Package ‘pcFactorStan’

February 17, 2020

**Title**  Stan Models for the Paired Comparison Factor Model  
**Version**  1.4.0  
**Description**  Provides convenience functions and pre-programmed Stan models related to the paired comparison factor model. Its purpose is to make fitting paired comparison data using Stan easy.  
**License**  GPL (>= 3)  
**Encoding**  UTF-8  
**LazyData**  true  
**LinkingTo**  Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), StanHeaders (>= 2.18.0), BH (>= 1.66.0), rstan (>= 2.18.1)  
**Imports**  rstan (>= 2.18), reshape2, mvtnorm, igraph, loo  
**Depends**  R (>= 3.4), methods, Rcpp (>= 0.12.0)  
**Suggests**  knitr, rmarkdown, testthat, shiny, ggplot2, covr, qgraph  
**SystemRequirements**  GNU make  
**VignetteBuilder**  knitr  
**NeedsCompilation**  yes  
**URL**  https://github.com/jpritikin/pcFactorStan  
**BugReports**  https://github.com/jpritikin/pcFactorStan/issues  
**RoxygenNote**  7.0.2  
**Author**  Joshua N. Pritikin [aut, cre] (<https://orcid.org/0000-0002-9862-5484>), Daniel C. Furr [ctb], Trustees of Columbia University [cph]  
**Maintainer**  Joshua N. Pritikin <jpritikin@pobox.com>  
**Repository**  CRAN  
**Date/Publication**  2020-02-17 16:40:06 UTC
Description

**pcFactorStan** makes it easy to fit the paired comparison factor model using **rstan**.

A user will generally want to use **prepData** and **pcStan** to fit a model.

The package includes a number of Stan models (see **findModel** for a list) and an example dataset **phyActFlowPropensity**.

After gaining some experience with the pre-defined models, we anticipate that users may write their own Stan models and fit them with **stan**, for which **pcStan** is a wrapper.
calibrateItems

Determine the optimal scale constant for a set of items

Description

Data are passed through `filterGraph` and `normalizeData`. Then the ‘unidim_adapt’ model is fit to each item individually. A larger `varCorrection` will obtain a more accurate scale, but is also more likely to produce an intractable model. A good compromise is between 2.0 and 4.0.

Usage

```r
calibrateItems(
  df,
  iter = 2000L,
  chains = 4L,
  varCorrection = 3,
  maxAttempts = 5L,
  ...
)
```

Arguments

- `df`: a data frame with pairs of vertices given in columns `pa1` and `pa2`, and item response data in other columns
- `iter`: A positive integer specifying the number of iterations for each chain (including warmup).
- `chains`: A positive integer specifying the number of Markov chains.
- `varCorrection`: A correction factor greater than or equal to 1.0
- `maxAttempts`: How many times to try re-running a model with more iterations.
- `...`: Additional options passed to `stan`.

Value

A data.frame (one row per item) with the following columns:

- `item`: Name of the item
- `iter`: Number of iterations per chain
- `divergent`: Number of divergent transitions observed after warmup
- `treedepth`: Number of times the treedepth was exceeded
- `low_bfmi`: Number of chains with low E-BFMI
- `n_eff`: Minimum effective number of samples across all parameters
- `Rhat`: Maximum Rhat across all parameters
- `scale`: Median marginal posterior of `scale`
- `thetaVar`: Median variance of theta (latent scores)
References

See Also
check_hmc_diagnostics

Examples

```r
result <- calibrateItems(phyActFlowPropensity) # takes more than 5 seconds
print(result)
```

---

**filterGraph**  
*Filter graph to remove vertices that are not well connected*

**Description**

Vertices not part of the largest connected component are excluded (Hopcroft & Tarjan, 1973). Vertices that have fewer than `minAny` edges and are not connected to `minDifferent` or more different vertices are excluded. For example, vertex ‘a’ connected to vertices ‘b’ and ‘c’ will be include so long as these vertices are part of the largest connected component.

**Usage**

```r
filterGraph(df, minAny = 11L, minDifferent = 2L)
```

**Arguments**

- `df`  
a data frame with pairs of vertices given in columns `pa1` and `pa2`, and item response data in other columns
- `minAny`  
the minimum number of edges
- `minDifferent`  
the minimum number of vertices

**Details**

Given that `minDifferent` defaults to 2, if activity $A$ was compared to at least two other activities, $B$ and $C$, then $A$ is retained. The rationale is that, although little may be learned about $A$, there may be a transitive relationship, such as $B < A < C$, by which the model can infer that $B < C$. Therefore, per-activity sample size is less of a concern when the graph is densely connected.

A young novice asked the wise master, "Why is 11 the default `minAny` instead of 10?" The master answered, "Because 11 is a prime number."
findModel

Value

The same graph excluding some vertices.

References


Examples

```r
df <- filterGraph(phyActFlowPropensity[,c(paste0('pa',1:2),'predict'))])
head(df)
```

findModel  
*Given a model name, return stanmodel object*

Description

This is a convenience function to help you look up the path to an appropriate model for your data.

Usage

```r
findModel(model = NULL)
```

Arguments

- `model`: the name of a model

Details

There are essentially three models: ‘unidim’, ‘covariance’, and ‘factor’. ‘unidim’ analyzes a single item. ‘covariance’ is suitable for two or more items. Once you have vetted your items with the ‘unidim’ and ‘covariance’ models, then you can try the ‘factor’ model. For each model, there is a ‘_ll’ variation. This model includes row-wise log likelihoods suitable for feeding to `loo` for efficient approximate leave-one-out cross-validation (Vehtari, Gelman, & Gabry, 2017).

There is also a special model ‘unidim_adapt’. Except for this model, the other models require a scaling constant. To find an appropriate scaling constant, we recommend fitting ‘unidim_adapt’ to each item separately and then take the median of median point estimates to set the scale. ‘unidim_adapt’ requires a varCorrection constant. In general, a varCorrection of 2.0 or 3.0 should provide optimal results.

Since version 1.1.0, the factor model permits an arbitrary number of factors and arbitrary factor-to-item paths. If you were using the old factor model, you’ll need to update your code to call `prepSingleFactorModel`. Arbitrary factor model structure should be specified using `prepFactorModel`. The single factor model is called ‘factor1’ and the general factor model is called ‘factor’.
generateCovItems

Generate paired comparison data with random correlations between items

Description

If you need access to the correlation matrix used to generate the absolute latent scores then you will need to generate them yourself. This is not difficult. See how in the example.

Usage

generateCovItems(df, numItems, th = 0.5, name, ..., scale = 1, alpha = 1)

Arguments

df a data frame with pairs of vertices given in columns pa1 and pa2, and item response data in other columns
numItems how many items to create
th a vector of thresholds
name a vector of item names
... Not used. Forces remaining arguments to be specified by name.
scale a vector of scaling constants
alpha a vector of item discriminations

Value

The given data.frame df with additional columns for each item. In addition, you can obtain the correlation matrix used to generate the latent worths from attr(df,"cor") and and latent worths from attr(df,"worth").
Response model

The paired comparison item response model has thresholds and a scale parameter similar to the partial credit model (Masters, 1982). The model is cumbersome to describe in traditional mathematical notation, but the R code is fairly straightforward,

```r
softmax <- function(y) exp(y) / sum(exp(y))

cmp_probs <- function(scale, alpha, pa1, pa2, thRaw) {
  th <- cumsum(thRaw)
  diff <- scale * (pa2 - pa1)
  unsummed <- c(0, diff + rev(th), diff - th, use.names = FALSE)
  softmax(cumsum(alpha * unsummed))
}
```

The function `cmp_probs` takes a `scale` constant, `alpha` discrimination, the latent scores for two objects `pa1` and `pa2`, and a vector of thresholds `thRaw`. The thresholds are parameterized as the difference from the previous threshold. For example, thresholds `c(0.5, 0.5)` are not at the same location but are at locations `c(0.5, 1.0)`. Thresholds are symmetric. If there is one threshold then the model admits three possible response outcomes (e.g. win, tie, and lose). Responses are always stored centered with zero representing a tie. Therefore, it is necessary to add one plus the number of thresholds to response data to index into the vector returned by `cmp_probs`. For example, if our response data is `-1, 0, 1` and has one threshold then we would add 2 (1 + 1 threshold) to obtain the indices (1, 2, 3).

Use `itemModelExplorer` to explore the item model. In this `shiny` app, the `discrimination` parameter does what is customary in item response models. However, it is not difficult to show that discrimination is a function of thresholds and scale. That is, discrimination is not an independent parameter. In paired comparison models, discrimination and measurement error are confounded.

References


See Also

Other item generators: `generateFactorItems()`, `generateItem()`, `generateSingleFactorItems()`

Examples

```r
library(mvtnorm)
df <- twoLevelGraph(letters[1:10], 100)
df <- generateCovItems(df, 3)

# generateCovItems essentially does the same thing as:
numItems <- 3
palist <- letters[1:10]
trueCor <- cov2cor(rWishart(1, numItems, diag(numItems))[,1])
theta <- rmvnorm(length(palist), sigma=trueCor)
dimnames(theta) <- list(palist, paste0('i', 3 + 1:numItems))
```
df <- generateItem(df, theta)
attr(df, "cor")

generateFactorItems Generate paired comparison data for a factor model

Description
Generate paired comparison data given a mapping from factors to items.

Usage
generateFactorItems(
  df,
  path,
  factorScalePrior,
  th = 0.5,
  name,
  ...,  
  scale = 1,
  alpha = 1
)

Arguments
df a data frame with pairs of vertices given in columns pa1 and pa2, and item response data in other columns
path a named list of item names
factorScalePrior a named numeric vector
th a vector of thresholds
name a vector of item names
... Not used. Forces remaining arguments to be specified by name.
scale a vector of scaling constants
alpha a vector of item discriminations

Details
For each factor, you need to specify its name, which items it predicts, and its scale prior. The connections from factors to items is specified by the 'path' argument. The scale priors are given in the 'factorScalePrior' argument. Both factors and items are specified by name (not index). The example shows how everything fits together. Paths are ordered as given in the 'path' argument.

The units of 'factorScalePrior' is a standard deviation of the normal prior for the logit transformed factor proportion.
Path proportions (factor-to-item loadings) are sampled from a logistic transformed normal distribution with scale `factorScalePrior`. A few attempts are made to resample path proportions if any of the item proportions sum to more than 1.0. An exception will be raised if repeated attempts fail to produce viable proportion assignments.

**Value**

The given data.frame `df` with additional columns for each item. In addition, you can obtain path proportions (factor-to-item loadings) from `attr(df,"pathProp")`, the factor scores from `attr(df,"score")`, and latent worths from `attr(df,"worth")`.

**Response model**

The paired comparison item response model has thresholds and a scale parameter similar to the partial credit model (Masters, 1982). The model is cumbersome to describe in traditional mathematical notation, but the R code is fairly straightforward,

```r
softmax <- function(y) exp(y) / sum(exp(y))
cmp_probs <- function(scale, alpha, pa1, pa2, thRaw) {
  th <- cumsum(thRaw)
  diff <- scale * (pa2 - pa1)
  unsummed <- c(0, diff + rev(th), diff - th, use.names = FALSE)
  softmax(cumsum(alpha * unsummed))
}
```

The function `cmp_probs` takes a `scale` constant, `alpha` discrimination, the latent scores for two objects `pa1` and `pa2`, and a vector of thresholds `thRaw`. The thresholds are parameterized as the difference from the previous threshold. For example, thresholds c(0.5, 0.5) are not at the same location but are at locations c(0.5, 1.0). Thresholds are symmetric. If there is one threshold then the model admits three possible response outcomes (e.g. `win`, `tie`, and `lose`). Responses are always stored centered with zero representing a tie. Therefore, it is necessary to add one plus the number of thresholds to response data to index into the vector returned by `cmp_probs`. For example, if our response data is (-1, 0, 1) and has one threshold then we would add 2 (1 + 1 threshold) to obtain the indices (1, 2, 3).

Use `itemModelExplorer` to explore the item model. In this shiny app, the `discrimination` parameter does what is customary in item response models. However, it is not difficult to show that discrimination is a function of thresholds and scale. That is, discrimination is not an independent parameter. In paired comparison models, discrimination and measurement error are confounded.

**Backward incompatibility**

The function `generateFactorItems` was renamed to `generateSingleFactorItems` (version 1.1.0) to make space for a more flexible factor model with an arbitrary number of factors and arbitrary factor-to-item loading pattern. If you don’t need this flexibility, you can call the old function `generateSingleFactorItems`.
generateItem

References


See Also

To fit a factor model: prepFactorModel

Other item generators: generateCovItems(), generateItem(), generateSingleFactorItems()

Examples

df <- twoLevelGraph(letters[1:10], 100)
df <- generateFactorItems(df, list(f1=paste0("Var",1:4),
                                 f2=paste0("Var",2:4)),
                                 c(f1=0.9, f2=0.5))
head(df)
attr(df, "pathProp")
attr(df, "score")
attr(df, "worth")

---

**generateItem** Generate paired comparison data for one or more items given absolute latent scores

**Description**

To add a single item, theta should be a vector of latent scores. To add multiple items at a time, theta should be a matrix with one item in each column. Item names can be given as the colnames of theta.

The interpretation of theta depends on the context where the data were generated. For example, in chess, theta represents unobserved chess skill that is partially revealed by match outcomes.

The graph can be regarded as undirected, but data are generated relative to the order of vertices within each row. Vertices do not commute. For example, a -1 for vertices ‘a’ and ‘b’ is the same as 1 for vertices ‘b’ and ‘a’.

**Usage**

generateItem(df, theta, th = 0.5, name, ..., scale = 1, alpha = 1)
**Arguments**

- **df**: a data frame with pairs of vertices given in columns `pa1` and `pa2`, and item response data in other columns.
- **theta**: a vector or matrix of absolute latent scores. See details below.
- **th**: a vector of thresholds.
- **name**: a vector of item names.
- **...**: Not used. Forces remaining arguments to be specified by name.
- **scale**: a vector of scaling constants.
- **alpha**: a vector of item discriminations.

**Value**

The given data.frame `df` with additional columns for each item.

**Response model**

The paired comparison item response model has thresholds and a scale parameter similar to the partial credit model (Masters, 1982). The model is cumbersome to describe in traditional mathematical notation, but the R code is fairly straightforward,

```r
softmax <- function(y) exp(y) / sum(exp(y))

cmp_probs <- function(scale, alpha, pa1, pa2, thRaw) {
  th <- cumsum(thRaw)
  diff <- scale * (pa2 - pa1)
  unsummed <- c(0, diff + rev(th), diff - th, use.names = FALSE)
  softmax(cumsum(alpha * unsummed))
}
```

The function `cmp_probs` takes a `scale` constant, `alpha` discrimination, the latent scores for two objects `pa1` and `pa2`, and a vector of thresholds `thRaw`. The thresholds are parameterized as the difference from the previous threshold. For example, thresholds `c(0.5, 0.5)` are not at the same location but are at locations `c(0.5, 1.0)`. Thresholds are symmetric. If there is one threshold then the model admits three possible response outcomes (e.g. `win`, `tie`, and `lose`). Responses are always stored centered with zero representing a tie. Therefore, it is necessary to add one plus the number of thresholds to response data to index into the vector returned by `cmp_probs`. For example, if our response data is `(-1, 0, 1)` and has one threshold then we would add 2 (1 + 1 threshold) to obtain the indices (1, 2, 3).

Use `itemModelExplorer` to explore the item model. In this shiny app, the `discrimination` parameter does what is customary in item response models. However, it is not difficult to show that discrimination is a function of thresholds and scale. That is, discrimination is not an independent parameter. In paired comparison models, discrimination and measurement error are confounded.

**References**

See Also

Other item generators: `generateCovItems()`, `generateFactorItems()`, `generateSingleFactorItems()`

Examples

```r
df <- roundRobinGraph(letters[1:5], 40)
df <- generateItem(df)
```

---

generateSingleFactorItems

*Generate paired comparison data with a common factor that accounts for some proportion of the variance*

Description

Imagine that there are people that play in tournaments of more than one board game. For example, the computer player AlphaZero (Silver et al. 2018) has trained to play chess, shogi, and Go. We can take the tournament match outcome data and find rankings among the players for each of these games. We may also suspect that there is a latent board game skill that accounts for some proportion of the variance in the per-board game rankings.

Usage

```r
generateSingleFactorItems(df, prop, th = 0.5, name, ..., scale = 1, alpha = 1)
```

Arguments

- `df`: a data frame with pairs of vertices given in columns `pa1` and `pa2`, and item response data in other columns
- `prop`: the number of items or a vector of signed proportions of variance
- `th`: a vector of thresholds
- `name`: a vector of item names
- `...`: Not used. Forces remaining arguments to be specified by name.
- `scale`: a vector of scaling constants
- `alpha`: a vector of item discriminations

Value

The given data.frame `df` with additional columns for each item.
Response model

The paired comparison item response model has thresholds and a scale parameter similar to the partial credit model (Masters, 1982). The model is cumbersome to describe in traditional mathematical notation, but the R code is fairly straightforward,

```r
softmax <- function(y) exp(y) / sum(exp(y))

cmp_probs <- function(scale, alpha, pa1, pa2, thRaw) {
  th <- cumsum(thRaw)
  diff <- scale * (pa2 - pa1)
  unsummed <- c(0, diff + rev(th), diff - th, use.names = FALSE)
  softmax(cumsum(alpha * unsummed))
}
```

The function `cmp_probs` takes a `scale` constant, `alpha` discrimination, the latent scores for two objects `pa1` and `pa2`, and a vector of thresholds `thRaw`. The thresholds are parameterized as the difference from the previous threshold. For example, thresholds `c(0.5, 0.5)` are not at the same location but are at locations `c(0.5, 1.0)`. Thresholds are symmetric. If there is one threshold then the model admits three possible response outcomes (e.g. `win`, `tie`, and `lose`). Responses are always stored centered with zero representing a tie. Therefore, it is necessary to add one plus the number of thresholds to response data to index into the vector returned by `cmp_probs`. For example, if our response data is `(-1, 0, 1)` and has one threshold then we would add `2` (1 + 1 threshold) to obtain the indices `(1, 2, 3).

Use `itemModelExplorer` to explore the item model. In this shiny app, the `discrimination` parameter does what is customary in item response models. However, it is not difficult to show that discrimination is a function of thresholds and scale. That is, discrimination is not an independent parameter. In paired comparison models, discrimination and measurement error are confounded.

Backward incompatibility

The function `generateFactorItems` was renamed to `generateSingleFactorItems` (version 1.1.0) to make space for a more flexible factor model with an arbitrary number of factors and arbitrary factor-to-item loading pattern. If you don’t need this flexibility, you can call the old function `generateSingleFactorItems`.

References


See Also

Other item generators: `generateCovItems()`, `generateFactorItems()`, `generateItem()`
Examples

```r
df <- twoLevelGraph(letters[1:10], 100)
df <- generateSingleFactorItems(df, 3)
```

Description

When data `dl` and fitted model `fit` are provided, the item parameters associated with `item` are loaded for inspection.

Usage

```r
itemModelExplorer(dl = NULL, fit = NULL, item = NULL)
```

Arguments

- `dl`: a data list prepared by `prepData`
- `fit`: a `stanfit` object
- `item`: name of the item to visualize

Response model

The paired comparison item response model has thresholds and a scale parameter similar to the partial credit model (Masters, 1982). The model is cumbersome to describe in traditional mathematical notation, but the R code is fairly straightforward,

```r
softmax <- function(y) exp(y) / sum(exp(y))

cmp_probs <- function(scale, alpha, pa1, pa2, thRaw) {
  th <- cumsum(thRaw)
  diff <- scale * (pa2 - pa1)
  unsummed <- c(0, diff + rev(th), diff - th, use.names = FALSE)
  softmax(cumsum(alpha * unsummed))
}
```

The function `cmp_probs` takes a scale constant, alpha discrimination, the latent scores for two objects `pa1` and `pa2`, and a vector of thresholds `thRaw`. The thresholds are parameterized as the difference from the previous threshold. For example, thresholds `c(0.5, 0.5)` are not at the same location but are at locations `c(0.5, 1.0)`. Thresholds are symmetric. If there is one threshold then the model admits three possible response outcomes (e.g. `win`, `tie`, and `lose`). Responses are always stored centered with zero representing a tie. Therefore, it is necessary to add one plus the number of thresholds to response data to index into the vector returned by `cmp_probs`. For example, if our response data is `(-1, 0, 1)` and has one threshold then we would add 2 (1 + 1 threshold) to obtain the indices `(1, 2, 3)`. 
Use **itemModelExplorer** to explore the item model. In this **shiny** app, the *discrimination* parameter does what is customary in item response models. However, it is not difficult to show that discrimination is a function of thresholds and scale. That is, discrimination is not an independent parameter. In paired comparison models, discrimination and measurement error are confounded.

### References


### Examples

```r
itemModelExplorer()  # will launch a browser in RStudio
```

---

**normalizeData**  
Normalize data according to a canonical order

### Description

Pairwise comparison data are not commutative. Alice beating Bob in chess is equivalent to Bob losing to Alice. **normalizeData** assigns an arbitrary order to all vertices and reorders vertices column-wise to match, flipping signs as needed.

### Usage

```r
normalizeData(df, ..., .palist = NULL, .sortRows = TRUE)
```

### Arguments

- `df`: a data frame with pairs of vertices given in columns `pa1` and `pa2`, and item response data in other columns
- `...`: Not used. Forces remaining arguments to be specified by name.
- `.palist`: a character vector giving an order to use instead of the default
- `.sortRows`: logical. Using the same order, sort rows in addition to vertex pairs.

### Examples

```r
df <- data.frame(pa1=NA, pa2=NA, i1=c(1, -1))
df[1, paste0('pa', 1:2)] <- c('a', 'b')
df[2, paste0('pa', 1:2)] <- c('b', 'a')
normalizeData(df)
```
outlierTable

List observations with Pareto values larger than a given threshold

Description

The function prepCleanData compresses observations into the most efficient format for evaluation by Stan. This function maps indices of observations back to the actual observations, filtering by the largest Pareto k values. It is assumed that data was processed by normalizeData or is in the same order as seen by prepCleanData.

Usage

outlierTable(data, x, threshold = 0.5)

Arguments

data  a data list prepared for processing by Stan
x     An object created by loo
threshold    threshold is the minimum k value to include

Value

A data.frame (one row per observation) with the following columns:

- **pa1** Name of object 1
- **pa2** Name of object 2
- **item** Name of item
- **pick** Observed response
- **k** Associated Pareto k value

See Also

toLoo, pareto_k_ids

Examples

```r
palist <- letters[1:10]
df <- twoLevelGraph(palist, 300)
theta <- rnorm(length(palist))
names(theta) <- palist
df <- generateItem(df, theta, th=rep(0.5, 4))

df <- filterGraph(df)
df <- normalizeData(df)
dl <- prepCleanData(df)
dl$scale <- 1.5
```
\[
\begin{align*}
\text{m1} & \leftarrow \text{pcStan}("\text{unidim_11}"; \text{dl}) \\
\text{loo1} & \leftarrow \text{tlooo} (\text{m1}, \text{cores}=1) \\
\text{ot} & \leftarrow \text{outlierTable} (\text{dl}, \text{loo1}, \text{threshold}=0.2) \\
\text{df}[\text{df}p1==\text{ot}[1, \text{'p1'}] \& \text{df}p2==\text{ot}[1, \text{'p2'}], \text{'i1'}]
\end{align*}
\]

\section*{parDistributionCustom}

\textit{Produce data suitable for plotting parameter distributions}

\subsection*{Description}

Produce data suitable for plotting parameter distributions

\subsection*{Usage}

\begin{verbatim}
parDistributionCustom(
  fit, 
  pars, 
  nameVec, 
  label = \text{withoutIndex(pars[1])}, 
  samples = 500 
)
\end{verbatim}

\begin{verbatim}
parDistributionFor(fit, pi, samples = 500)
\end{verbatim}

\subsection*{Arguments}

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit</td>
<td>a \textit{stanfit} object</td>
</tr>
<tr>
<td>pars</td>
<td>a vector of parameter names</td>
</tr>
<tr>
<td>nameVec</td>
<td>a vector of explanatory parameters names</td>
</tr>
<tr>
<td>label</td>
<td>column name for nameVec</td>
</tr>
<tr>
<td>samples</td>
<td>number of posterior samples</td>
</tr>
<tr>
<td>pi</td>
<td>a data.frame returned by \textit{parInterval}</td>
</tr>
</tbody>
</table>

\subsection*{Value}

A data.frame with the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample</td>
<td>Sample index</td>
</tr>
<tr>
<td>label</td>
<td>A name from nameVec</td>
</tr>
<tr>
<td>value</td>
<td>A single sample of the associated parameter</td>
</tr>
</tbody>
</table>

\subsection*{See Also}

Other data extractor: \textit{parInterval()}, \textit{responseCurve()}
parInterval

*Produce data suitable for plotting parameter estimates*

**Description**

Produce data suitable for plotting parameter estimates

**Usage**

```r
parInterval(fit, pars, nameVec, label = withoutIndex(pars[1]), width = 0.8)
```

**Arguments**

- `fit` a *stanfit* object
- `pars` a vector of parameter names
- `nameVec` a vector of explanatory parameters names
- `label` column name for `nameVec`
- `width` a width in probability units for the uncertainty interval

**Value**

A data.frame with the following columns:

- L Lower quantile
- M Median
- U Upper quantile
- `label` `nameVec`

**See Also**

Other data extractor: `parDistributionCustom()`, `responseCurve()`

**Examples**

```r
vignette('manual', 'pcFactorStan')
```
pcStan

Fit a paired comparison Stan model

Description

Uses `findModel` to find the appropriate model and then invokes `sampling`.

Usage

pcStan(model, data, ...)

Arguments

- `model`: the name of a model
- `data`: a data list prepared for processing by Stan
- `...`: Additional options passed to `stan`.

Value

A `stanfit` object.

An object of S4 class `stanfit`.

See Also

See `sampling`, for which this function is a wrapper, for additional options. See `prepData` to create a suitable data list. See `print.stanfit` for ways of getting tables summarizing parameter posteriors.

`calibrateItems`, `outlierTable`

Examples

```r
dl <- prepData(phyActFlowPropensity[,c(1,2,3)])
dl$varCorrection <- 2.0
pcStan('unidim_adapt', data=d1) # takes more than 5 seconds
```

phyActFlowPropensity  Physical activity flow propensity
Description
A dataset containing paired comparisons of 87 physical activities on 16 flow-related facets. Participants submitted two activities using free-form input. These activities were substitute into item templates. For example, the ‘predict’ item asked, “How predictable is the action?” with response options:

- A1 is much more predictable than A2.
- A1 is somewhat more predictable than A2.
- Both offer roughly equal predictability.
- A2 is somewhat more predictable than A1.
- A2 is much more predictable than A1.

Most items were adapted from Jackson & Eklund (2002).

Usage
phyActFlowPropensity

Format
A data.frame with one row per activity comparison and items in the columns. All item responses are between -2 and 2. Zero indicates that both activities were judged equal on the trait.

Source
A manuscript fully describing the study is in preparation. Data are made available under the Community Data License Agreement - Sharing - Version 1.0

References

prepCleanData  Transforms data into a form tailored for efficient evaluation by Stan

Description
Vertex names, if not already factors, are converted to factors. The number of thresholds per item is determined by the largest absolute response value. Missing responses are filtered out. Responses on the same pair of vertices on the same item are grouped together. Within a vertex pair and item, responses are ordered from negative to positive.

Usage
prepCleanData(df)
**Arguments**

df  

a data frame with pairs of vertices given in columns pa1 and pa2, and item response data in other columns

**Details**

Note: Reordering of responses is likely unless something like `normalizeData` has been used with `.sortRows=TRUE`.

**Value**

a data list suitable for passing as the data argument to `pcStan` or `stan`

**See Also**

Other data preppers: `prepData()`, `prepFactorModel()`, `prepSingleFactorModel()`

**Examples**

df <- prepCleanData(phyActFlowPropensity)
str(df)

---

**Description**

Invokes `filterGraph` and `normalizeData`. Vertex names, if not already factors, are converted to factors. The number of thresholds per item is determined by the largest absolute response value. Missing responses are filtered out. Responses on the same pair of vertices on the same item are grouped together. Within a vertex pair and item, responses are ordered from negative to positive.

**Usage**

`prepData(df)`

**Arguments**

df  

a data frame with pairs of vertices given in columns pa1 and pa2, and item response data in other columns

**Value**

a data list suitable for passing as the data argument to `pcStan` or `stan`

**See Also**

Other data preppers: `prepCleanData()`, `prepFactorModel()`, `prepSingleFactorModel()`
**Examples**

```r
df <- prepData(phyActFlowPropensity)
str(df)
```

---

**prepFactorModel**  

type: function

**Specify a factor model**

**Description**

Specify a factor model with an arbitrary number of factors and arbitrary factor-to-item structure.

**Usage**

```r
prepFactorModel(data, path, factorScalePrior, psiScalePrior = NULL)
```

**Arguments**

- `data`: a data list prepared for processing by Stan
- `path`: a named list of item names
- `factorScalePrior`: a named numeric vector
- `psiScalePrior`: matrix of priors for factor correlations

**Details**

Both `factorScalePrior` and `psiScalePrior` are in the same units. A logistic transformation is applied to the signed proportion or correlation such that the parameter value becomes an unbounded real. The prior is a zero mean normal on this value with the given standard deviation.

For each factor, you need to specify its name, which items it predicts, and its scale prior. The connections from factors to items is specified by the ‘path’ argument. The scale priors are given in the ‘factorScalePrior’ argument. Both factors and items are specified by name (not index). The example shows how everything fits together. Paths are ordered as given in the ‘path’ argument.

The units of ‘factorScalePrior’ is a standard deviation of the normal prior for the logit transformed factor proportion.

**Value**

a data list suitable for passing as the data argument to `pcStan` or `stan`

**See Also**

To simulate data from a factor model: `generateFactorItems`

Other factor model: `prepSingleFactorModel()`

Other data preppers: `prepCleanData()`, `prepData()`, `prepSingleFactorModel()`
Examples

```r
pa <- phyActFlowPropensity[, setdiff(colnames(phyActFlowPropensity),
        c('goal1', 'feedback1'))]

dl <- prepData(pa)
psi <- diag(3)
psi[lower.tri(psi)] <- runif(3, 0, .8)
psi[upper.tri(psi)] <- t(psi)[upper.tri(psi)]
fname <- c('flow', 'f2', 'rc')
dimnames(psi) <- list(fname, fname)
dl <- prepFactorModel(dl,
    list(flow=c('complex', 'skill', 'predict',
               'creative', 'novelty', 'stakes',
               'present', 'reward', 'chatter',
               'body'),
         f2=c('waiting', 'control', 'evaluated', 'spont'),
         rc=c('novelty', 'waiting')),
    c(flow=0.9, f2=0.5, rc=0.2), psi)

str(dl)
```

---

**prepSingleFactorModel**  Specify a single factor model

**Description**

Specify a single latent factor with a path to each item.

**Usage**

```r
prepSingleFactorModel(data, factorScalePrior)
```

**Arguments**

- **data** a data list prepared for processing by Stan
- **factorScalePrior** standard deviation of the normal prior for the logit transformed factor proportion

**Value**

a data list suitable for passing as the data argument to `pcStan` or `stan`

**See Also**

Other factor model: `prepFactorModel()`
Other data preppers: `prepCleanData()`, `prepData()`, `prepFactorModel()`

**Examples**

```r
dl <- prepData(phyActFlowPropensity)
dl <- prepSingleFactorModel(dl, 0.9)
str(dl)
```
responseCurve

Produce data suitable for plotting item response curves

Description

Selects samples random draws from the posterior and evaluates the item response curve on the grid given by seq(from, to, by). All items use the same responseNames. If you have some items with a different number of thresholds or different response names then you can call responseCurve for each item separately and rbind the results together.

Usage

responseCurve(
  dl,  
  fit,  
  responseNames,  
  item = dl$nameInfo$item,  
  samples = 100,  
  from = -6,  
  to = -from,  
  by = 0.1
)

Arguments

dl a data list prepared by prepData
fit a stanfit object
responseNames a vector of labels for the possible responses
item a vector of item names
samples number of posterior samples
from the starting latent difference value
to the ending latent difference value
by the grid increment

Value

A data.frame with the following columns:

  response Which response
  worthDiff Difference in worth
  item Which item
  sample Which sample
  prob Associated probability
  responseSample A grouping index for independent item response samples
Response model

The paired comparison item response model has thresholds and a scale parameter similar to the partial credit model (Masters, 1982). The model is cumbersome to describe in traditional mathematical notation, but the R code is fairly straightforward,

```r
softmax <- function(y) exp(y) / sum(exp(y))

cmp_probs <- function(scale, alpha, pa1, pa2, thRaw) {
  th <- cumsum(thRaw)
  diff <- scale * (pa2 - pa1)
  unsummed <- c(0, diff + rev(th), diff - th, use.names = FALSE)
  softmax(cumsum(alpha * unsummed))
}
```

The function `cmp_probs` takes a scale constant, alpha discrimination, the latent scores for two objects `pa1` and `pa2`, and a vector of thresholds `thRaw`. The thresholds are parameterized as the difference from the previous threshold. For example, thresholds c(0.5, 0.5) are not at the same location but are at locations c(0.5, 1.0). Thresholds are symmetric. If there is one threshold then the model admits three possible response outcomes (e.g. win, tie, and lose). Responses are always stored centered with zero representing a tie. Therefore, it is necessary to add one plus the number of thresholds to response data to index into the vector returned by `cmp_probs`. For example, if our response data is (-1, 0, 1) and has one threshold then we would add 2 (1 + 1 threshold) to obtain the indices (1, 2, 3).

Use `itemModelExplorer` to explore the item model. In this shiny app, the discrimination parameter does what is customary in item response models. However, it is not difficult to show that discrimination is a function of thresholds and scale. That is, discrimination is not an independent parameter. In paired comparison models, discrimination and measurement error are confounded.

References


See Also

Other data extractor: `parDistributionCustom()`, `parInterval()`

Examples

vignette('manual', 'pcFactorStan')
roundRobinGraph  
Create an edge list with round-robin connectivity

**Description**
Create an edge list with round-robin connectivity

**Usage**
roundRobinGraph(name, N)

**Arguments**
- **name**: vector of vertex names
- **N**: number of comparisons

**Value**
An undirected graph represented as a data frame with each row describing an edge.

**See Also**
Other graph generators: twoLevelGraph()

**Examples**
roundRobinGraph(letters[1:5], 10)

toLoo  
Compute approximate leave-one-out (LOO) cross-validation for Bayesian models using Pareto smoothed importance sampling (PSIS)

**Description**
You must use an `_ll` model variation (see findModel).

**Usage**
toLoo(fit, ...)

**Arguments**
- **fit**: a stanfit object
- **...**: Additional options passed to `loo`. 
twoLevelGraph

Value

a loo object

See Also

outlierTable, loo

Examples

palist <- letters[1:10]
df <- twoLevelGraph(palist, 300)
theta <- rnorm(length(palist))
names(theta) <- palist
df <- generateItem(df, theta, th=rep(0.5, 4))

df <- filterGraph(df)
df <- normalizeData(df)
dl <- prepCleanData(df)
dl$scale <- 1.5

m1 <- pcStan("unidim.ll", dl)

loo1 <- toLoo(m1, cores=1)
print(loo1)

Description

Initially, edges are added from the first vertex to all the other vertices. Thereafter, the first vertex is drawn from a Beta(shape1, 1.0) distribution and the second vertex is drawn from a Beta(shape2, 1.0) distribution. The idea is that the edges will tend to connect a small subset of vertices from the top of the tree to leaf vertices. These vertex connections are similar to the pairs that you might observe in an elimination tournament. The selected vertices are sorted so it doesn’t matter whether shape1 > shape2 or shape1 < shape2.

Usage

twoLevelGraph(name, N, shape1 = 0.8, shape2 = 0.5)

Arguments

name vector of vertex names
N number of comparisons
shape1 beta distribution parameter for first edge
shape2 beta distribution parameter for second edge
Value
An undirected graph represented as a data frame with each row describing an edge.

See Also
Other graph generators: `roundRobinGraph()`

Examples
twoLevelGraph(letters[1:5], 20)

---

unfactor

Turn a factor back into a vector of integers

Description
Factors store values as integers and use a 'levels' attribute to map the integers to labels. This function removes the 'factor' class and levels attribute, leaving the vector of integers.

Usage
unfactor(f)

Arguments
f a factor

Examples
f <- factor(letters[1:3])
print(f)
print(unfactor(f))

---

withoutIndex

Remove the array indexing from a parameter name

Description
Remove the array indexing from a parameter name

Usage
withoutIndex(name)

Arguments
name a parameter name
withoutIndex

Value
the name without the square bracket parameter indexing

Examples
withoutIndex("foo[1,2]")
Index

*Topic datasets
   phyActFlowPropensity, 19

   calibrateItems, 3, 19
   check_hmc_diagnostics, 4
   filterGraph, 3, 4, 21
   findModel, 2, 5, 19, 26
   generateCovItems, 6, 10, 12, 13
   generateFactorItems, 7, 8, 9, 12, 13, 22
   generateItem, 7, 10, 10, 13
   generateSingleFactorItems, 7, 9, 10, 12, 12, 13
   itemModelExplorer, 7, 9, 11, 13, 14, 15, 25
   loo, 16, 26, 27
   normalizeData, 3, 15, 16, 21
   outlierTable, 16, 19, 27
   parDistributionCustom, 17, 18, 25
   parDistributionFor
      (parDistributionCustom), 17
   pareto_k_ids, 16
   parInterval, 17, 18, 25
   pcFactorStan (pcFactorStan-package), 2
   pcFactorStan-package, 2
   pcStan, 2, 6, 19, 21–23
   phyActFlowPropensity, 2, 19
   prepCleanData, 16, 20, 21–23
   prepData, 2, 14, 19, 21, 22–24
   prepFactorModel, 5, 10, 21, 22, 23
   prepSingleFactorModel, 5, 21, 22, 23
   print.stanfit, 19
   responseCurve, 17, 18, 24
   roundRobinGraph, 26, 28
   sampling, 19

stan, 2, 3, 19, 21–23
stanfit, 14, 17–19, 24, 26
stanmodel, 6
toLoo, 6, 16, 26
twoLevelGraph, 26, 27
unfactor, 28
withoutIndex, 28