = paste0("EAF", 1:5), F2 = paste0("EAF", 6:10))
guided <- specification_search_theory(Data_Expectativas, items, theory,
            theory_weight = 0.5, max_factors = 2, max_iter_per_config = 5,
            try_bifactor = FALSE, estimator = "MLR", ordered = FALSE)
guided$table[, c("config", "cfi", "rmsea", "congruence", "loss")]
guided$best$factors
```

---

theory_recovery *How Much of the Theoretical Structure a Solution Recovers*

Description

Compares the item-to-factor partition of a solution with the theoretical key of the instrument and reports how much of the theory survives. It works on the output of any routine in the package (efa_boosting, cfa_boosting, discriminant_boosting, specification_search_theory), on a plain partition, or on a loading matrix, so different algorithms can be compared on the same scale.

Usage

```
theory_recovery(theory, solution, loadings = NULL)
```

Arguments

theory	Named list encoding the theoretical key, of the form `list(Dimension = c("item1", "item2"), ...)`.
solution	The solution to evaluate. One of: a named list factor -> items; a loading matrix or data frame (items in rows, factors in columns; each item is assigned to its largest absolute loading); or the object returned by `efa_boosting()`, `cfa_boosting()`, `discriminant_boosting()`, `specification_search_theory()` or `local_fit_search()`.
loadings	Optional loading matrix (items in rows) used for the Tucker congruence. Taken from `solution` automatically when it is a loading matrix or an `efa_boosting()` result.

Details

No single number answers the question, because a solution can depart from the theory in two ways that one index confounds: it can *drop* items or *move* them to another factor. The metrics are therefore meant to be read together:

retention Proportion of the theoretical items that survive.

accuracy Among the retained items, proportion that sit on the factor matched to their own dimension. Factors are matched to dimensions by the assignment that maximises the number of hits, searched exhaustively, so the result does not depend on the order of the factors.

recovery `retention * accuracy`: proportion of the theoretical items that end up in their place. This is the headline figure.

ari Adjusted Rand Index (Hubert & Arabie, 1985) between the empirical and theoretical partitions of the retained items. It is 0 when the agreement is what chance alone would produce and 1 when it is perfect, and it does not require the numbers of factors to coincide.

tucker_mean Only when loadings are available: mean Tucker congruence between each factor's loadings and the binary key of its matched dimension. Values of .85-.94 indicate fair similarity and .95 or above equality (Lorenzo-Seva & ten Berge, 2006). Absolute loadings are used, so a reverse-keyed item is not penalised for its sign: polarity is a matter of scoring, not of structure.

Accuracy must never be read alone. A confirmatory routine such as `cfa_boosting` cannot move items between factors, only drop them, so its accuracy is 1 by construction and its departure from the theory shows up in retention.

Value

A list with `summary` (one-row data frame with `k_theory`, `k_solution`, `retention`, `accuracy`, `recovery`, `ari` and `tucker_mean`), `per_dimension` (retention and recovery of each theoretical dimension and the empirical factor matched to it), `matching`, `tucker` and `misplaced` (retained items that sit outside their dimension's factor).

References

Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification, 2*(1), 193-218. doi:10.1007/BF01908075

Lorenzo-Seva, U., & ten Berge, J. M. F. (2006). Tucker's congruence coefficient as a meaningful index of factor similarity. *Methodology, 2*(2), 57-64. doi:10.1027/16142241.2.2.57

See Also

`algorithm_stability`, which applies these metrics to every split of a cross-validation.

Examples

```
theory <- list(A = c("x1", "x2", "x3"), B = c("x4", "x5", "x6"))

# x3 moved to the other factor and x6 dropped
sol <- list(F1 = c("x1", "x2"), F2 = c("x3", "x4", "x5"))
theory_recovery(theory, sol)$summary

# A loading matrix works too
L <- matrix(c(.7, .6, .5, .1, .0, .2,
              .1, .0, .2, .8, .7, .6), ncol = 2,
            dimnames = list(paste0("x", 1:6), c("F1", "F2")))
theory_recovery(theory, L)$summary
```

Index

∗ **datasets**
Data_Expectativas, 11
Data_Personality, 11
∗ **print**
print_cfa_results, 44
print_efa_results, 47

algorithm_stability, 3, *62*

bifactor_indices, 5

cfa, *13*, *31*, *34*
cfa_boosting, 7, *24*, *42*, *43*, *50*, *51*, *53*, *55*, *61*, *62*
cross_validate_cfa, *3*, 9, *14*, *30*, *32*, *49*

Data_Expectativas, 11
Data_Personality, 11
discriminant_boosting, 12, *32*, *41*, *61*

efa_boosting, *9*, 14, *28*, *30*, *51–53*, *55*, *58*, *61*
efa_optimizer, 20, *25*, *26*, *45*, *46*
export_cfa_boosting, 24, *43*
export_conceptual_analysis, *19*, 25, *53*

imprimir_items_eliminados, 27
item_stability, *3*, *5*, 28, *58*

local_fit_search, *14*, 30, *41*

optimal_cfa_with_ai, 32
optimal_efa_with_ai, 35

plot.simulate_cfa_recovery, 39
plot.simulate_recovery, 40
print.discriminant_boosting, 41
print.item_stability *(*item_stability*)*, 28
print.local_fit_search, 41
print.simulate_cfa_recovery *(*simulate_cfa_recovery*)*, 53
print.simulate_recovery *(*simulate_recovery*)*, 55
print.specification_search, 42
print_cfa_boosting, *9*, *24*, 42
print_cfa_results, 44
print_conceptual_analysis, *19*, *25*, *26*, 45, *53*
print_efa_results, 47

redundancy_short_form, *9*, *10*, *31*, *32*, 48
report_cfa_results, 50
report_efa_results, *50*, 51, *51*

simulate_cfa_recovery, *39*, 53
simulate_recovery, *30*, *40*, *53*, 55, *55*
specification_search, *42*, *58–60*
specification_search_theory, *14*, 58, *61*

theory_recovery, *3–5*, 61