Package 'grmtree'

January 14, 2026

Title Recursive Partitioning for Graded Response Models

Version 0.1.0

Date 2026-01-08

Maintainer Olayinka I. Arimoro <olayinka.arimoro@ucalgary.ca>

Description Provides methods for recursive partitioning based on the 'Graded Response Model' ('GRM'), extending the 'MOB' algorithm from the 'partykit' package. The package allows for fitting 'GRM' trees that partition the population into homogeneous subgroups based on item response patterns and covariates. Includes specialized plotting functions for visualizing 'GRM' trees with different terminal node displays (threshold regions, parameter profiles, and factor score distributions). For more details on the methods, see Samejima (1969) <doi:10.1002/J.2333-8504.1968.TB00153.X>, Komboz et al. (2018) <doi:10.1177/0013164416664394> and Arimoro et al. (2025) <doi:10.1007/s11136-025-04018-6>.

License GPL-3

Depends R (>= 4.1.0), partykit (>= 1.2-9), mirt (>= 1.36.1)

Imports stats, graphics, grid, dplyr, ggplot2, rlang, magrittr, strucchange

Suggests hlt, testthat (>= 3.0.0), knitr, rmarkdown, psychotools, psychotree, psych

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/Predicare1/grmtree

BugReports https://github.com/Predicare1/grmtree/issues

Repository CRAN

VignetteBuilder knitr

LazyData true

NeedsCompilation no

Author Olayinka I. Arimoro [aut, cre] (ORCID:
<https://orcid.org/0009-0009-9464-589X>),
Tolulope T. Sajobi [aut],
Lisa M. Lix [aut],
Matthew T. James [ctb],
Maria Santana [ctb],
Emmanuel Ugochukwu [ctb]

Date/Publication 2026-01-14 18:00:02 UTC

Contents

`discrpar_grmtree` *Extract Discrimination Parameters from GRM Tree*

Description

Extracts discrimination parameters (slope parameters) for each item from all terminal nodes of a graded response model tree. The discrimination parameter indicates how well an item distinguishes between respondents with different levels of the latent trait.

Usage

```
discrpar_grmtree(object, node = NULL, ...)
```

Arguments

`object`	A `grmtree` object.
`node`	Optional vector of node IDs to extract from. If NULL (default), extracts from all terminal nodes.
`...`	Additional arguments (currently unused).

Value

A data.frame with discrimination parameters for each item in each node, with columns:

`Node`	Node ID
`Item`	Item name
`Discrimination`	Discrimination parameter (a1)

See Also

`grmtree` fits a Graded Response Model Tree, `grmforest` for GRM Forests, `fscores_grmtree` for computing factor scores, `threshpar_grmtree` for extracting threshold parameters, `itempar_grmtree` for extracting item parameters

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit GRM tree with gender and group as partitioning variables
tree <- grmtree(resp ~ gender + group,
        data = asti,
        control = grmtree.control(minbucket = 30))

# Get all discrimination parameters
discr <- discrpar_grmtree(tree)
print(discr)
```

`fscores_grmtree` *Compute Latent Factor Scores for Each Terminal Node in a GRM Tree*

Description

This function calculates latent factor scores for each terminal node in a GRM tree object using specified scoring method (EAP, MAP, ML, or WLE).

Usage

```
fscores_grmtree(grmtree_obj, method = "EAP")
```

Arguments

`grmtree_obj`	A GRM tree object (from `grmtree()` function) containing fitted models in its terminal nodes.
`method`	Scoring method to use: "EAP" (default), "MAP", "ML", or "WLE". See `mirt::fscores()` for details.

Value

A named list where each element contains the factor scores for a terminal node. Names correspond to node IDs. Returns NULL for nodes where computation fails. If no scores can be computed for any node, returns NULL with a warning.

See Also

`fscores` for factor scoring methods, `grmtree` fits a Graded Response Model Tree, `grmforest` for GRM Forests, `threshpar_grmtree` for extracting threshold parameters, `discrpar_grmtree` for extracting discrimination parameters, `itempar_grmtree` for extracting item parameters, `generate_node_scores_dataset` generates combined dataset with node assignments and factor scores

Examples

```
  library(grmtree)
  library(hlt)
  data("asti", package = "hlt")
  asti$resp <- data.matrix(asti[, 1:4])

  # Fit GRM tree with gender and group as partitioning variables
  tree <- grmtree(resp ~ gender + group,
          data = asti,
          control = grmtree.control(minbucket = 30))

# Compute EAP scores for all terminal nodes
node_scores <- fscores_grmtree(tree)

# Compute MAP scores instead
node_scores_map <- fscores_grmtree(tree, method = "MAP")
```

---

generate_node_scores_dataset

Generate Combined Dataset with Node Assignments and Factor Scores

---

Description

Creates a dataset combining original data with node assignments and computed factor scores. Maintains original row order while adding node membership and factor score information.

Usage

```
generate_node_scores_dataset(grmtree_obj, method = "EAP")
```

Arguments

`grmtree_obj`	A GRM tree object (from `grmtree()` function).
`method`	Scoring method to use: "EAP" (default), "MAP", "ML", or "WLE".

Value

A data.frame containing: - Original variables from the model frame - 'node': Factor indicating terminal node membership (e.g., "Node 1") - 'factor_score': Computed latent factor scores Rows are in original order with sequential row names.

See Also

`grmtree` fits a Graded Response Model Tree, `grmforest` for GRM Forests, `fscores_grmtree` for computing factor scores, `threshpar_grmtree` for extracting threshold parameters, `discrpar_grmtree` for extracting discrimination parameters, `itempar_grmtree` for extracting item parameters

Examples

```
  library(grmtree)
  library(hlt)
  data("asti", package = "hlt")
  asti$resp <- data.matrix(asti[, 1:4])

  # Fit GRM tree with gender and group as partitioning variables
  tree <- grmtree(resp ~ gender + group,
          data = asti,
          control = grmtree.control(minbucket = 30))

# Generate combined dataset
scored_data <- generate_node_scores_dataset(tree)

# Plot scores by node
boxplot(factor_score ~ node, data = scored_data)
```

`grmforest` *Fit a Forest of Graded Response Model Trees for Ensemble-Based DIF Detection*

Description

This function implements a forest of graded response model trees (GRM Forest) using bootstrap aggregation (bagging) or random subsampling to enhance the detection and analysis of differential item functioning (DIF) in polytomous items. The GRM Forest approach combines the strengths of multiple GRMTrees to provide more robust and stable DIF detection, particularly for complex datasets with high-dimensional covariates or subtle DIF patterns.

Usage

```
grmforest(formula, data, control = grmforest.control(), ...)
```

Arguments

`formula`	A formula specifying the model structure with the response matrix on the left and partitioning variables on the right (e.g., `response_matrix ~ age + gender + education + clinical_variables`).
`data`	A data frame containing the response matrix and partitioning variables. The response matrix should contain polytomous items coded as ordered factors.
`control`	A control object created by `grmforest.control()`.
`...`	Additional arguments passed to underlying `grmtree()` function.

Details

The algorithm works by fitting multiple GRMTrees, each on a random sample of the original data (either through bootstrap sampling or subsampling). For each tree, approximately one-third of the observations are left out as out-of-bag (OOB) samples, which are used for internal validation and variable importance calculation. The ensemble approach reduces variance, minimizes overfitting, and provides more reliable identification of covariates associated with DIF.

Key advantages of the GRM Forest approach include:

- Enhanced stability in DIF detection across different sampling variations
- Robust variable importance measures that quantify the relative contribution of each covariate to DIF patterns
- Reduced false positive rates through consensus-based detection
- Ability to handle high-dimensional covariate spaces effectively
- Internal validation through out-of-bag error estimation

The forest implementation supports both bootstrap aggregation (where samples are drawn with replacement) and subsampling (without replacement), allowing flexibility for different data characteristics and research objectives.

Value

An object of class `grmforest` containing:

`trees`	List of fitted GRM trees
`oob_samples`	List of out-of-bag samples for each tree
`formula`	The model formula
`data`	The original dataset
`call`	The function call

See Also

`grmtree` fits a Graded Response Model Tree, `grmtree.control` creates a control object for `grmtree`, `grmforest.control` creates a control object for `grmforest`, `varimp` calculates the variable importance for GRM Forest, `plot.varimp` creates a bar plot of variable importance scores

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit forest with default parameters
forest <- grmforest(resp ~ gender + group, data = asti)

# Fit with custom control
ctrl <- grmforest.control(n_tree = 50, sampling = "subsample")
forest <- grmforest(resp ~ gender + group, data = asti, control = ctrl)
```

`grmforest.control` *Control Parameters for GRM Forest*

Description

Creates a control object for `grmforest` containing parameters that control the forest growing process including sampling, tree growing, and error handling.

Usage

```
grmforest.control(
  n_tree = 100,
  sampling = "bootstrap",
  sample_fraction = 0.632,
  mtry = NULL,
  remove_dead_trees = TRUE,
  control = grmtree.control(),
  alpha = 0.05,
  minbucket = 20,
  seed = NULL
)
```

Arguments

`n_tree`	Number of trees in the forest (default: 100).
`sampling`	Sampling method: "bootstrap" (with replacement) or "subsample" (without replacement) (default: "bootstrap").
`sample_fraction`	Fraction of data to sample for each tree (default: 0.632).
`mtry`	Number of variables randomly sampled as candidates at each split. If NULL, all variables are considered (default: NULL).

`remove_dead_trees`	Logical indicating whether to remove trees that encounter errors during fitting (default: TRUE).
`control`	Control parameters for individual trees created by `grmtree.control()`.
`alpha`	Significance level for splitting (default: 0.05).
`minbucket`	Minimum number of observations in terminal nodes (default: 20).
`seed`	Random seed for reproducibility (default: NULL).

Value

A list of class `grmforest_control` containing:

`n_tree`	Number of trees
`sampling`	Sampling method
`sample_fraction`	Sample fraction
`mtry`	Number of variables to try at each split
`remove_dead_trees`	Whether to remove failed trees
`control`	Tree control parameters
`seed`	Random seed

See Also

`grmtree.control` creates a control object for `grmtree`, `plot.grmtree` creates plot for the `grmtree` object, `grmforest` for GRM Forests,

Examples

```
library(grmtree)
# Control with 50 trees using subsampling
ctrl <- grmforest.control(n_tree = 50, sampling = "subsample")

# Control with specific tree parameters
ctrl <- grmforest.control(
  control = grmtree.control(minbucket = 30, alpha = 0.01)
)
```

grmtree — *Fit a Graded Response Model Tree for Differential Item Functioning Detection*

Description

This function implements a tree-based graded response model (GRM) using model-based recursive partitioning to detect and account for differential item functioning (DIF) in polytomous items. The GRMTree combines the statistical framework of item response theory with recursive partitioning to identify heterogeneous subgroups in the population where item parameters (discrimination and thresholds) vary systematically across covariates.

Usage

```
grmtree(
  formula,
  data,
  na.action = na.omit,
  control = grmtree.control(),
  mtry = NULL,
  ...
)
```

Arguments

`formula`	A formula specifying the model structure with the response matrix on the left and partitioning variables on the right (e.g., `response_matrix ~ age + gender`).
`data`	A data frame containing the variables in the model.
`na.action`	How to handle missing values (default: `na.omit`).
`control`	A list of control parameters created by `grmtree.control()`.
`mtry`	Number of variables randomly sampled as candidates at each split. If NULL, all variables are considered.
`...`	Additional arguments passed to the fitting function.

Details

The algorithm works by first estimating a global GRM for the entire sample, then recursively testing for parameter instability with respect to available covariates. When significant DIF is detected, the sample is partitioned into homogeneous subgroups, each with their own set of item parameters. This approach allows for the identification of complex interaction effects and provides interpretable tree structures that visualize how item functioning varies across different patient subgroups.

GRMTree is particularly useful in health outcomes research where patient-reported outcome measures may function differently across diverse demographic, clinical, or socioeconomic subgroups. The resulting tree diagrams facilitate the development of personalized assessment strategies and can inform targeted interventions by identifying specific patient characteristics associated with differential item interpretation.

Conventional Graded Response Model (GRM):

Let Y_{im} denote the response of the i^{th} ($i = 1, \ldots, N$) individual to the m^{th} ($m = 1, 2, \ldots, M$) item. The graded response model is described as:

$$P(Y_{im} \geq j | \tau_{mj}, \lambda_m, \theta_i) = \frac{\exp(-(\tau_{mj} - \lambda_m \theta_i))}{1 + \exp(-(\tau_{mj} - \lambda_m \theta_i))}$$

where:

- $P(Y_{im} \geq j | \tau_{mj}, \lambda_m, \theta_i)$ is the probability that individual i's response is in category j or higher on item m,
- τ_{mj} is the threshold parameter between categories $j - 1$ and j for item m,
- λ_m is the discrimination parameter for item m,
- $\theta_i \sim N(0, 1)$ is the latent trait score for individual i.

This parametrization is equivalent to the conventional IRT formulation where item discrimination is $a_m = \lambda_m$ and item difficulty is $b_{mj} = \tau_{mj}/\lambda_m$.

Graded Response Model Tree (GRMTree) Implementation:

The GRMTree is a hybrid model that integrates the GRM with model-based recursive partitioning to detect and account for differential item functioning (DIF) across subgroups defined by covariates. The algorithm proceeds through the following steps:

Step 1: Global Model Estimation

Estimate the GRM item parameters $(\hat{\tau}_{mj}, \hat{\lambda}_m)$ jointly for all individuals in the study cohort at the root node via maximum likelihood estimation:

$$\hat{\beta}_{\text{global}} = \arg\max_{\beta} \sum_{i=1}^{N} \log L(\beta; \mathbf{y}_i)$$

where $\beta = (a_1, \ldots, a_J, b_{11}, \ldots, b_{J,m-1})$ contains all item parameters (discrimination and difficulty), providing a baseline model assuming parameter invariance.

Step 2: Parameter Stability Testing

For each available covariate X_p ($p = 1, \ldots, P$), assess the stability of the item parameters by conducting score-based structural change tests. This involves: 1. Calculating the score function contributions $s(\hat{\beta}; y_i, x_i)$ for each individual, 2. Ordering these scores with respect to each covariate X_p, 3. Testing the null hypothesis $H_0 : \mathbb{E}[s(\hat{\beta}; y_i, x_i)] = 0$ for all i against the alternative that scores fluctuate systematically with X_p, indicating parameter instability (DIF).

Step 3: Recursive Partitioning

If significant instability is detected ($p < \alpha_{\text{adj}}$):

- **Covariate Selection:** Identify the covariate X_p^* with the most significant instability (smallest adjusted p-value),
- **Split Point Determination:** Find the optimal cut-point c^* that maximizes the partitioned log-likelihood:

  $$\ell_{\text{left}}(\beta) + \ell_{\text{right}}(\beta) = \sum_{i:X_{pi} \leq c^*} \log L(\beta; y_i) + \sum_{i:X_{pi} > c^*} \log L(\beta; y_i)$$

  over all possible cut-points on X_p^*,

- **Sample Splitting:** Partition the sample into two child nodes based on the rule $X_p^* \leq c^*$.

Step 4: Recursive Application & Stopping Criteria

Repeat Steps 1-3 recursively within each resulting child node until one of the following stopping criteria is met:

1. **No Significant Instability:** No covariate shows significant parameter instability after multiple testing correction ($\alpha_{\text{adj}} = \alpha/m$, where multiple adjustment methods can be applied, including Bonferroni, Holm, Benjamini-Hochberg, etc.).
2. **Minimum Node Size:** The subsample size falls below a prespecified minimum (e.g., $n < 10\times$ the number of item parameters).

Formal GRMTree Structure:

The final GRMTree provides a piecewise GRM where each terminal node represents a subgroup with homogeneous item parameters, explicitly modeling the detected DIF structure within the data. The resulting GRMTree model can be expressed as a mixture of subgroup-specific GRMs:

$$P(Y_{ij} = k|\theta_i, \mathbf{x}_i) = \sum_{b=1}^{B} I(\mathbf{x}_i \in \mathcal{X}_b) \cdot P_b(Y_{ij} = k|\theta_i)$$

where:

- B is the number of terminal nodes,
- $\mathcal{X}_b$ is the covariate subspace defining terminal node b,
- P_b is the node-specific GRM with parameters β_b,
- $I(\cdot)$ is the indicator function.

Each terminal node b contains a complete GRM with:

- **Node-specific item parameters:** $\beta_b = (a_{1b}, \ldots, a_{Jb}, b_{11b}, \ldots, b_{J,m-1,b})$
- **Local ability distribution:** $\theta_i|\mathbf{x}_i \in \mathcal{X}_b \sim N(0, 1)$

This approach allows **differential item functioning (DIF)** to be detected and modeled explicitly through the tree structure, where item parameters can vary across subgroups defined by covariates, while maintaining the conditional distribution of the latent trait within each subgroup.

Post-Hoc Multiple Comparison Adjustments:

For holm, BH, BY, hochberg, and hommel methods, the algorithm employs a two-stage approach: (1) build a tree using `initial_alpha` as the splitting threshold, (2) apply global p-value adjustment across all splits, and (3) prune splits that do not meet the adjusted threshold `alpha`. This differs from Bonferroni correction, which is applied locally during tree construction via `partykit::mob`. The choice of `initial_alpha` represents a statistical trade-off. The default (`min(3 * alpha, 0.20)`) provides a balance between power and computational efficiency. Note that global post-hoc adjustments in hierarchical tree structures may be conservative compared to per-node adjustments, as they account for all tests performed during tree exploration. This global adjustment approach controls the tree-wide error rate and may be conservative (Type I error often <3%). This conservativeness ensures strong control of family-wise error rate or false discovery rate across all splits in the tree. Users requiring less conservative control may prefer the Bonferroni method, which applies per-node adjustment during tree construction.

Value

An object of class `grmtree` inheriting from `modelparty` containing the fitted tree structure.

Author(s)

Olayinka Imisioluwa Arimoro <`olayinka.arimoro@ucalgary.ca`>, Lisa M. Lix, Tolulope T. Sajobi

References

Methodological Foundations:

Samejima, F. (1969). Estimation of latent ability using a response pattern of graded scores. Psychometrika Monograph Supplement, 34, 100-114.

Strobl, C., Kopf, J., & Zeileis, A. (2015). Rasch trees: A new method for detecting differential item functioning in the Rasch model. Psychometrika, 80(2), 289-316.

Komboz, B., Strobl, C., & Zeileis, A. (2018). Tree-based global model tests for polytomous Rasch models. Educational and psychological measurement, 78(1), 128-166. https://doi.org/10.1177/0013164416664394

Arimoro, O. I., Lix, L. M., Patten, S. B., Sawatzky, R., Sebille, V., Liu, J., Wiebe, S., Josephson, C. B., & Sajobi, T. T. (2025). Tree-based latent variable model for assessing differential item functioning in patient-reported outcome measures: a simulation study. Quality of Life Research. https://doi.org/10.1007/s11136-025-04018-6

Applied Examples:

Arimoro, O. I., Josephson, C. B., James, M. T., Patten, S. B., Wiebe, S., Lix, L. M., & Sajobi, T. T. (2024). Screening for depression in patients with epilepsy: same questions but different meaning to different patients. Quality of Life Research, 33(12), 3409-3419. https://doi.org/10.1007/s11136-024-03782-1

See Also

`print.grmtree` prints the detailed summary results of the `grmtree` object, `grmtree.control` creates a control object for `grmtree`, `plot.grmtree` creates plot for the `grmtree` object, `grmforest` for GRM Forests, `varimp` calculates the variable importance for GRM Forest, `fscores_grmtree` for computing factor scores, `threshpar_grmtree` for extracting threshold parameters, `discrpar_grmtree` for extracting discrimination parameters, `itempar_grmtree` for extracting item parameters

Examples

```
library(grmtree)
library(hlt)

# Prepare the asti data (from the hlt package)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit GRM tree with gender and group as partitioning variables
tree <- grmtree(resp ~ gender + group,
        data = asti,
        control = grmtree.control(minbucket = 30))

## Print and plot the tree
print(tree)
plot(tree)
```

```
# Extract item parameters for specific subgroups
discr_params <- discrpar_grmtree(tree)
threshold_params <- threshpar_grmtree(tree)
```

`grmtree.control` *Control Parameters for GRM Trees*

Description

Creates a control object for `grmtree` containing various parameters that control the tree growing process.

Usage

```
grmtree.control(
  minbucket = 20,
  p_adjust = "none",
  alpha = 0.05,
  initial_alpha = NULL,
  ...
)
```

Arguments

`minbucket`	Minimum number of observations in a terminal node (default: 20).
`p_adjust`	Method for p-value adjustment. One of: "none", "bonferroni", "holm", "BH", "BY", "hochberg", or "hommel" (default: "none").
`alpha`	Significance level for splitting (default: 0.05).
`initial_alpha`	For post-hoc adjustment methods (holm, BH, BY, hochberg, hommel), the significance threshold for initial tree construction before pruning. Must satisfy `alpha` < `initial_alpha` < `1`. Default is `min(3 * alpha, 0.20)`. Lower values produce more conservative results but run faster; higher values provide more power but require more computation and may increase Type I error. Ignored for "none" and "bonferroni" methods.
`...`	Additional arguments passed to `partykit::mob_control()`.

Value

A list of control parameters with class `grmtree_control`.

See Also

`grmtree` fits a Graded Response Model Tree

Examples

```
# Use Bonferroni correction with alpha = 0.01
ctrl <- grmtree.control(p_adjust = "bonferroni", alpha = 0.01)
```

`grmtree_data` *Medical Outcomes Study Social Support Survey (MOS-SS) Test Data*

Description

A dataset containing sample responses to the MOS-SS emotional domain items and demographic variables. This data is provided for testing and demonstration purposes within the grmtree package. The items are numbered 1-5, representing None of the time, A little of the time, Some of the time, Most of the time, All of the time, respectively.

Usage

```
grmtree_data
```

Format

A tibble with 3,500 rows and 17 variables:

MOS_Listen Someone you can count on to listen to you when you need to talk (1-5 Likert scale)

MOS_Info Someone to give you information to help you understand a situation (1-5 Likert scale)

MOS_Advice_Crisis Someone to give good advice about a crisis (1-5 Likert scale)

MOS_Confide Someone to confide in or talk to about yourself or your problems (1-5 Likert scale)

MOS_Advice_Want Someone whose advice you really want (1-5 Likert scale)

MOS_Fears Someone to share private worries or fears (1-5 Likert scale)

MOS_Personal Someone to turn to for suggestions about how to deal with a personal problem (1-5 Likert scale)

MOS_Understand Someone who understands your problems (1-5 Likert scale)

sex Gender (Male, Female)

age Age in years (numeric)

residency Residence location (rural, urban)

depressed Depression status (No, Yes)

bmi Body Mass Index (numeric)

Education Education level (Primary/High school, College/University)

job Employment status (Employed, Unemployed)

smoker Smoking status (No, Yes)

multimorbidity Number of chronic conditions (0, 1, 2+)

Source

Simulated data generated for package testing and demonstration

Examples

```
library(dplyr)

# Load and take a glimpse at the data
data(grmtree_data, package = "grmtree")
glimpse(grmtree_data)
```

`itempar_grmtree` *Extract Item Parameters from GRM Tree*

Description

Extracts both discrimination parameters and average threshold parameters for each item from all terminal nodes of a graded response model tree. This provides a comprehensive view of item characteristics across different nodes of the tree.

Usage

```
itempar_grmtree(object, node = NULL, ...)
```

Arguments

`object`	A `grmtree` object.
`node`	Optional vector of node IDs to extract from. If NULL (default), extracts from all terminal nodes.
`...`	Additional arguments (currently unused).

Value

A data.frame with item parameters for each item in each node, with columns:

`Node`	Node ID
`Item`	Item name
`Discrimination`	Discrimination parameter (a1)
`AvgThreshold`	Average of threshold parameters
`Thresholds`	All threshold parameters as a list column

See Also

`grmtree` fits a Graded Response Model Tree, `grmforest` for GRM Forests, `fscores_grmtree` for computing factor scores, `threshpar_grmtree` for extracting threshold parameters, `discrpar_grmtree` for extracting discrimination parameters

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit GRM tree with gender and group as partitioning variables
tree <- grmtree(resp ~ gender + group,
        data = asti,
        control = grmtree.control(minbucket = 30))

# Get all item parameters
items <- itempar_grmtree(tree)
print(items)
```

`plot.grmtree` *Plot Method for GRM Tree Objects*

Description

Visualizes a GRM (Graded Response Model) tree with different types of terminal node plots. This function extends `plot.modelparty` from the partykit package with specialized visualizations for GRM trees.

Usage

```
## S3 method for class 'grmtree'
plot(
  x,
  type = c("regions", "profile", "histogram"),
  what = c("item", "threshold", "discrimination"),
  tnex = 2L,
  drop_terminal = TRUE,
  spacing = 0.1,
  ...
)
```

Arguments

`x`	A GRM tree object of class 'grmtree'.
`type`	Type of terminal node plot to display: **"regions"** Threshold regions plot (default) **"profile"** Item parameter profile plot **"histogram"** Histogram of factor scores with normal curve
`what`	Type of parameters to plot when `type = "profile"`:

	"item" Both discrimination and threshold parameters (default)
	"threshold" Only threshold parameters
	"discrimination" Only discrimination parameters
`tnex`	Numeric scaling factor for terminal node extension (default: 2).
`drop_terminal`	Logical indicating whether to drop terminal node IDs (default: TRUE).
`spacing`	Numeric value controlling spacing between elements (default: 0.1).
`...`	Additional arguments passed to the terminal panel functions.

Details

The function provides three visualization types:

- **Regions plot**: Shows threshold parameters as colored regions, useful for visualizing the difficulty parameters across items and nodes.
- **Profile plot**: Displays either item parameters (discrimination and average thresholds), just thresholds, or just discrimination parameters as line plots across items.
- **Histogram**: Shows the distribution of factor scores in each node with an overlaid normal curve.

Value

Invisibly returns the GRM tree object. Primarily called for its side effect of producing a plot.

See Also

`plot.modelparty` for the underlying plotting infrastructure, `grmtree` for creating GRM tree objects, `plot.varimp` creates a bar plot of variable importance scores

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit GRM tree with gender and group as partitioning variables
tree <- grmtree(resp ~ gender + group,
          data = asti,
          control = grmtree.control(minbucket = 30))

# Default regions plot
plot(tree)

# Profile plot showing item parameters
plot(tree, type = "profile")

# Profile plot showing only thresholds
plot(tree, type = "profile", what = "threshold")
```

```
# Histograms of factor scores
plot(tree, type = "histogram")
```

plot.varimp *Plot Variable Importance*

Description

Creates a bar plot of variable importance scores with options for both ggplot2 and base R graphics.

Usage

```
## S3 method for class 'varimp'
plot(x, top_n = NULL, use_ggplot = TRUE, ...)
```

Arguments

x	A varimp object from varimp().
top_n	Number of top variables to display (NULL for all).
use_ggplot	Logical indicating whether to use ggplot2 (if available).
...	Additional arguments passed to plotting functions.

Value

Invisibly returns the input object.

See Also

varimp calculates the variable importance for GRM Forest, grmforest for GRM Forests, grmforest.control creates a control object for grmforest, plot.grmtree creates plot for the grmtree object

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit forest with default parameters
forest <- grmforest(resp ~ gender + group, data = asti)
imp <- varimp(forest)
plot(imp)
plot(imp, top_n = 1) ## select top 1 importance variable
plot(imp, use_ggplot = FALSE) # Use base R graphics
```

`print.grmforest` *Print Method for GRM Forest*

Description

Print Method for GRM Forest

Usage

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

Arguments

`x`	A `grmforest` object
`...`	Additional arguments (currently unused)

Value

Invisibly returns the input object

`print.grmtree` *Print Method for GRM Tree Objects*

Description

Displays a formatted summary of a GRM (Graded Response Model) tree object. This function extends `print.modelparty` from the partykit package with specialized formatting for GRM trees.

Usage

```
## S3 method for class 'grmtree'
print(
  x,
  title = "Graded Response Model Tree",
  objfun = "negative log-likelihood",
  ...
)
```

Arguments

`x`	A GRM tree object of class ’grmtree’.
`title`	Character string specifying the title for the print output (default: "Graded Response Model Tree").
`objfun`	Character string labeling the objective function (default: "negative log-likelihood").
`...`	Additional arguments passed to `print.modelparty`.

Details

The print method provides a comprehensive summary of the GRM tree, including:

- Model formula used for fitting
- Tree structure with node information
- Item parameter estimates for each terminal node
- Confidence intervals for parameters
- Group parameters (mean and covariance)
- Summary statistics (number of nodes, objective function value)

Value

Invisibly returns the GRM tree object. Primarily called for its side effect of printing a formatted summary.

See Also

`print.modelparty` for the underlying printing infrastructure, `grmtree` for creating GRM tree objects

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit GRM tree
tree <- grmtree(resp ~ gender + group,
                data = asti,
                control = grmtree.control(minbucket = 30))

# Print the tree summary
print(tree)

# Alternative syntax (automatically calls print.grmtree)
tree
```

`threshpar_grmtree` *Extract Threshold Parameters from GRM Tree*

Description

Extracts threshold parameters for each item from all terminal nodes of a graded response model tree. The thresholds represent the points on the latent trait continuum where the probability of scoring in adjacent response categories is equal.

Usage

```
threshpar_grmtree(object, node = NULL, ...)
```

Arguments

`object`	A `grmtree` object.
`node`	Optional vector of node IDs to extract from. If NULL (default), extracts from all terminal nodes.
`...`	Additional arguments (currently unused).

Value

A data.frame with threshold parameters for each item in each node, with columns:

`Node`	Node ID
`Item`	Item name
`d1`, `d2`, `...`	Threshold parameters for each category

See Also

`grmtree` fits a Graded Response Model Tree, `grmforest` for GRM Forests, `fscores_grmtree` for computing factor scores, `discrpar_grmtree` for extracting discrimination parameters, `itempar_grmtree` for extracting item parameters

Examples

```
library(grmtree)
library(hlt)

data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

# Fit GRM tree with gender and group as partitioning variables
tree <- grmtree(resp ~ gender + group,
        data = asti,
        control = grmtree.control(minbucket = 30))

# Get all thresholds
thresholds <- threshpar_grmtree(tree)
print(thresholds)
```

varimp — *Calculate Variable Importance for GRM Forest*

Description

Computes permutation importance scores for variables in a GRM forest using out-of-bag samples. Importance is measured by the decrease in log-likelihood when a variable's values are permuted.

Usage

```
varimp(forest, method = "permutation", verbose = FALSE, seed = NULL)
```

Arguments

`forest`	A `grmforest` object created by `grmforest()`.
`method`	Importance calculation method (currently only "permutation").
`verbose`	Logical indicating whether to show progress messages.
`seed`	Random seed for reproducibility.

Value

A named numeric vector of importance scores with class `varimp`. Higher values indicate more important variables.

See Also

`grmtree` fits a Graded Response Model Tree, `grmforest` for GRM Forests, `grmforest.control` creates a control object for `grmforest`, `plot.varimp` creates a bar plot of variable importance scores

Examples

```
library(grmtree)
library(hlt)
data("asti", package = "hlt")
asti$resp <- data.matrix(asti[, 1:4])

## Fit the GRM Forest
forest <- grmforest(resp ~ gender + group, data = asti,
control = grmforest.control(n_tree = 5))
  importance <- varimp(forest)

## Print and plot the variable importance scores
print(importance)
plot(importance)
```

Index