Package 'spatialRF'

December 19, 2025

Title Easy Spatial Modeling with Random Forest

Version 1.1.5

URL https://blasbenito.github.io/spatialRF/

BugReports https://github.com/BlasBenito/spatialRF/issues/

Description Automatic generation and selection of spatial predictors for Random Forest models fitted to spatially structured data. Spatial predictors are constructed from a distance matrix among training samples using Moran's Eigenvector Maps (MEMs; Dray, Legendre, and Peres-Neto 2006 <DOI:10.1016/j.ecolmodel.2006.02.015>) or the RFsp approach (Hengl et al. <DOI:10.7717/peerj.5518>). These predictors are used alongside user-supplied explanatory variables in Random Forest models. The package provides functions for model fitting, multicollinearity reduction, interaction identification, hyperparameter tuning, evaluation via spatial cross-validation, and result visualization using partial dependence and interaction plots. Model fitting relies on the 'ranger' package (Wright and Ziegler 2017 <DOI:10.18637/jss.v077.i01>).

License MIT + file LICENSE

Depends R (>= 2.10)

Imports dplyr, ggplot2, magrittr, stats, tibble, utils, foreach, doParallel, ranger, rlang, tidyr, tidyselect, huxtable (>= 5.8.0), patchwork (>= 1.3.2), viridis

Suggests testthat, spelling

Encoding UTF-8

LazyData true

LazyDataCompression xz

Roxygen list(markdown = TRUE, old_usage = FALSE)

RoxygenNote 7.3.3

Language en-US

NeedsCompilation no

Author Blas M. Benito [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-5105-7232>)

Maintainer Blas M. Benito <blasbenito@gmail.com>

Contents

`.vif_to_df` *Convert VIF values to data frame*

Description

Computes variance inflation factors for all variables in a data frame and returns them in a tidy format, sorted by VIF in descending order.

Usage

```
.vif_to_df(x)
```

Arguments

`x`	Data frame with numeric predictors for which to compute VIF values.

Value

Data frame with two columns: `variable` (character, variable names) and `vif` (numeric, VIF scores), sorted by VIF in descending order.

See Also

Other utilities: `auc()`, `beowulf_cluster()`, `objects_size()`, `optimization_function()`, `prepare_importance_spatia`
`rescale_vector()`, `root_mean_squared_error()`, `setup_parallel_execution()`, `standard_error()`,
`statistical_mode()`, `thinning()`, `thinning_til_n()`

---

auc *Area under the ROC curve*

Description

Computes the area under the ROC curve (AUC) for binary classification.

Usage

```
auc(o, p)
```

Arguments

o	Numeric vector of actual binary labels (0 or 1). Must have the same length as p.
p	Numeric vector of predicted probabilities (typically 0 to 1). Must have the same length as o.

Value

Numeric value between 0 and 1 representing the AUC. Higher values indicate better classification performance, with 0.5 indicating random performance and 1.0 indicating perfect classification.

See Also

Other utilities: `.vif_to_df()`, `beowulf_cluster()`, `objects_size()`, `optimization_function()`,
`prepare_importance_spatial()`, `rescale_vector()`, `root_mean_squared_error()`, `setup_parallel_execution()`,
`standard_error()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

Examples

```
auc(
  o = c(0, 0, 1, 1),
  p = c(0.1, 0.6, 0.4, 0.8)
  )
```

`auto_cor` *Multicollinearity reduction via Pearson correlation*

Description

Filters predictors using sequential evaluation of pairwise correlations. Predictors are ranked by user preference (or column order) and evaluated sequentially. Each candidate is added to the selected pool only if its maximum absolute correlation with already-selected predictors does not exceed the threshold.

Usage

```
auto_cor(
  x = NULL,
  preference.order = NULL,
  cor.threshold = 0.5,
  verbose = TRUE
)
```

Arguments

`x`	Data frame with predictors, or a `variable_selection` object from `auto_vif()`. Default: `NULL`.
`preference.order`	Character vector specifying variable preference order. Does not need to include all variables in `x`. If `NULL`, column order is used. Default: `NULL`.
`cor.threshold`	Numeric between 0 and 1 (recommended: 0.5 to 0.9). Maximum allowed absolute Pearson correlation between selected variables. Default: `0.50`
`verbose`	Logical. If `TRUE`, prints messages about operations and removed variables. Default: `TRUE`

Details

The algorithm follows these steps:

1. Rank predictors by `preference.order` (or use column order if NULL).
2. Initialize selection pool with first predictor.
3. For each remaining candidate:
   - Compute maximum absolute correlation with selected predictors.
   - If max correlation equal or lower than `cor.threshold`, add to selected pool.
   - Otherwise, skip candidate.
4. Return selected predictors.

Data cleaning: Variables in `preference.order` not found in `colnames(x)` are silently removed. Non-numeric columns are removed with a warning. Rows with NA values are removed via `na.omit()`. Zero-variance columns trigger a warning but are not removed.

This function can be chained with `auto_vif()` through pipes (see examples).

Value

List with class `variable_selection` containing:

- `cor`: Correlation matrix of selected variables (only if 2+ variables selected).
- `selected.variables`: Character vector of selected variable names.
- `selected.variables.df`: Data frame containing selected variables.

See Also

`auto_vif()`

Other preprocessing: `auto_vif()`, `case_weights()`, `default_distance_thresholds()`, `double_center_distance_mat`
`is_binary()`, `make_spatial_fold()`, `make_spatial_folds()`, `the_feature_engineer()`, `weights_from_distance_ma`

Examples

```
data(
  plants_df,
  plants_predictors
)

y <- auto_cor(
  x = plants_df[, plants_predictors]
)

y$selected.variables
y$cor
head(y$selected.variables.df)
```

---

auto_vif *Multicollinearity reduction via Variance Inflation Factor*

Description

Filters predictors using sequential evaluation of variance inflation factors. Predictors are ranked by user preference (or column order) and evaluated sequentially. Each candidate is added to the selected pool only if the maximum VIF of all predictors (candidate plus already-selected) does not exceed the threshold.

Usage

```
auto_vif(x = NULL, preference.order = NULL, vif.threshold = 5, verbose = TRUE)
```

Arguments

x	Data frame with predictors, or a variable_selection object from auto_cor(). Default: NULL.
preference.order	Character vector specifying variable preference order. Does not need to include all variables in x. If NULL, column order is used. Default: NULL.
vif.threshold	Numeric (recommended: 2.5 to 10). Maximum allowed VIF among selected variables. Higher values allow more collinearity. Default: 5.
verbose	Logical. If TRUE, prints messages about operations and removed variables. Default: TRUE

Details

The algorithm follows these steps:

1. Rank predictors by preference.order (or use column order if NULL).
2. Initialize selection pool with first predictor.
3. For each remaining candidate:
   - Compute VIF for candidate plus all selected predictors.
   - If max VIF equal or lower than vif.threshold, add candidate to selected pool.
   - Otherwise, skip candidate.
4. Return selected predictors with their VIF values.

Data cleaning: Variables in preference.order not found in colnames(x) are silently removed. Non-numeric columns are removed with a warning. Rows with NA values are removed via na.omit(). Zero-variance columns trigger a warning but are not removed.

This function can be chained with auto_cor() through pipes (see examples).

Value

List with class variable_selection containing:

- vif: Data frame with selected variable names and their VIF scores.
- selected.variables: Character vector of selected variable names.
- selected.variables.df: Data frame containing selected variables.

See Also

auto_cor()

Other preprocessing: auto_cor(), case_weights(), default_distance_thresholds(), double_center_distance_mat
is_binary(), make_spatial_fold(), make_spatial_folds(), the_feature_engineer(), weights_from_distance_ma

Examples

```
data(
  plants_df,
  plants_predictors
)

y <- auto_vif(
  x = plants_df[, plants_predictors]
)

y$selected.variables
y$vif
head(y$selected.variables.df)
```

`beowulf_cluster` *Create a Beowulf cluster for parallel computing*

Description

Creates a Beowulf cluster configuration from machine IPs, core counts, and user credentials.

Usage

```
beowulf_cluster(
  cluster.ips = NULL,
  cluster.cores = NULL,
  cluster.user = Sys.info()[["user"]],
  cluster.port = "11000",
  outfile = NULL
)
```

Arguments

`cluster.ips`	Character vector of machine IP addresses in the cluster. The first IP is the main node (typically the machine running this code). Default: `NULL`.
`cluster.cores`	Integer vector of core counts for each machine. Must match the length of `cluster.ips`. Default: `NULL`.
`cluster.user`	Character string for the user name across all machines. Default: current system user.
`cluster.port`	Character string specifying the communication port. Default: `"11000"`.
`outfile`	Character string or `NULL`. Path to append worker messages, `""` to print to console, or `NULL` (default) for `/dev/null` (Linux) or `nul:` (Windows).

Details

Network requirements: Firewalls on all machines must allow traffic on the specified port.

Usage workflow:

1. Create cluster with this function
2. Register with `doParallel::registerDoParallel()`
3. Run parallelized code (e.g., foreach loops)
4. Stop cluster with `parallel::stopCluster()`

Value

Cluster object created by `parallel::makeCluster()`, ready for registration with `doParallel::registerDoParallel()`.

See Also

Other utilities: `.vif_to_df()`, `auc()`, `objects_size()`, `optimization_function()`, `prepare_importance_spatial()`, `rescale_vector()`, `root_mean_squared_error()`, `setup_parallel_execution()`, `standard_error()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

Examples

```
## Not run:
# Create cluster with 3 machines
beowulf.cluster <- beowulf_cluster(
  cluster.ips = c(
    "192.168.1.10",  # main node
    "192.168.1.11",
    "192.168.1.12"
  ),
  cluster.cores = c(7, 4, 4),
  cluster.user = "username",
  cluster.port = "11000"
)

# Register cluster for parallel processing
doParallel::registerDoParallel(cl = beowulf.cluster)

# Run parallelized code (e.g., foreach loop)
# your_parallel_code_here

# Stop cluster when done
parallel::stopCluster(cl = beowulf.cluster)

## End(Not run)
```

case_weights *Generate case weights for imbalanced binary data*

Description

Generates case weights to balance binary response variables for use with ranger models. Used internally by rf().

Usage

```
case_weights(data = NULL, dependent.variable.name = NULL)
```

Arguments

data	Data frame containing the response variable. Default: NULL.
dependent.variable.name	Character string specifying the response variable name. Must be a column in data. Default: NULL.

Details

The weighting scheme assigns higher weights to the minority class to balance training:

- Cases with value 0: weight = 1 / n_zeros
- Cases with value 1: weight = 1 / n_ones

This ensures both classes contribute equally to model training regardless of class imbalance.

Value

Numeric vector of length nrow(data) with case weights. Each weight is the inverse of the class frequency: 1/n_zeros for 0s and 1/n_ones for 1s.

See Also

Other preprocessing: auto_cor(), auto_vif(), default_distance_thresholds(), double_center_distance_matrix()
is_binary(), make_spatial_fold(), make_spatial_folds(), the_feature_engineer(), weights_from_distance_ma

Examples

```
# Imbalanced dataset: 3 zeros, 2 ones
weights <- case_weights(
  data = data.frame(
    response = c(0, 0, 0, 1, 1)
  ),
  dependent.variable.name = "response"
)

weights
```

```
# Returns: 0.333, 0.333, 0.333, 0.5, 0.5
# Zeros get weight 1/3, ones get weight 1/2
```

`default_distance_thresholds` *Default distance thresholds for spatial predictors*

Description

Generates four evenly-spaced distance thresholds for spatial predictor generation, ranging from 0 to half the maximum distance in the matrix.

Usage

```
default_distance_thresholds(distance.matrix = NULL)
```

Arguments

`distance.matrix` Numeric distance matrix (typically square and symmetric). Default: `NULL`.

Details

The maximum threshold is set to half the maximum distance to avoid spatial predictors based on distances that are too large to capture meaningful spatial autocorrelation. The four thresholds are evenly spaced using `seq()` with `length.out = 4`.

Value

Numeric vector of length 4 with distance thresholds (floored to integers).

See Also

Other preprocessing: `auto_cor()`, `auto_vif()`, `case_weights()`, `double_center_distance_matrix()`, `is_binary()`, `make_spatial_fold()`, `make_spatial_folds()`, `the_feature_engineer()`, `weights_from_distance_ma`

Examples

```
data(plants_distance)

thresholds <- default_distance_thresholds(
  distance.matrix = plants_distance
)

thresholds
# Example output: c(0, 3333, 6666, 10000)
# Four evenly-spaced thresholds from 0 to max(plants_distance)/2
```

`double_center_distance_matrix` *Double-center a distance matrix*

Description

Double-centers a distance matrix by converting it to weights and centering to zero row and column means. Required for computing Moran's Eigenvector Maps.

Usage

```
double_center_distance_matrix(distance.matrix = NULL, distance.threshold = 0)
```

Arguments

`distance.matrix`	Numeric distance matrix. Default: `NULL`.
`distance.threshold`	Numeric distance threshold for weight calculation. Distances above this threshold are set to 0 during weighting. Default: `0`.

Details

Double-centering is performed in two steps:

1. Convert distances to weights using `weights_from_distance_matrix()`
2. Center the matrix: subtract row means, subtract column means, and add the grand mean

The resulting matrix is symmetric with zero row and column means, suitable for Moran's Eigenvector Maps computation.

Value

Double-centered numeric matrix with the same dimensions as `distance.matrix`. The matrix has row means and column means of zero.

See Also

`weights_from_distance_matrix()`, `mem()`, `mem_multithreshold()`

Other preprocessing: `auto_cor()`, `auto_vif()`, `case_weights()`, `default_distance_thresholds()`, `is_binary()`, `make_spatial_fold()`, `make_spatial_folds()`, `the_feature_engineer()`, `weights_from_distance_ma`

Examples

```
data(plants_distance)

# Double-center the distance matrix
centered <- double_center_distance_matrix(
  distance.matrix = plants_distance
)

# Verify row means are zero
head(rowMeans(centered))

# Verify column means are zero
head(colMeans(centered))
```

`filter_spatial_predictors`

Remove redundant spatial predictors

Description

Removes spatial predictors that are highly correlated with other spatial predictors or with non-spatial predictors. Particularly useful when using multiple distance thresholds that produce correlated spatial predictors.

Usage

```
filter_spatial_predictors(
  data = NULL,
  predictor.variable.names = NULL,
  spatial.predictors.df = NULL,
  cor.threshold = 0.5
)
```

Arguments

`data`	Data frame containing the predictor variables. Default: `NULL`.
`predictor.variable.names`	Character vector of non-spatial predictor names. Must match column names in `data`. Can also be a `variable_selection` object. Default: `NULL`.
`spatial.predictors.df`	Data frame of spatial predictors (e.g., from `mem_multithreshold()`). Default: `NULL`.
`cor.threshold`	Numeric between 0 and 1 (recommended: 0.5 to 0.75). Maximum allowed absolute Pearson correlation. Default: `0.50`.

Details

Filtering is performed in two steps:

1. Remove spatial predictors correlated with each other (using `auto_cor()`)
2. Remove spatial predictors correlated with non-spatial predictors

This two-step process ensures the retained spatial predictors are independent of both each other and the environmental predictors, improving model interpretability and reducing multicollinearity.

Value

Data frame containing only spatial predictors with correlations below `cor.threshold` (both among themselves and with non-spatial predictors).

See Also

Other spatial_analysis: `mem()`, `mem_multithreshold()`, `moran()`, `moran_multithreshold()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(
  plants_df,
  plants_predictors,
  plants_distance
)

# Generate spatial predictors using multiple distance thresholds
mem.df <- mem_multithreshold(
  distance.matrix = plants_distance,
  distance.thresholds = c(0, 1000)
)

# Filter spatial predictors to remove redundancy
# Removes spatial predictors correlated > 0.50 with each other
# or with environmental predictors
spatial.predictors.filtered <- filter_spatial_predictors(
  data = plants_df,
  predictor.variable.names = plants_predictors,
  spatial.predictors.df = mem.df,
  cor.threshold = 0.50
)

# Check dimensions
ncol(mem.df)  # original number
ncol(spatial.predictors.filtered)  # after filtering
```

get_evaluation *Extract evaluation metrics from cross-validated model*

Description

Extracts aggregated performance metrics from a model evaluated with rf_evaluate().

Usage

```
get_evaluation(model)
```

Arguments

model	Model object with class rf_evaluate from rf_evaluate().

Details

This function returns aggregated statistics across all cross-validation repetitions. The "Testing" model metrics indicate the model's ability to generalize to unseen spatial locations.

Value

Data frame with aggregated evaluation metrics containing:

- model: Model type - "Full" (original model), "Training" (trained on training folds), or "Testing" (performance on testing folds, representing generalization ability).
- metric: Metric name - "rmse", "nrmse", "r.squared", or "pseudo.r.squared".
- mean, sd, min, max: Summary statistics across cross-validation repetitions.

See Also

rf_evaluate(), plot_evaluation(), print_evaluation()

Other model_info: get_importance(), get_importance_local(), get_moran(), get_performance(), get_predictions(), get_residuals(), get_response_curves(), get_spatial_predictors(), print.rf(), print_evaluation(), print_importance(), print_moran(), print_performance()

Examples

```
if(interactive()){

data(plants_rf, plants_xy)

# Evaluate model with spatial cross-validation
m_evaluated <- rf_evaluate(
  model = plants_rf,
  xy = plants_xy,
  repetitions = 5,
  n.cores = 1
```

```
)

# Extract evaluation metrics
eval_metrics <- get_evaluation(m_evaluated)
eval_metrics

}
```

`get_importance` *Extract variable importance from model*

Description

Extracts variable importance scores from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_importance(model)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.

Details

For spatial models (`rf_spatial()`) with many spatial predictors, this function returns aggregated importance statistics for spatial predictors to improve readability. Non-spatial models return per-variable importance scores directly.

Value

Data frame with columns `variable` (character) and `importance` (numeric), sorted by decreasing importance.

See Also

`rf()`, `rf_repeat()`, `rf_spatial()`, `plot_importance()`, `print_importance()`

Other model_info: `get_evaluation()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract variable importance
importance <- get_importance(plants_rf)
head(importance)

# View top 5 most important variables
importance[1:5, ]
```

`get_importance_local` *Extract local variable importance from model*

Description

Extracts local (case-specific) variable importance scores from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_importance_local(model)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.

Details

Local importance measures how much each predictor contributes to predictions for individual observations, unlike global importance which summarizes contributions across all observations. This can reveal spatial or contextual patterns in variable influence.

Value

Data frame with one row per observation and one column per predictor variable. Each cell contains the local importance score for that variable at that observation.

See Also

`rf()`, `rf_repeat()`, `rf_spatial()`, `get_importance()`, `plot_importance()`, `print_importance()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract local importance scores
local_imp <- get_importance_local(plants_rf)

# View structure: rows = observations, columns = variables
dim(local_imp)
head(local_imp)

# Find which variable is most important for first observation
colnames(local_imp)[which.max(local_imp[1, ])]
```

---

get_moran *Extract Moran's I test results for model residuals*

Description

Extracts Moran's I test results for spatial autocorrelation in model residuals from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_moran(model)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.

Details

Moran's I tests for spatial autocorrelation in model residuals. Significant positive values indicate residuals are spatially clustered, suggesting the model hasn't fully captured spatial patterns. For spatial models (`rf_spatial()`), low or non-significant Moran's I values indicate successful removal of spatial autocorrelation.

Value

Data frame with Moran's I statistics at multiple distance thresholds. Columns include `distance.threshold`, `moran.i` (statistic), `p.value`, `interpretation`, and `method`.

See Also

`moran()`, `moran_multithreshold()`, `plot_moran()`, `print_moran()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract Moran's I test results
moran_results <- get_moran(plants_rf)
moran_results

# Check for significant spatial autocorrelation
significant <- moran_results[moran_results$p.value < 0.05, ]
significant
```

`get_performance` *Extract out-of-bag performance metrics from model*

Description

Extracts out-of-bag (OOB) performance metrics from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_performance(model)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.

Details

Out-of-bag (OOB) performance is computed using observations not included in bootstrap samples during model training. Metrics typically include R-squared, pseudo R-squared, RMSE, and normalized RMSE. For repeated models, the median and median absolute deviation summarize performance across repetitions.

Value

Data frame with performance metrics:

- For `rf()` and `rf_spatial()`: columns `metric` and `value`
- For `rf_repeat()`: columns `metric`, `median`, and `median_absolute_deviation` (MAD across repetitions)

See Also

`rf()`, `rf_repeat()`, `rf_spatial()`, `print_performance()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract OOB performance metrics
performance <- get_performance(plants_rf)
performance

# For repeated models, median and MAD are returned
# (example would require rf_repeat model)
```

`get_predictions` *Extract fitted predictions from model*

Description

Extracts fitted (in-sample) predictions from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_predictions(model)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.

Details

This function returns fitted predictions for the training data used to build the model, not predictions for new data. For out-of-sample predictions on new data use `stats::predict()`.

Value

Numeric vector of fitted predictions with length equal to the number of training observations. For `rf_repeat()` models, returns the median prediction across repetitions.

See Also

`rf()`, `rf_repeat()`, `rf_spatial()`, `get_residuals()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract fitted predictions
predictions <- get_predictions(plants_rf)
head(predictions)

# Check length matches number of observations
length(predictions)

# Compare with observed values to assess fit
# (observed values would be in original data)
```

`get_residuals` *Extract model residuals*

Description

Extracts residuals (observed - predicted values) from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_residuals(model)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.

Details

Residuals are calculated as observed minus predicted values. They can be used to assess model fit, check assumptions, and diagnose patterns such as spatial autocorrelation (see `get_moran()`). Ideally, residuals should be randomly distributed with no systematic patterns.

Value

Numeric vector of residuals with length equal to the number of training observations. For `rf_repeat()` models, returns the median residual across repetitions.

See Also

`rf()`, `rf_repeat()`, `rf_spatial()`, `get_predictions()`, `get_moran()`, `plot_residuals_diagnostics()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract residuals
residuals <- get_residuals(plants_rf)
head(residuals)

# Check basic statistics
summary(residuals)

# Plot distribution to check for patterns
hist(residuals, main = "Residual Distribution", xlab = "Residuals")
```

---

get_response_curves *Extract response curve data for plotting*

Description

Extracts data for plotting partial dependence (response) curves showing how predictions vary with each predictor from models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

Usage

```
get_response_curves(
  model = NULL,
  variables = NULL,
  quantiles = c(0.1, 0.5, 0.9),
  grid.resolution = 200,
  verbose = TRUE
)
```

Arguments

`model`	Model object from `rf()`, `rf_repeat()`, or `rf_spatial()`.
`variables`	Character vector of predictor names to plot. If `NULL`, automatically selects the top 50% most important variables. Default: `NULL`.
`quantiles`	Numeric vector of quantiles (0 to 1) at which to fix non-plotted predictors. Multiple quantiles show response variation under different scenarios. Default: `c(0.1, 0.5, 0.9)`.
`grid.resolution`	Integer (20 to 500) specifying the number of points along the predictor axis. Higher values produce smoother curves. Default: `200`.
`verbose`	Logical. If `TRUE`, prints progress messages. Default: `TRUE`.

Details

Response curves (also called partial dependence plots) show how predicted values change as a focal predictor varies while holding other predictors constant at specified quantile values. This reveals the marginal effect of each predictor.

The function generates curves by:

1. Creating a grid of values for the focal predictor
2. Fixing non-plotted predictors at each quantile (e.g., 0.1, 0.5, 0.9)
3. Predicting responses across the grid
4. Repeating for each selected predictor and quantile combination

Multiple quantiles reveal whether the effect of a predictor is consistent across different environmental contexts (parallel curves) or varies depending on other conditions (non-parallel curves).

Value

Data frame with the following columns:

- `response`: Predicted response values.
- `predictor`: Predictor values along the gradient.
- `quantile`: Factor indicating which quantile was used to fix other predictors.
- `model`: Model index (only for `rf_repeat()` models with multiple repetitions).
- `predictor.name`: Character name of the focal predictor.
- `response.name`: Character name of the response variable.

See Also

`rf()`, `rf_repeat()`, `rf_spatial()`, `plot_response_curves()`, `get_importance()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_spatial_predictors()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

# Extract response curve data for plotting
curves <- get_response_curves(
  model = plants_rf,
  variables = NULL,  # auto-select important variables
  quantiles = c(0.1, 0.5, 0.9)
)

# View structure
head(curves)
str(curves)

# Check unique predictors included
```

```
unique(curves$predictor.name)
```

`get_spatial_predictors` *Extract spatial predictors from spatial model*

Description

Extracts the spatial predictors (Moran's Eigenvector Maps) used in a model fitted with `rf_spatial()`.

Usage

```
get_spatial_predictors(model)
```

Arguments

`model`	Model object from `rf_spatial()` (must have class `rf_spatial`).

Details

Spatial predictors are Moran's Eigenvector Maps (MEMs) automatically generated and selected by `rf_spatial()` to capture spatial autocorrelation patterns in the data. This function extracts these predictors, which can be useful for understanding spatial structure or for making predictions on new spatial locations.

Value

Data frame containing the spatial predictor values for each observation, with predictors ordered by decreasing importance.

See Also

`rf_spatial()`, `mem()`, `mem_multithreshold()`, `get_importance()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `print.rf()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf_spatial)

# Extract spatial predictors
spatial_preds <- get_spatial_predictors(plants_rf_spatial)
head(spatial_preds)

# Check dimensions
dim(spatial_preds)
```

```
# View predictor names (ordered by importance)
colnames(spatial_preds)
```

`is_binary` *Check if variable is binary with values 0 and 1*

Description

Tests whether a variable contains only the values 0 and 1.

Usage

```
is_binary(data = NULL, dependent.variable.name = NULL)
```

Arguments

`data`	Data frame containing the variable to check.
`dependent.variable.name`	Character string with the name of the variable to test. Must be a column name in `data`.

Details

This function is used internally by spatialRF to determine whether to apply classification-specific methods (e.g., case weighting with `case_weights()`). The function returns `FALSE` if:

- The variable has more than two unique values
- The variable has only one unique value (constant)
- The unique values are not exactly 0 and 1 (e.g., 1 and 2, or TRUE and FALSE)

Missing values (NA) are ignored when determining unique values.

Value

Logical. `TRUE` if the variable contains exactly two unique values (0 and 1), `FALSE` otherwise.

See Also

`case_weights()`

Other preprocessing: `auto_cor()`, `auto_vif()`, `case_weights()`, `default_distance_thresholds()`, `double_center_distance_matrix()`, `make_spatial_fold()`, `make_spatial_folds()`, `the_feature_engineer()`, `weights_from_distance_matrix()`

Examples

```
# Binary variable (returns TRUE)
is_binary(
  data = data.frame(response = c(0, 0, 0, 1, 1)),
  dependent.variable.name = "response"
)

# Non-binary variable (returns FALSE)
is_binary(
  data = data.frame(response = c(1, 2, 3, 4, 5)),
  dependent.variable.name = "response"
)

# Binary but wrong values (returns FALSE)
is_binary(
  data = data.frame(response = c(1, 1, 2, 2)),
  dependent.variable.name = "response"
)
```

---

make_spatial_fold *Create spatially independent training and testing folds*

Description

Generates two spatially independent data folds by growing a rectangular buffer from a focal point until a specified fraction of records falls inside. Used internally by `make_spatial_folds()` and `rf_evaluate()` for spatial cross-validation.

Usage

```
make_spatial_fold(
  data = NULL,
  dependent.variable.name = NULL,
  xy.i = NULL,
  xy = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
  training.fraction = 0.8
)
```

Arguments

`data`	Data frame containing response variable and predictors. Required only for binary response variables.
`dependent.variable.name`	Character string with the name of the response variable. Must be a column name in `data`. Required only for binary response variables.

`xy.i` Single-row data frame with columns "x" (longitude), "y" (latitude), and "id" (record identifier). Defines the focal point from which the buffer grows.

`xy` Data frame with columns "x" (longitude), "y" (latitude), and "id" (record identifier). Contains all spatial coordinates for the dataset.

`distance.step.x` Numeric value specifying the buffer growth increment along the x-axis. Default: `NULL` (automatically set to 1/1000th of the x-coordinate range).

`distance.step.y` Numeric value specifying the buffer growth increment along the y-axis. Default: `NULL` (automatically set to 1/1000th of the y-coordinate range).

`training.fraction` Numeric value between 0.1 and 0.9 specifying the fraction of records to include in the training fold. Default: `0.8`.

Details

This function creates spatially independent training and testing folds for spatial cross-validation. The algorithm works as follows:

1. Starts with a small rectangular buffer centered on the focal point (`xy.i`)
2. Grows the buffer incrementally by `distance.step.x` and `distance.step.y`
3. Continues growing until the buffer contains the desired number of records (`training.fraction * total records`)
4. Assigns records inside the buffer to training and records outside to testing

Special handling for binary response variables:

When `data` and `dependent.variable.name` are provided and the response is binary (0/1), the function ensures that `training.fraction` applies to the number of presences (1s), not total records. This prevents imbalanced sampling in presence-absence models.

Value

List with two elements:

- `training`: Integer vector of record IDs (from `xy$id`) in the training fold.
- `testing`: Integer vector of record IDs (from `xy$id`) in the testing fold.

See Also

`make_spatial_folds()`, `rf_evaluate()`, `is_binary()`

Other preprocessing: `auto_cor()`, `auto_vif()`, `case_weights()`, `default_distance_thresholds()`, `double_center_distance_matrix()`, `is_binary()`, `make_spatial_folds()`, `the_feature_engineer()`, `weights_from_distance_matrix()`

Examples

```
data(plants_df, plants_xy)

# Create spatial fold centered on first coordinate
fold <- make_spatial_fold(
  xy.i = plants_xy[1, ],
  xy = plants_xy,
  training.fraction = 0.6
)

# View training and testing record IDs
fold$training
fold$testing

# Visualize the spatial split (training = red, testing = blue, center = black)
if (interactive()) {
  plot(plants_xy[c("x", "y")], type = "n", xlab = "", ylab = "")
  points(plants_xy[fold$training, c("x", "y")], col = "red4", pch = 15)
  points(plants_xy[fold$testing, c("x", "y")], col = "blue4", pch = 15)
  points(plants_xy[1, c("x", "y")], col = "black", pch = 15, cex = 2)
}
```

---

make_spatial_folds *Create multiple spatially independent training and testing folds*

Description

Applies `make_spatial_fold()` to every row in `xy.selected`, generating one spatially independent fold centered on each focal point. Used for spatial cross-validation in `rf_evaluate()`.

Usage

```
make_spatial_folds(
  data = NULL,
  dependent.variable.name = NULL,
  xy.selected = NULL,
  xy = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
  training.fraction = 0.75,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`data`	Data frame containing response variable and predictors. Required only for binary response variables.

`dependent.variable.name`
: Character string with the name of the response variable. Must be a column name in `data`. Required only for binary response variables.

`xy.selected`
: Data frame with columns "x" (longitude), "y" (latitude), and "id" (record identifier). Defines the focal points for fold creation. Typically a spatially thinned subset of `xy` created with `thinning()` or `thinning_til_n()`.

`xy`
: Data frame with columns "x" (longitude), "y" (latitude), and "id" (record identifier). Contains all spatial coordinates for the dataset.

`distance.step.x`
: Numeric value specifying the buffer growth increment along the x-axis. Default: `NULL` (automatically set to 1/1000th of the x-coordinate range).

`distance.step.y`
: Numeric value specifying the buffer growth increment along the y-axis. Default: `NULL` (automatically set to 1/1000th of the y-coordinate range).

`training.fraction`
: Numeric value between 0.1 and 0.9 specifying the fraction of records to include in the training fold. Default: `0.75`.

`n.cores`
: Integer specifying the number of CPU cores for parallel execution. Default: `parallel::detectCores() - 1`.

`cluster`
: Optional cluster object created with `parallel::makeCluster()`. If provided, overrides `n.cores`. User is responsible for stopping the cluster with `parallel::stopCluster()`. Default: `NULL`.

Details

This function creates multiple spatially independent folds for spatial cross-validation by calling `make_spatial_fold()` once for each row in `xy.selected`. Each fold is created by growing a rectangular buffer from the corresponding focal point until the desired `training.fraction` is achieved.

Parallel execution:

The function uses parallel processing to speed up fold creation. You can control parallelization with `n.cores` or provide a pre-configured cluster object.

Typical workflow:

1. Thin spatial points with `thinning()` or `thinning_til_n()` to create `xy.selected`
2. Create spatial folds with this function
3. Use the folds for spatial cross-validation in `rf_evaluate()`

Value

List where each element corresponds to a row in `xy.selected` and contains:

- `training`: Integer vector of record IDs (from `xy$id`) in the training fold.
- `testing`: Integer vector of record IDs (from `xy$id`) in the testing fold.

See Also

make_spatial_fold(), rf_evaluate(), thinning(), thinning_til_n()

Other preprocessing: auto_cor(), auto_vif(), case_weights(), default_distance_thresholds(), double_center_distance_matrix(), is_binary(), make_spatial_fold(), the_feature_engineer(), weights_from_distance_matrix()

Examples

```
data(plants_df, plants_xy)

# Thin to 10 focal points to speed up example
xy.thin <- thinning_til_n(
  xy = plants_xy,
  n = 10
)

# Create spatial folds centered on the 10 thinned points
folds <- make_spatial_folds(
  xy.selected = xy.thin,
  xy = plants_xy,
  distance.step.x = 0.05,
  training.fraction = 0.6,
  n.cores = 1
)

# Each element is a fold with training and testing indices
length(folds)  # 10 folds
names(folds[[1]])  # "training" and "testing"

# Visualize first fold (training = red, testing = blue, center = black)
if (interactive()) {
  plot(plants_xy[c("x", "y")], type = "n", xlab = "", ylab = "")
  points(plants_xy[folds[[1]]$training, c("x", "y")], col = "red4", pch = 15)
  points(plants_xy[folds[[1]]$testing, c("x", "y")], col = "blue4", pch = 15)
  points(
    plants_xy[folds[[1]]$training[1], c("x", "y")],
    col = "black",
    pch = 15,
    cex = 2
  )
}
```

---

mem *Compute Moran's Eigenvector Maps from distance matrix*

Description

Computes Moran's Eigenvector Maps (MEMs) from a distance matrix. Returns only eigenvectors with positive spatial autocorrelation, which capture broad to medium-scale spatial patterns.

Usage

```
mem(distance.matrix = NULL, distance.threshold = 0, colnames.prefix = "mem")
```

Arguments

`distance.matrix`
: Numeric distance matrix between spatial locations.

`distance.threshold`
: Numeric value specifying the maximum distance for spatial neighbors. Distances above this threshold are set to zero. Default: `0` (no thresholding).

`colnames.prefix`
: Character string used as prefix for column names in the output. Default: `"mem"`.

Details

Moran's Eigenvector Maps (MEMs) are spatial variables that represent spatial structures at different scales. The function creates MEMs through the following steps:

1. Double-centers the distance matrix using `double_center_distance_matrix()`
2. Computes eigenvectors and eigenvalues using `base::eigen()`
3. Normalizes eigenvalues by dividing by the maximum absolute eigenvalue
4. Selects only eigenvectors with positive normalized eigenvalues

Positive vs. negative eigenvalues:

Eigenvectors with positive eigenvalues represent positive spatial autocorrelation (nearby locations are similar), capturing broad to medium-scale spatial patterns. Eigenvectors with negative eigenvalues represent negative spatial autocorrelation (nearby locations are dissimilar) and are excluded. The returned MEMs are ordered by eigenvalue magnitude, with the first columns capturing the broadest spatial patterns.

These MEMs are used as spatial predictors in `rf_spatial()` to account for spatial autocorrelation in model residuals.

Value

Data frame where each column is a MEM (spatial predictor) representing a different scale of spatial pattern. Columns are named with the pattern `<prefix>_<number>` (e.g., "mem_1", "mem_2").

See Also

`mem_multithreshold()`, `rf_spatial()`, `double_center_distance_matrix()`

Other spatial_analysis: `filter_spatial_predictors()`, `mem_multithreshold()`, `moran()`, `moran_multithreshold()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_distance)

# Compute MEMs from distance matrix
mems <- mem(distance.matrix = plants_distance)

# View structure
head(mems)
dim(mems)

# Check column names
colnames(mems)[1:5]
```

`mem_multithreshold` *Compute Moran's Eigenvector Maps across multiple distance thresholds*

Description

Computes Moran's Eigenvector Maps (MEMs) using `mem()` at multiple distance thresholds and combines them into a single data frame. This creates spatial predictors capturing patterns at different spatial scales.

Usage

```
mem_multithreshold(
  distance.matrix = NULL,
  distance.thresholds = NULL,
  max.spatial.predictors = NULL
)
```

Arguments

`distance.matrix`
: Numeric distance matrix between spatial locations.

`distance.thresholds`
: Numeric vector of distance thresholds. Each threshold defines the maximum distance for spatial neighbors at that scale. Default: `NULL` (automatically computed with `default_distance_thresholds()`).

`max.spatial.predictors`
: Integer specifying the maximum number of spatial predictors to return. If the total number of MEMs exceeds this value, only the first `max.spatial.predictors` columns are returned. Default: `NULL` (no limit).

Details

This function generates spatial predictors at multiple spatial scales by computing MEMs at different distance thresholds. Different thresholds capture spatial patterns at different scales:

- Smaller thresholds (e.g., 0) capture fine-scale spatial patterns
- Larger thresholds capture broad-scale spatial patterns

Algorithm:

1. For each distance threshold, calls `mem()` to compute MEMs
2. Each `mem()` call applies the threshold, double-centers the matrix, and extracts positive eigenvectors
3. Combines all MEMs into a single data frame
4. Optionally limits the total number of predictors with `max.spatial.predictors`

The resulting MEMs are used as spatial predictors in `rf_spatial()` to model spatial autocorrelation at multiple scales simultaneously.

Value

Data frame with one row per observation (matching `distance.matrix` dimensions) and columns representing MEMs at different distance thresholds. Column names follow the pattern `spatial_predictor_<threshold>_<` (e.g., "spatial_predictor_0_1", "spatial_predictor_1000_2").

See Also

`mem()`, `rf_spatial()`, `default_distance_thresholds()`, `double_center_distance_matrix()`

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `moran()`, `moran_multithreshold()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_distance)

# Compute MEMs for multiple distance thresholds
mems <- mem_multithreshold(
  distance.matrix = plants_distance,
  distance.thresholds = c(0, 1000, 5000)
)

# View structure
head(mems)
dim(mems)

# Check column names showing threshold and predictor number
colnames(mems)[1:6]

# Limit number of spatial predictors
mems_limited <- mem_multithreshold(
```

```
  distance.matrix = plants_distance,
  distance.thresholds = c(0, 1000, 5000),
  max.spatial.predictors = 20
)
dim(mems_limited)
```

`moran` *Moran's I test for spatial autocorrelation*

Description

Computes Moran's I, a measure of spatial autocorrelation that tests whether values are more similar (positive autocorrelation) or dissimilar (negative autocorrelation) among spatial neighbors than expected by chance.

Usage

```
moran(
  x = NULL,
  distance.matrix = NULL,
  distance.threshold = NULL,
  verbose = TRUE
)
```

Arguments

`x`	Numeric vector to test for spatial autocorrelation. Typically model residuals or a response variable.
`distance.matrix`	Numeric distance matrix between observations. Must have the same number of rows as the length of `x`.
`distance.threshold`	Numeric value specifying the maximum distance for spatial neighbors. Distances above this threshold are set to zero during weighting. Default: `NULL` (automatically set to `0`, meaning no thresholding).
`verbose`	Logical. If `TRUE`, displays a Moran's scatterplot. Default: `TRUE`.

Details

Moran's I is a measure of spatial autocorrelation that quantifies the degree to which nearby observations have similar values. The statistic ranges approximately from -1 to +1:

- **Positive values**: Similar values cluster together (positive spatial autocorrelation)
- **Values near zero**: Random spatial pattern (no spatial autocorrelation)
- **Negative values**: Dissimilar values are adjacent (negative spatial autocorrelation, rare in practice)

Statistical testing:

The function compares the observed Moran's I to the expected value under the null hypothesis of no spatial autocorrelation (EI = -1/(n-1)). The p-value is computed using a normal approximation. Results are interpreted at 0.05 significance level.

Moran's scatterplot:

The plot shows original values (x-axis) against spatially lagged values (y-axis). The slope of the fitted line approximates Moran's I. Points in quadrants I and III indicate positive spatial autocorrelation; points in quadrants II and IV indicate negative spatial autocorrelation.

This implementation is inspired by the `Moran.I()` function in the ape package.

Value

List of class "moran" with three elements:

- `test`: Data frame containing:
  - `distance.threshold`: The distance threshold used
  - `moran.i.null`: Expected Moran's I under null hypothesis of no spatial autocorrelation
  - `moran.i`: Observed Moran's I statistic
  - `p.value`: Two-tailed p-value from normal approximation
  - `interpretation`: Text interpretation of the result
- `plot`: ggplot object showing Moran's scatterplot (values vs. spatial lag values with linear fit).
- `plot.df`: Data frame with columns `x` (original values) and `x.lag` (spatially lagged values) used to generate the plot.

See Also

`moran_multithreshold()`, `get_moran()`

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `mem_multithreshold()`, `moran_multithreshold()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_df, plants_distance, plants_response)

# Test for spatial autocorrelation in response variable
moran_test <- moran(
  x = plants_df[[plants_response]],
  distance.matrix = plants_distance,
  distance.threshold = 1000
)

# View test results
moran_test$test

# Access components
moran_test$test$moran.i  # Observed Moran's I
moran_test$test$p.value  # P-value
```

```
moran_test$test$interpretation  # Text interpretation
```

`moran_multithreshold` *Moran's I test across multiple distance thresholds*

Description

Computes Moran's I at multiple distance thresholds to assess spatial autocorrelation across different neighborhood scales. Identifies the distance threshold with the strongest spatial autocorrelation.

Usage

```
moran_multithreshold(
  x = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  verbose = TRUE
)
```

Arguments

`x`	Numeric vector to test for spatial autocorrelation. Typically model residuals or a response variable.
`distance.matrix`	Numeric distance matrix between observations. Must have the same number of rows as the length of `x`.
`distance.thresholds`	Numeric vector of distance thresholds defining different neighborhood scales. Each threshold specifies the maximum distance for spatial neighbors at that scale. Default: `NULL` (automatically computed with `default_distance_thresholds()`).
`verbose`	Logical. If `TRUE`, displays a plot of Moran's I values across distance thresholds. Default: `TRUE`.

Details

This function applies `moran()` at multiple distance thresholds to explore spatial autocorrelation at different spatial scales. This multi-scale approach is valuable for several reasons:

- **Scale exploration**: Different processes may operate at different spatial scales. Testing multiple thresholds reveals the scale(s) at which spatial autocorrelation is strongest.
- **Optimal neighborhood definition**: Identifies the distance threshold that best captures the spatial structure in the data.
- **Uncertainty assessment**: Spatial neighborhoods are often uncertain in ecological and spatial data. Testing multiple thresholds accounts for this uncertainty.

Interpreting results:

The plot shows Moran's I values across distance thresholds. Peaks in Moran's I indicate spatial scales where autocorrelation is strongest. The `max.moran` and `max.moran.distance.threshold` values identify the optimal scale. Significant results (p equal or lower than 0.05) indicate spatial autocorrelation at that particular scale.

This function is commonly used to:

1. Detect spatial autocorrelation in model residuals at multiple scales
2. Determine appropriate distance thresholds for generating spatial predictors with `mem_multithreshold()`
3. Assess whether spatial patterns vary across scales

Value

List with four elements:

- `per.distance`: Data frame with one row per distance threshold, containing columns:
  - `distance.threshold`: Distance threshold used
  - `moran.i`: Observed Moran's I statistic
  - `moran.i.null`: Expected Moran's I under null hypothesis
  - `p.value`: Two-tailed p-value
  - `interpretation`: Text interpretation of the result
- `plot`: ggplot object showing how Moran's I varies across distance thresholds, highlighting significant results.
- `max.moran`: Numeric value of the maximum Moran's I observed across all thresholds.
- `max.moran.distance.threshold`: Distance threshold (in distance matrix units) where Moran's I is maximized.

See Also

`moran()`, `mem_multithreshold()`, `default_distance_thresholds()`, `get_moran()`

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `mem_multithreshold()`, `moran()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_df, plants_distance, plants_response)

# Test spatial autocorrelation at multiple distance thresholds
moran_multi <- moran_multithreshold(
  x = plants_df[[plants_response]],
  distance.matrix = plants_distance,
  distance.thresholds = c(0, 1000, 5000)
)

# View results for all thresholds
moran_multi$per.distance
```

```
# Find optimal distance threshold
moran_multi$max.moran.distance.threshold
moran_multi$max.moran

# Plot shows spatial autocorrelation across scales
moran_multi$plot
```

`objects_size` *Display sizes of objects in current R environment*

Description

Returns a summary of objects in the current R workspace, sorted from largest to smallest by memory size. Useful for identifying memory-intensive objects and diagnosing memory issues.

Usage

```
objects_size(n = 10)
```

Arguments

`n`	Integer specifying the number of largest objects to display. Default: `10`.

Details

This utility function helps monitor memory usage by displaying the largest objects in your workspace. It's particularly useful for:

- Identifying memory bottlenecks during large spatial analyses
- Deciding which objects to remove to free memory
- Understanding the memory footprint of different data structures

The function examines all objects in the global environment (`.GlobalEnv`) and calculates their memory usage using `utils::object.size()`. Objects are automatically sorted by size in descending order.

Value

Data frame with object names as row names and four columns:

- `Type`: Object class (e.g., "data.frame", "matrix", "list").
- `Size`: Memory size with automatic unit formatting (e.g., "1.2 Mb", "500 bytes").
- `Length/Rows`: Number of elements (for vectors) or rows (for data frames/matrices).
- `Columns`: Number of columns (for data frames/matrices; `NA` for vectors and other objects).

See Also

utils::object.size(), base::ls(), base::rm()

Other utilities: .vif_to_df(), auc(), beowulf_cluster(), optimization_function(), prepare_importance_spatial(
rescale_vector(), root_mean_squared_error(), setup_parallel_execution(), standard_error(),
statistical_mode(), thinning(), thinning_til_n()

Examples

```
# Create some objects of different sizes
small_vector <- runif(100)
medium_matrix <- matrix(runif(10000), 100, 100)
large_matrix <- matrix(runif(100000), 1000, 100)

# View the 5 largest objects
objects_size(n = 5)

# Check all objects (up to 10 by default)
objects_size()
```

---

optimization_function *Compute optimization scores for spatial predictor selection*

Description

Computes optimization scores for candidate spatial predictor sets using either the "moran.i" or "p.value" method. Higher scores indicate better trade-offs between spatial autocorrelation reduction, model performance, and parsimony.

Usage

```
optimization_function(
  x = NULL,
  weight.r.squared = NULL,
  weight.penalization.n.predictors = NULL,
  optimization.method = "moran.i"
)
```

Arguments

x	Data frame containing optimization metrics for candidate spatial predictor sets. Generated internally by select_spatial_predictors_sequential() or select_spatial_predictor Must include columns: moran.i, r.squared, penalization.per.variable, and p.value.binary (for "p.value" method).
weight.r.squared	Numeric value between 0 and 1 specifying the weight for R-squared in the optimization score. Higher values prioritize model performance.

`weight.penalization.n.predictors`
: Numeric value between 0 and 1 specifying the weight for penalizing the number of spatial predictors. Higher values favor more parsimonious models.

`optimization.method`
: Character string specifying the optimization method: `"moran.i"` (default) or `"p.value"`. Default: `"moran.i"`.

Details

This function balances three objectives when selecting spatial predictors:

1. **Reduce spatial autocorrelation**: Maximize `1 - Moran's I` to minimize residual spatial autocorrelation
2. **Maintain model performance**: Account for model R-squared
3. **Favor parsimony**: Penalize models with many spatial predictors

Optimization methods:

The **"moran.i"** method computes:
`score = (1 - Moran's I) + w1 × R² - w2 × penalization`

where all components are rescaled to the range 0 to 1, `w1` = `weight.r.squared`, and `w2` = `weight.penalization.n.predict`

The **"p.value"** method computes:
`score = max(1 - Moran's I, binary p-value) + w1 × R² - w2 × penalization`

where the binary p-value is 1 if p equal or lower than 0.05 (no significant spatial autocorrelation), and 0 otherwise.

Practical differences:

- The "moran.i" method uses continuous Moran's I values and typically selects more spatial predictors to achieve lower spatial autocorrelation
- The "p.value" method uses binary significance thresholds and typically selects fewer predictors, stopping once spatial autocorrelation becomes non-significant

The optimal model is the one with the highest optimization score.

Value

Numeric vector of optimization scores, one per row in `x`. Higher scores indicate better solutions. All values are rescaled between 0 and 1 for comparability.

See Also

`select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`, `moran()`

Other utilities: `.vif_to_df()`, `auc()`, `beowulf_cluster()`, `objects_size()`, `prepare_importance_spatial()`, `rescale_vector()`, `root_mean_squared_error()`, `setup_parallel_execution()`, `standard_error()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

Examples

```
## Not run:
# This function is typically called internally during spatial predictor selection
# Example showing the structure of input data:

# Simulated optimization data frame
opt_data <- data.frame(
  moran.i = c(0.5, 0.3, 0.2, 0.15),
  r.squared = c(0.6, 0.65, 0.68, 0.69),
  penalization.per.variable = c(0.1, 0.2, 0.3, 0.4),
  p.value.binary = c(0, 0, 1, 1)
)

# Compute optimization scores
scores_moran <- optimization_function(
  x = opt_data,
  weight.r.squared = 0.5,
  weight.penalization.n.predictors = 0.5,
  optimization.method = "moran.i"
)

# Compare methods
scores_pvalue <- optimization_function(
  x = opt_data,
  weight.r.squared = 0.5,
  weight.penalization.n.predictors = 0.5,
  optimization.method = "p.value"
)

# Higher score indicates better solution
which.max(scores_moran)
which.max(scores_pvalue)

## End(Not run)
```

---

pca *Compute Principal Component Analysis*

Description

Computes principal components from a numeric matrix or data frame with automatic scaling and zero-variance removal. Returns all principal components as a data frame. Wrapper for `stats::prcomp()`.

Usage

```
pca(x = NULL, colnames.prefix = "pca_factor")
```

Arguments

x	Numeric matrix or data frame to decompose into principal components.
colnames.prefix	Character string used as prefix for column names in the output. Default: "pca_factor".

Details

This function performs Principal Component Analysis (PCA) to create uncorrelated linear combinations of the original variables. The PCA process:

1. Removes columns with zero variance (constant values)
2. Scales all remaining variables to mean = 0 and standard deviation = 1
3. Computes principal components using singular value decomposition
4. Returns all principal components ordered by decreasing variance explained

Usage in spatial analysis:

PCA is useful for dimension reduction when working with spatial distance matrices or multiple correlated spatial predictors. It creates orthogonal (uncorrelated) variables that capture the main patterns of variation while reducing dimensionality.

For spatial modeling with `rf_spatial()`, principal components of distance matrices can serve as alternative spatial predictors to Moran's Eigenvector Maps (MEMs). Use `pca_multithreshold()` to compute PCA across multiple distance thresholds for multi-scale spatial modeling.

Value

Data frame where each column is a principal component, ordered by decreasing variance explained. Columns are named with the pattern <prefix>_<number> (e.g., "pca_factor_1", "pca_factor_2"). The number of rows matches the number of rows in x.

See Also

`pca_multithreshold()`, `mem()`, `stats::prcomp()`

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `mem_multithreshold()`, `moran()`, `moran_multithreshold()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_distance)

# Compute principal components from distance matrix
pca_components <- pca(x = plants_distance)

# View structure
head(pca_components)
dim(pca_components)

# Check column names
```

```
colnames(pca_components)[1:5]

# Custom column prefix
pca_custom <- pca(
  x = plants_distance,
  colnames.prefix = "distance_pc"
)
colnames(pca_custom)[1:3]
```

`pca_multithreshold` *Compute Principal Component Analysis at multiple distance thresholds*

Description

Computes principal components of a distance matrix at multiple distance thresholds to generate multi-scale spatial predictors for `rf_spatial()`. Each distance threshold defines a different neighborhood scale, and PCA is applied to the weighted distance matrix at each scale.

Usage

```
pca_multithreshold(
  distance.matrix = NULL,
  distance.thresholds = NULL,
  max.spatial.predictors = NULL
)
```

Arguments

`distance.matrix`
: Numeric distance matrix between observations.

`distance.thresholds`
: Numeric vector of distance thresholds defining different neighborhood scales. Each threshold specifies the maximum distance for spatial neighbors at that scale. If `NULL`, automatically computed with `default_distance_thresholds()`. Default: `NULL`.

`max.spatial.predictors`
: Integer specifying the maximum number of spatial predictors to retain. If the total number of generated predictors exceeds this value, only the first `max.spatial.predictors` are kept (ordered by variance explained). Useful for managing memory when `distance.matrix` is very large. Default: `NULL` (keeps all predictors).

Details

This function generates multi-scale spatial predictors by applying PCA to distance matrices at different neighborhood scales. The process for each distance threshold:

1. Converts the distance matrix to weights using `weights_from_distance_matrix()`, where distances above the threshold are set to zero
2. Applies `pca()` to the weighted distance matrix to extract principal components
3. Names the resulting predictors with the distance threshold for identification
4. Filters out predictors with all near-zero values

Multi-scale spatial modeling:

Different distance thresholds capture spatial patterns at different scales. Combining predictors from multiple thresholds allows `rf_spatial()` to account for spatial autocorrelation operating at multiple spatial scales simultaneously. This is analogous to `mem_multithreshold()` but uses PCA instead of Moran's Eigenvector Maps.

Comparison with MEMs:

Both `pca_multithreshold()` and `mem_multithreshold()` generate spatial predictors from distance matrices, but differ in their approach:

- **PCA**: Captures the main patterns of variation in the weighted distance matrix without considering spatial autocorrelation structure
- **MEMs**: Explicitly extracts spatial patterns with specific autocorrelation scales (positive and negative eigenvalues)

In practice, MEMs are generally preferred for spatial modeling because they explicitly target spatial autocorrelation patterns, but PCA can serve as a simpler alternative or for comparison.

Value

Data frame where each column is a spatial predictor derived from PCA at a specific distance threshold. Columns are named with the pattern `spatial_predictor_<distance>_<number>` (e.g., "spatial_predictor_1000_1", "spatial_predictor_5000_2"), where `<distance>` is the distance threshold and `<number>` is the principal component rank. The number of rows matches the number of observations in `distance.matrix`.

See Also

`pca()`, `mem_multithreshold()`, `weights_from_distance_matrix()`, `default_distance_thresholds()`

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `mem_multithreshold()`, `moran()`, `moran_multithreshold()`, `pca()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_distance)

# Compute PCA spatial predictors at multiple distance thresholds
pca_predictors <- pca_multithreshold(
  distance.matrix = plants_distance,
  distance.thresholds = c(0, 1000, 5000)
)
```

```
# View structure
head(pca_predictors)
dim(pca_predictors)

# Check predictor names (show scale information)
colnames(pca_predictors)[1:6]

# Limit number of predictors to save memory
pca_limited <- pca_multithreshold(
  distance.matrix = plants_distance,
  distance.thresholds = c(0, 1000, 5000),
  max.spatial.predictors = 20
)
ncol(pca_limited)  # At most 20 predictors
```

`plants_df` *Plant richness and predictors for American ecoregions*

Description

Vascular plant species richness for American ecoregions as defined in Ecoregions 2017.

Usage

```
data(plants_df)
```

Format

A data frame with 227 rows and 22 columns:

- `ecoregion_id`: Ecoregion identifier.
- `x`: Longitude in degrees (WGS84).
- `y`: Latitude in degrees (WGS84).
- `richness_species_vascular`: Number of vascular plant species (response variable).
- `bias_area_km2`: Ecoregion area in square kilometers.
- `bias_species_per_record`: Species count divided by GBIF spatial records (sampling bias metric).
- `climate_aridity_index_average`: Average aridity index.
- `climate_hypervolume`: Climatic envelope volume computed with hypervolume.
- `climate_velocity_lgm_average`: Average climate velocity since the Last Glacial Maximum.
- `neighbors_count`: Number of immediate neighbors (connectivity metric).
- `neighbors_percent_shared_edge`: Percentage of shared edge with neighbors (connectivity metric).

- human_population_density: Human population density.
- topography_elevation_average: Average elevation.
- landcover_herbs_percent_average: Average herb cover from MODIS Vegetation Continuous Fields.
- fragmentation_cohesion: Cohesion index computed with landscapemetrics.
- fragmentation_division: Division index computed with landscapemetrics.
- neighbors_area: Total area of immediate neighbors.
- human_population: Total human population.
- human_footprint_average: Average human footprint index.
- climate_bio1_average: Average mean annual temperature.
- climate_bio15_minimum: Minimum precipitation seasonality.

See Also

Other data: plants_distance, plants_predictors, plants_response, plants_rf, plants_rf_spatial, plants_xy

---

plants_distance *Distance matrix between ecoregion edges*

Description

Distance matrix (in km) between the edges of American ecoregions in plants_df.

Usage

```
data(plants_distance)
```

Format

Numeric matrix with 227 rows and 227 columns.

See Also

Other data: plants_df, plants_predictors, plants_response, plants_rf, plants_rf_spatial, plants_xy

plants_predictors *Predictor variable names for plant richness examples*

Description

Character vector of predictor variable names from plants_df (columns 5 to 21).

Usage

```
data(plants_predictors)
```

Format

A character vector of length 17.

See Also

Other data: plants_df, plants_distance, plants_response, plants_rf, plants_rf_spatial, plants_xy

plants_response *Response variable name for plant richness examples*

Description

Character string containing the name of the response variable in plants_df: "richness_species_vascular".

Usage

```
data(plants_response)
```

Format

A character string of length 1.

See Also

Other data: plants_df, plants_distance, plants_predictors, plants_rf, plants_rf_spatial, plants_xy

plants_rf *Example fitted random forest model*

Description

Fitted random forest model using plants_df. Provided for testing and examples without requiring model fitting. Fitted with reduced complexity for faster computation and smaller object size.

Usage

```
data(plants_rf)
```

Format

An object of class `rf` fitted with the following parameters:

- `data`: plants_df
- `dependent.variable.name`: plants_response ("richness_species_vascular")
- `predictor.variable.names`: plants_predictors (17 variables)
- `distance.matrix`: plants_distance
- `xy`: plants_xy
- `distance.thresholds`: `c(100, 1000, 2000, 4000)`
- `num.trees`: 50
- `min.node.size`: 30
- `n.cores`: 1

Details

This model uses reduced complexity (50 trees, min.node.size = 30) to keep object size small for package distribution. For actual analyses, use higher values (e.g., num.trees = 500, min.node.size = 5).

See Also

`rf()`, plants_df, plants_response, plants_predictors

Other data: `plants_df`, `plants_distance`, `plants_predictors`, `plants_response`, `plants_rf_spatial`, `plants_xy`

plants_rf_spatial *Example fitted spatial random forest model*

Description

Fitted spatial random forest model using plants_df with spatial predictors from Moran's Eigenvector Maps. Provided for testing and examples without requiring model fitting. Fitted with reduced complexity for faster computation and smaller object size.

Usage

```
data(plants_rf_spatial)
```

Format

An object of class `rf` fitted with the following parameters:

- `data`: plants_df
- `dependent.variable.name`: plants_response ("richness_species_vascular")
- `predictor.variable.names`: plants_predictors (17 variables)
- `distance.matrix`: plants_distance
- `xy`: plants_xy
- `distance.thresholds`: `c(100, 1000, 2000, 4000)`
- `method`: "mem.effect.recursive"
- `num.trees`: 50
- `min.node.size`: 30
- `n.cores`: 14

Details

This spatial model includes spatial predictors (Moran's Eigenvector Maps) selected using the recursive method to minimize residual spatial autocorrelation. Uses reduced complexity (50 trees, min.node.size = 30) to keep object size small for package distribution. For actual analyses, use higher values (e.g., num.trees = 500, min.node.size = 5).

See Also

`rf_spatial()`, `rf()`, plants_rf, plants_df, plants_response, plants_predictors

Other data: `plants_df`, `plants_distance`, `plants_predictors`, `plants_response`, `plants_rf`, `plants_xy`

plants_xy *Coordinates for plant richness data*

Description

Spatial coordinates (longitude and latitude) extracted from plants_df for use in spatial modeling functions.

Usage

```
data(plants_xy)
```

Format

A data frame with 227 rows and 2 columns:

- x: Longitude in degrees (WGS84).
- y: Latitude in degrees (WGS84).

See Also

Other data: plants_df, plants_distance, plants_predictors, plants_response, plants_rf, plants_rf_spatial

plot_evaluation *Visualize spatial cross-validation results*

Description

Creates boxplots comparing model performance metrics across training, testing, and full datasets from spatial cross-validation performed by rf_evaluate(). Displays distributions of R-squared, RMSE, and other metrics across all spatial folds.

Usage

```
plot_evaluation(
  model,
  fill.color = viridis::viridis(3, option = "F", alpha = 0.8, direction = -1),
  line.color = "gray30",
  verbose = TRUE,
  notch = TRUE
)
```

Arguments

`model`	Model fitted with `rf_evaluate()`. Must be of class "rf_evaluate".
`fill.color`	Character vector with three colors (one for each model type: Testing, Training, Full) or a function that generates a color palette. Accepts hexadecimal codes (e.g., `c("#440154FF", "#21908CFF", "#FDE725FF")`) or palette functions (e.g., `viridis::viridis(3)`). Default: `viridis::viridis(3, option = "F", alpha = 0.8, direction = -1)`.
`line.color`	Character string specifying the color of boxplot borders. Default: `"gray30"`.
`verbose`	Logical. If `TRUE`, prints the plot to the graphics device. Default: `TRUE`.
`notch`	Logical. If `TRUE`, displays notched boxplots where notches represent approximate 95% confidence intervals around the median. Non-overlapping notches suggest significant differences between medians. Default: `TRUE`.

Details

This function visualizes the distribution of performance metrics across spatial folds, with separate boxplots for three model variants:

- **Testing**: Performance on spatially independent testing folds (most reliable estimate of generalization)
- **Training**: Performance on training folds (typically optimistic)
- **Full**: Performance on the complete dataset (reference baseline)

Interpreting the plot:

The boxplots show the distribution of each metric across all spatial folds. Ideally:

- Testing performance should be reasonably close to training performance (indicates good generalization)
- Large gaps between training and testing suggest overfitting
- Low variance across folds indicates stable, consistent model performance
- High variance suggests performance depends strongly on spatial location

The plot includes a title showing the number of spatial folds used in the evaluation.

Available metrics:

Displayed metrics depend on the response variable type:

- **Continuous response**: R-squared, RMSE (Root Mean Squared Error), NRMSE (Normalized RMSE)
- **Binary response**: AUC (Area Under ROC Curve), pseudo R-squared

Value

ggplot object that can be further customized or saved. The plot displays boxplots of performance metrics (R-squared, RMSE, NRMSE, pseudo R-squared, or AUC depending on model type) across spatial folds, faceted by metric.

See Also

rf_evaluate(), get_evaluation(), print_evaluation()

Other visualization: plot_importance(), plot_moran(), plot_optimization(), plot_residuals_diagnostics(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
if(interactive()){

data(plants_rf, plants_xy)

# Perform spatial cross-validation
plants_rf <- rf_evaluate(
  model = plants_rf,
  xy = plants_xy,
  repetitions = 5,
  n.cores = 1
)

# Visualize evaluation results
plot_evaluation(plants_rf)

# Without notches for simpler boxplots
plot_evaluation(plants_rf, notch = FALSE)

# Custom colors
plot_evaluation(
  plants_rf,
  fill.color = c("#E64B35FF", "#4DBBD5FF", "#00A087FF")
)

# Print summary statistics
print_evaluation(plants_rf)

# Extract evaluation data for custom analysis
evaluation_data <- get_evaluation(plants_rf)
head(evaluation_data)

}
```

---

plot_importance *Visualize variable importance scores*

Description

Creates a visualization of variable importance scores from models fitted with rf(), rf_repeat(), or rf_spatial(). For single-run models (rf(), rf_spatial()), displays points ordered by importance. For repeated models (rf_repeat()), displays violin plots showing the distribution of importance scores across model repetitions.

Usage

```
plot_importance(
  model,
 fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 1, end = 0.9),
  line.color = "white",
  verbose = TRUE
)
```

Arguments

model	Model fitted with rf(), rf_repeat(), or rf_spatial(). Alternatively, a data frame with variable importance scores (for internal use only).
fill.color	Character vector of colors or a function generating a color palette. Accepts hexadecimal codes (e.g., c("#440154FF", "#21908CFF", "#FDE725FF")) or palette functions (e.g., viridis::viridis(100)). For single-run models, creates a continuous gradient. For repeated models, assigns discrete colors to variables. Default: viridis::viridis(100, option = "F", direction = -1, alpha = 1, end = 0.9).
line.color	Character string specifying the color of point borders (single-run models) or violin plot outlines (repeated models). Default: "white".
verbose	Logical. If TRUE, prints the plot to the graphics device. Default: TRUE.

Details

This function creates different visualizations depending on the model type:

Single-run models (rf(), rf_spatial() without repetitions):

- Displays points showing the importance value for each variable
- Variables ordered top-to-bottom by importance (most important at top)
- Point color represents importance magnitude using a continuous gradient

Repeated models (rf_repeat(), rf_spatial() with repetitions):

- Displays violin plots showing the distribution of importance across repetitions
- Variables ordered top-to-bottom by median importance (most important at top)
- The median line within each violin shows the center of the distribution
- Width of violin reflects the density of importance values at each level
- Each variable receives a distinct fill color

Importance metric:

The x-axis shows permutation importance, which measures the increase in prediction error when a variable's values are randomly shuffled. Higher values indicate more important variables. Importance is computed on out-of-bag (OOB) samples, providing an unbiased estimate of variable contribution.

Spatial predictors:

In rf_spatial() models, all spatial predictors (MEMs or PCA factors) are grouped into a single category labeled "spatial_predictors" to simplify comparison with non-spatial predictors.

Note on violin plots:

Violin plots display kernel density estimates. The median line shown is the median of the density estimate, which may differ slightly from the actual data median. However, variables are always ordered by the true median importance to ensure accurate ranking.

Cross-validated importance:

This function does not plot results from rf_importance(). For cross-validated importance plots, access model$importance$cv.per.variable.plot after running rf_importance().

Value

ggplot object that can be further customized or saved. The plot displays variable importance on the x-axis and variable names on the y-axis, ordered by importance (highest at top).

See Also

print_importance(), get_importance(), rf_importance()

Other visualization: plot_evaluation(), plot_moran(), plot_optimization(), plot_residuals_diagnostics(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
data(plants_rf, plants_rf_spatial)

# Plot importance from Random Forest model
plot_importance(plants_rf)

# Plot importance from Spatial Random Forest model
plot_importance(plants_rf_spatial)
```

---

plot_moran *Plots a Moran's I test of model residuals*

Description

Plots the results of spatial autocorrelation tests for a variety of functions within the package. The x axis represents the Moran's I estimate, the y axis contains the values of the distance thresholds, the dot sizes represent the p-values of the Moran's I estimate, and the red dashed line represents the theoretical null value of the Moran's I estimate.

Usage

```
plot_moran(
  model,
  point.color = viridis::viridis(
    100,
    option = "F",
    direction = -1
   ),
  line.color = "gray30",
  option = 1,
  ncol = 1,
  verbose = TRUE
)
```

Arguments

model	A model fitted with rf(), rf_repeat(), or rf_spatial(), or a data frame generated by moran(). Default: NULL
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F")
line.color	Character string, color of the line produced by ggplot2::geom_smooth(). Default: "gray30"
option	Integer, type of plot. If 1 (default) a line plot with Moran's I and p-values across distance thresholds is returned. If 2, scatterplots of residuals versus lagged residuals per distance threshold and their corresponding slopes are returned. In models fitted with rf_repeat(), the residuals and lags of the residuals are computed from the median residuals across repetitions. Option 2 is disabled if x is a data frame generated by moran().
ncol	Number of columns of the plot. Only relevant when option = 2. Argument ncol of wrap_plots.
verbose	Logical, if TRUE, the resulting plot is printed, Default: TRUE

Value

A ggplot.

See Also

moran(), moran_multithreshold()

Other visualization: plot_evaluation(), plot_importance(), plot_optimization(), plot_residuals_diagnostics(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
data(plants_rf)

plot_moran(plants_rf)

plot_moran(plants_rf, option = 2)
```

`plot_optimization` *Optimization plot of a selection of spatial predictors*

Description

Plots optimization data frames produced by `select_spatial_predictors_sequential()` and `select_spatial_predictors_recursive()`.

Usage

```
plot_optimization(
  model,
  point.color = viridis::viridis(
    100,
    option = "F",
    direction = -1
  ),
  verbose = TRUE
)
```

Arguments

`model`	A model produced by `rf_spatial()`, or an optimization data frame produced by `select_spatial_predictors_sequential()` or `select_spatial_predictors_recursive()`.
`point.color`	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. `viridis::viridis(100)`). Default: `viridis::viridis(100, option = "F", direction = -1)`
`verbose`	Logical, if `TRUE` the plot is printed. Default: `TRUE`

Details

The function returns `NULL` invisibly (without plotting) when:

- The method used to fit a model with `rf_spatial()` is "hengl" (no optimization required)
- No spatial predictors were selected during model fitting
- The model is non-spatial

Value

A ggplot, or NULL invisibly if no optimization data is available.

See Also

Other visualization: plot_evaluation(), plot_importance(), plot_moran(), plot_residuals_diagnostics(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
data(plants_rf_spatial)

plot_optimization(plants_rf_spatial)
```

---

plot_residuals_diagnostics

Plot residuals diagnostics

Description

Plots normality and autocorrelation tests of model residuals.

Usage

```
plot_residuals_diagnostics(
  model,
  point.color = viridis::viridis(100, option = "F"),
  line.color = "gray10",
  fill.color = viridis::viridis(4, option = "F", alpha = 0.95)[2],
  option = 1,
  ncol = 1,
  verbose = TRUE
)
```

Arguments

model	A model produced by rf(), rf_repeat(), or rf_spatial().
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F")
line.color	Character string, color of the line produced by ggplot2::geom_smooth(). Default: "gray30"
fill.color	Character string, fill color of the bars produced by ggplot2::geom_histogram(). Default: viridis::viridis(4, option = "F", alpha = 0.95)[2]

option	(argument of plot_moran()) Integer, type of plot. If 1 (default) a line plot with Moran's I and p-values across distance thresholds is returned. If 2, scatterplots of residuals versus lagged residuals per distance threshold and their corresponding slopes are returned. In models fitted with rf_repeat(), the residuals and lags of the residuals are computed from the median residuals across repetitions. Option 2 is disabled if x is a data frame generated by moran().
ncol	(argument of plot_moran()) Number of columns of the Moran's I plot if option = 2.
verbose	Logical, if TRUE, the resulting plot is printed, Default: TRUE

Value

A patchwork object.

See Also

Other visualization: plot_evaluation(), plot_importance(), plot_moran(), plot_optimization(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
data(plants_rf)

plot_residuals_diagnostics(plants_rf)
```

---

plot_response_curves *Plots the response curves of a model.*

Description

Plots the response curves of models fitted with rf(), rf_repeat(), or rf_spatial().

Usage

```
plot_response_curves(
  model = NULL,
  variables = NULL,
  quantiles = c(0.1, 0.5, 0.9),
  grid.resolution = 200,
  line.color = viridis::viridis(length(quantiles), option = "F", end = 0.9),
  ncol = 2,
  show.data = FALSE,
  verbose = TRUE
)
```

Arguments

model	A model fitted with rf(), rf_repeat(), or rf_spatial().
variables	Character vector, names of predictors to plot. If NULL, the most important variables (importance higher than the median) in x are selected. Default: NULL.
quantiles	Numeric vector with values between 0 and 1, argument probs of quantile. Quantiles to set the other variables to. Default: c(0.1, 0.5, 0.9)
grid.resolution	Integer between 20 and 500. Resolution of the plotted curve Default: 100
line.color	Character vector with colors, or function to generate colors for the lines representing quantiles. Must have the same number of colors as quantiles are defined. Default: viridis::viridis(length(quantiles), option = "F", end = 0.9)
ncol	Integer, argument of wrap_plots. Defaults to the rounded squared root of the number of plots. Default: 2
show.data	Logical, if TRUE, the observed data is plotted along with the response curves. Default: FALSE
verbose	Logical, if TRUE the plot is printed. Default: TRUE

Details

All variables that are not plotted in a particular response curve are set to the values of their respective quantiles, and the response curve for each one of these quantiles is shown in the plot. When the input model was fitted with rf_repeat() with keep.models = TRUE, then the plot shows the median of all model runs, and each model run separately as a thinner line. The output list can be plotted all at once with patchwork::wrap_plots(p) or cowplot::plot_grid(plotlist = p), or one by one by extracting each plot from the list.

Value

A list with slots named after the selected variables, with one ggplot each.

See Also

plot_response_surface()

Other visualization: plot_evaluation(), plot_importance(), plot_moran(), plot_optimization(), plot_residuals_diagnostics(), plot_response_surface(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
data(plants_rf)

plot_response_curves(
  model = plants_rf,
  variables = "climate_bio1_average"
)
```

```
plot_response_curves(
  model = plants_rf,
  variables = "climate_bio1_average",
  show.data = TRUE
)
```

`plot_response_surface` *Plots the response surfaces of a random forest model*

Description

Plots response surfaces for any given pair of predictors in a `rf()`, `rf_repeat()`, or `rf_spatial()` model.

Usage

```
plot_response_surface(
  model = NULL,
  a = NULL,
  b = NULL,
  quantiles = 0.5,
  grid.resolution = 100,
  point.size.range = c(0.5, 2.5),
  point.alpha = 1,
  fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 0.9),
  point.color = "gray30",
  verbose = TRUE
)
```

Arguments

`model`	A model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`. Default `NULL`
`a`	Character string, name of a model predictor. If `NULL`, the most important variable in `model` is selected. Default: `NULL`
`b`	Character string, name of a model predictor. If `NULL`, the second most important variable in `model` is selected. Default: `NULL`
`quantiles`	Numeric vector between 0 and 1. Argument `probs` of the function quantile. Quantiles to set the other variables to. Default: `0.5`
`grid.resolution`	Integer between 20 and 500. Resolution of the plotted surface Default: `100`
`point.size.range`	Numeric vector of length 2 with the range of point sizes used by geom_point. Using `c(-1, -1)` removes the points. Default: `c(0.5, 2.5)`
`point.alpha`	Numeric between 0 and 1, transparency of the points. Setting it to `0` removes all points. Default: `1`.

fill.color	Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F", direction = -1, alpha = 0.9)
point.color	Character vector with a color name (e.g. "red4"). Default: gray30
verbose	Logical, if TRUE the plot is printed. Default: TRUE

Details

All variables that are not a or b in a response curve are set to the values of their respective quantiles to plot the response surfaces. The output list can be plotted all at once with patchwork::wrap_plots(p) or cowplot::plot_grid(plotlist = p), or one by one by extracting each plot from the list.

Value

A list with slots named after the selected quantiles, each one with a ggplot.

See Also

plot_response_curves()

Other visualization: plot_evaluation(), plot_importance(), plot_moran(), plot_optimization(), plot_residuals_diagnostics(), plot_response_curves(), plot_training_df(), plot_training_df_moran(), plot_tuning()

Examples

```
data(plants_rf)

plot_response_surface(
  model = plants_rf,
  a = "climate_bio1_average",
  b = "human_population",
  grid.resolution = 50
)
```

---

plot_training_df *Scatterplots of a training data frame*

Description

Plots the dependent variable against each predictor.

Usage

```
plot_training_df(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  ncol = 4,
  method = "loess",
  point.color = viridis::viridis(100, option = "F"),
  line.color = "gray30"
)
```

Arguments

`data`	Data frame with a response variable and a set of predictors. Default: `NULL`
`dependent.variable.name`	Character string with the name of the response variable. Must be in the column names of `data`. If the dependent variable is binary with values 1 and 0, the argument `case.weights` of `ranger` is populated by the function `case_weights()`. Default: `NULL`
`predictor.variable.names`	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of `data`. Optionally, the result of `auto_cor()` or `auto_vif()` Default: `NULL`
`ncol`	Number of columns of the plot. Argument `ncol` of wrap_plots.
`method`	Method for geom_smooth, one of: "lm", "glm", "gam", "loess", or a function, for example `mgcv::gam` Default: 'loess'
`point.color`	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. `viridis::viridis(100)`). Default: `viridis::viridis(100, option = "F")`
`line.color`	Character string, color of the line produced by `ggplot2::geom_smooth()`. Default: `"gray30"`

Value

A wrap_plots object.

See Also

Other visualization: `plot_evaluation()`, `plot_importance()`, `plot_moran()`, `plot_optimization()`, `plot_residuals_diagnostics()`, `plot_response_curves()`, `plot_response_surface()`, `plot_training_df_moran()`, `plot_tuning()`

Examples

```
data(
  plants_df,
  plants_response,
```

```
  plants_predictors
)

plot_training_df(
  data = plants_df,
  dependent.variable.name = plants_response,
  predictor.variable.names = plants_predictors[1:4]
)
```

`plot_training_df_moran` *Moran's I plots of a training data frame*

Description

Plots the the Moran's I test of the response and the predictors in a training data frame.

Usage

```
plot_training_df_moran(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  fill.color = viridis::viridis(100, option = "F", direction = -1),
  point.color = "gray30"
)
```

Arguments

`data`	Data frame with a response variable and a set of predictors. Default: `NULL`
`dependent.variable.name`	Character string with the name of the response variable. Must be in the column names of `data`. If the dependent variable is binary with values 1 and 0, the argument `case.weights` of `ranger` is populated by the function `case_weights()`. Default: `NULL`
`predictor.variable.names`	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of `data`. Optionally, the result of `auto_cor()` or `auto_vif()` Default: `NULL`
`distance.matrix`	Squared matrix with the distances among the records in `data`. The number of rows of `distance.matrix` and `data` must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: `NULL`

distance.thresholds
: Numeric vector, distances below each value are set to 0 on separated copies of the distance matrix for the computation of Moran's I at different neighborhood distances. If NULL, it defaults to seq(0, max(distance.matrix)/4, length.out = 2). Default: NULL

fill.color
: Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F", direction = -1)

point.color
: Character vector with a color name (e.g. "red4"). Default: gray30

Value

A ggplot2 object.

See Also

Other visualization: plot_evaluation(), plot_importance(), plot_moran(), plot_optimization(), plot_residuals_diagnostics(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_tuning()

Examples

```
data(
  plants_df,
  plants_response,
  plants_predictors,
  plants_distance
)

plot_training_df_moran(
  data = plants_df,
  dependent.variable.name = plants_response,
  predictor.variable.names = plants_predictors[1:4],
  distance.matrix = plants_distance,
  distance.thresholds = c(1000, 2000, 4000)
)
```

---

plot_tuning *Plots a tuning object produced by rf_tuning()*

Description

Plots the tuning of the hyperparameters num.trees, mtry, and min.node.size performed by rf_tuning().

Usage

```
plot_tuning(
  model,
  point.color = viridis::viridis(
    100,
    option = "F"
  ),
  verbose = TRUE
)
```

Arguments

model	A model fitted with rf_tuning(). Default: NULL
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F")
verbose	Logical, if TRUE, the plot is printed. Default: TRUE

Value

A ggplot.

See Also

rf_tuning()

Other visualization: plot_evaluation(), plot_importance(), plot_moran(), plot_optimization(), plot_residuals_diagnostics(), plot_response_curves(), plot_response_surface(), plot_training_df(), plot_training_df_moran()

Examples

```
if(interactive()){
  data(
    plants_rf,
    plants_xy
  )

  plants_rf_tuned <- rf_tuning(
    model = plants_rf,
    num.trees = c(25, 50),
    mtry = c(5, 10),
    min.node.size = c(10, 20),
    xy = plants_xy,
    repetitions = 5,
    n.cores = 1
  )

  plot_tuning(plants_rf_tuned)
```

```
}
```

`prepare_importance_spatial` *Prepares variable importance objects for spatial models*

Description

Prepares variable importance data frames and plots for models fitted with `rf_spatial()`.

Usage

```
prepare_importance_spatial(model)
```

Arguments

`model`	An importance data frame with spatial predictors, or a model fitted with `rf_spatial()`.

Value

A list with importance data frames in different formats depending on whether the model was fitted with `rf()` or `rf_repeat()`.

See Also

Other utilities: `.vif_to_df()`, `auc()`, `beowulf_cluster()`, `objects_size()`, `optimization_function()`, `rescale_vector()`, `root_mean_squared_error()`, `setup_parallel_execution()`, `standard_error()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

Examples

```
data(plants_rf_spatial)

prepare_importance_spatial(plants_rf_spatial) %>%
  head()
```

print.rf *Custom print method for random forest models*

Description

Custom print method for models fitted with `rf()`, `rf_repeat()`, and `rf_spatial()`.

Usage

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

Arguments

x	A model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.
...	Additional arguments for print methods.

Value

Prints model details to the console.

See Also

`print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print_evaluation()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
data(plants_rf)

print(plants_rf)

#or
plants_rf
```

print_evaluation *Prints cross-validation results*

Description

Prints the results of an spatial cross-validation performed with `rf_evaluate()`.

Usage

```
print_evaluation(model)
```

Arguments

model	A model resulting from `rf_evaluate()`.

Value

A table printed to the standard output.

See Also

`plot_evaluation()`, `get_evaluation()`

Other model_info: `get_evaluation()`, `get_importance()`, `get_importance_local()`, `get_moran()`, `get_performance()`, `get_predictions()`, `get_residuals()`, `get_response_curves()`, `get_spatial_predictors()`, `print.rf()`, `print_importance()`, `print_moran()`, `print_performance()`

Examples

```
if(interactive()){

data(
  plants_rf,
  plants_xy
)

plants_rf <- rf_evaluate(
  model = plants_rf,
  xy = plants_xy,
  repetitions = 5,
  n.cores = 1
)

print_evaluation(plants_rf)

}
```

print_importance *Prints variable importance*

Description

Prints variable importance scores from rf, rf_repeat, and rf_spatial models.

Usage

```
print_importance(
  model,
  verbose = TRUE
)
```

Arguments

model	A model fitted with rf, rf_repeat, or rf_spatial.
verbose	Logical, if TRUE, variable importance is returned. Default: TRUE

Value

A table printed to the standard output.

See Also

plot_importance(), get_importance()

Other model_info: get_evaluation(), get_importance(), get_importance_local(), get_moran(), get_performance(), get_predictions(), get_residuals(), get_response_curves(), get_spatial_predictors(), print.rf(), print_evaluation(), print_moran(), print_performance()

Examples

```
data(plants_rf)

print_importance(plants_rf)
```

print_moran *Prints results of a Moran's I test*

Description

Prints the results of a Moran's I test on the residuals of a model.

Usage

```
print_moran(
  model,
  caption = NULL,
  verbose = TRUE
)
```

Arguments

model	A model fitted with rf(), rf_repeat(), or rf_spatial().
caption	Character, caption of the output table, Default: NULL
verbose	Logical, if TRUE, the resulting table is printed into the console, Default: TRUE

Value

Prints a table in the console using the huxtable package.

See Also

moran(), moran_multithreshold(), get_moran(), plot_moran()

Other model_info: get_evaluation(), get_importance(), get_importance_local(), get_moran(), get_performance(), get_predictions(), get_residuals(), get_response_curves(), get_spatial_predictors(), print.rf(), print_evaluation(), print_importance(), print_performance()

Examples

```
data(plants_rf)

print_moran(plants_rf)
```

print_performance *print_performance*

Description

Prints the performance slot of a model fitted with rf(), rf_repeat(), or rf_spatial(). For models fitted with rf_repeat() it shows the median and the median absolute deviation of each performance measure.

Usage

```
print_performance(model)
```

Arguments

model	Model fitted with rf(), rf_repeat(), or rf_spatial().

Value

Prints model performance scores to the console.

See Also

print_performance(), get_performance()

Other model_info: get_evaluation(), get_importance(), get_importance_local(), get_moran(), get_performance(), get_predictions(), get_residuals(), get_response_curves(), get_spatial_predictors(), print.rf(), print_evaluation(), print_importance(), print_moran()

Examples

```
data(plants_rf)

print_performance(plants_rf)
```

rank_spatial_predictors *Ranks spatial predictors*

Description

Ranks spatial predictors generated by mem_multithreshold() or pca_multithreshold() by their effect in reducing the Moran's I of the model residuals (ranking.method = "effect"), or by their own Moran's I (ranking.method = "moran").

In the former case, one model of the type y ~ predictors + spatial_predictor_X is fitted per spatial predictor, and the Moran's I of this model's residuals is compared with the one of the model without spatial predictors (y ~ predictors), to finally rank the spatial predictor from maximum to minimum difference in Moran's I.

In the latter case, the spatial predictors are ordered by their Moran's I alone (this is the faster option).

In both cases, spatial predictors that are redundant with others at a Pearson correlation > 0.5 and spatial predictors with no effect (no reduction of Moran's I or Moran's I of the spatial predictor equal or lower than 0) are removed.

This function has been designed to be used internally by rf_spatial() rather than directly by a user.

Usage

```
rank_spatial_predictors(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  ranger.arguments = NULL,
  spatial.predictors.df = NULL,
  ranking.method = c("moran", "effect"),
  reference.moran.i = 1,
  verbose = FALSE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

data	Data frame with a response variable and a set of predictors. Default: NULL
dependent.variable.name	Character string with the name of the response variable. Must be in the column names of data. Default: NULL
predictor.variable.names	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL
distance.matrix	Squared matrix with the distances among the records in data. The number of rows of distance.matrix and data must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL

`distance.thresholds`
: Numeric vector with neighborhood distances. All distances in the distance matrix below each value in `dustance.thresholds` are set to 0 for the computation of Moran's I. If `NULL`, it defaults to seq(0, max(distance.matrix), length.out = 4). Default: `NULL`

`ranger.arguments`
: List with ranger arguments. See rf or rf_repeat for further details.

`spatial.predictors.df`
: Data frame of spatial predictors.

`ranking.method`
: Character, method used by to rank spatial predictors. The method "effect" ranks spatial predictors according how much each predictor reduces Moran's I of the model residuals, while the method "moran" ranks them by their own Moran's I. Default: `"moran"`.

`reference.moran.i`
: Moran's I of the residuals of the model without spatial predictors. Default: `1`

`verbose`
: Logical, ff `TRUE`, messages and plots generated during the execution of the function are displayed, Default: `TRUE`

`n.cores`
: Integer, number of cores to use for parallel execution. Creates a socket cluster with `parallel::makeCluster()`, runs operations in parallel with `foreach` and `%dopar%`, and stops the cluster with `parallel::clusterStop()` when the job is done. Default: `parallel::detectCores() - 1`

`cluster`
: A cluster definition generated with `parallel::makeCluster()`. If provided, overrides `n.cores`. When `cluster = NULL` (default value), and `model` is provided, the cluster in `model`, if any, is used instead. If this cluster is `NULL`, then the function uses `n.cores` instead. The function does not stop a provided cluster, so it should be stopped with `parallel::stopCluster()` afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the `model` argument, or using the `%>%` pipe. Default: `NULL`

Value

A list with four slots:

- `method`: Character, name of the method used to rank the spatial predictors.
- `criteria`: Data frame with two different configurations depending on the ranking method. If `ranking.method = "effect"`, the columns contain the names of the spatial predictors, the r-squared of the model, the Moran's I of the model residuals, the difference between the Moran's I of the model including the given spatial predictor, and the Moran's I of the model fitted without spatial predictors, and the interpretation of the Moran's I value. If `ranking.method = "moran"`, only the name of the spatial predictor and it's Moran's I are in the output data frame.
- `ranking`: Ordered character vector with the names of the spatial predictors selected.
- `spatial.predictors.df`: data frame with the selected spatial predictors in the order of the ranking.

See Also

Other spatial_analysis: filter_spatial_predictors(), mem(), mem_multithreshold(), moran(), moran_multithreshold(), pca(), pca_multithreshold(), residuals_diagnostics(), residuals_test(), select_spatial_predictors_recursive(), select_spatial_predictors_sequential()

Examples

```
if(interactive()){

data(
  plants_df,
  plants_response,
  plants_distance
)

#subset to speed up example
idx <- 50:90
plants_distance_sub <- plants_distance[idx, idx]

y <- mem(
  distance.matrix = plants_distance_sub,
  distance.threshold = 1000
)

#rank spatial predictors by Moran's I
y_rank <- rank_spatial_predictors(
  distance.matrix = plants_distance_sub,
  distance.thresholds = 1000,
  spatial.predictors.df = y,
  ranking.method = "moran",
  n.cores = 1
)

y_rank$criteria
y_rank$ranking

#rank spatial predictors by association with response
y_rank <- rank_spatial_predictors(
  data = plants_df[idx, ],
  dependent.variable.name = plants_response,
  distance.matrix = plants_distance_sub,
  distance.thresholds = 1000,
  spatial.predictors.df = y,
  ranking.method = "effect",
  n.cores = 1
)

y_rank$criteria
y_rank$ranking

}
```

rescale_vector *Rescales a numeric vector into a new range*

Description

Rescales a numeric vector to a new range.

Usage

```
rescale_vector(
  x = NULL,
  new.min = 0,
  new.max = 1,
  integer = FALSE
)
```

Arguments

x	Numeric vector. Default: NULL
new.min	New minimum value. Default: 0
new.max	New maximum value. Default: 1
integer	Logical, if TRUE, coerces the output to integer. Default: FALSE

Value

A numeric vector of the same length as x, but with its values rescaled between new.min and new.max.

See Also

Other utilities: .vif_to_df(), auc(), beowulf_cluster(), objects_size(), optimization_function(), prepare_importance_spatial(), root_mean_squared_error(), setup_parallel_execution(), standard_error(), statistical_mode(), thinning(), thinning_til_n()

Examples

```
y <- rescale_vector(
  x = rnorm(100),
  new.min = 0,
  new.max = 100,
  integer = TRUE
)
y
```

`residuals_diagnostics` *Normality test of a numeric vector*

Description

Applies a Shapiro-Wilks test to a numeric vector, and plots the qq plot and the histogram.

Usage

```
residuals_diagnostics(residuals, predictions)
```

Arguments

`residuals`	Numeric vector, model residuals.
`predictions`	Numeric vector, model predictions.

Details

The function `shapiro.test()` has a hard limit of 5000 cases. If the model residuals have more than 5000 cases, then `sample(x = residuals, size = 5000)` is applied to the model residuals before the test.

Value

A list with four slots:

/item `w` W statistic returned by `shapiro.test()`. /item `p.value` p-value of the Shapiro test. /item `interpretation` Character vector, one of "x is normal", "x is not normal". /item `plot` A patchwork plot with the qq plot and the histogram of x.

See Also

`ggplot`,`aes`,`geom_qq_line`,`ggtheme`,`labs`,`geom_freqpoly`,`geom_abline` `plot_annotation`

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `mem_multithreshold()`, `moran()`, `moran_multithreshold()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_test()`, `select_spatial_predictors_recursive()`, `select_spatial_predictors_sequential()`

Examples

```
data(plants_rf)

y <- residuals_diagnostics(
  residuals = get_residuals(plants_rf),
  predictions = get_predictions(plants_rf)
)
y
```

residuals_test *Normality test of a numeric vector*

Description

Applies a Shapiro-Wilks test to a numeric vector, and returns a list with the statistic W, its p-value, and a character string with the interpretation.

Usage

```
residuals_test(residuals)
```

Arguments

residuals	Numeric vector, model residuals.

Value

A list with four slots:

/item w W statistic returned by shapiro.test(). /item p.value p-value of the Shapiro test. /item interpretation Character vector, one of "x is normal", "x is not normal". /item plot A patchwork plot with the qq plot and the histogram of x.

See Also

Other spatial_analysis: filter_spatial_predictors(), mem(), mem_multithreshold(), moran(), moran_multithreshold(), pca(), pca_multithreshold(), rank_spatial_predictors(), residuals_diagnostics(), select_spatial_predictors_recursive(), select_spatial_predictors_sequential()

Examples

```
residuals_test(residuals = runif(100))
```

rf *Random forest models with Moran's I test of the residuals*

Description

Fits a random forest model using ranger and extends it with spatial diagnostics: residual autocorrelation (Moran's I) at multiple distance thresholds, performance metrics (RMSE, NRMSE via root_mean_squared_error()), and variable importance scores computed on scaled data (via scale).

Usage

```
rf(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  xy = NULL,
  ranger.arguments = NULL,
  scaled.importance = FALSE,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

data	Data frame with a response variable and a set of predictors. Default: NULL
dependent.variable.name	Character string with the name of the response variable. Must be a column name in data. For binary response variables (0/1), case weights are automatically computed using case_weights() to balance classes. Default: NULL
predictor.variable.names	Character vector with predictor variable names. All names must be columns in data. Alternatively, accepts the output of auto_cor() or auto_vif() for automated variable selection. Default: NULL
distance.matrix	Square matrix with pairwise distances between observations in data. Must have the same number of rows as data. If NULL, spatial autocorrelation of residuals is not computed. Default: NULL
distance.thresholds	Numeric vector of distance thresholds for spatial autocorrelation analysis. For each threshold, distances below that value are set to zero when computing Moran's I. If NULL, defaults to seq(0, max(distance.matrix), length.out = 4). Default: NULL
xy	Data frame or matrix with two columns containing coordinates, named "x" and "y". Not used by this function but stored in the model for use by rf_evaluate() and rf_tuning(). Default: NULL
ranger.arguments	Named list with ranger arguments. Arguments for this function can also be passed here. The default importance method is 'permutation' instead of ranger's default 'none'. The x, y, and formula arguments are not supported. See ranger help for available arguments. Default: NULL
scaled.importance	If TRUE, variable importance is computed on scaled data using scale, making importance scores comparable across models with different predictor units. Default: FALSE

seed	Random seed for reproducibility. Default: 1
verbose	If TRUE, display messages and plots during execution. Default: TRUE
n.cores	Number of cores for parallel execution. Default: parallel::detectCores() - 1
cluster	Cluster object from parallel::makeCluster(). Not used by this function but stored in the model for use in downstream functions. Default: NULL

Details

See ranger documentation for additional details. The formula interface is supported via ranger.arguments, but variable interactions are not permitted. For feature engineering including interactions, see the_feature_engineer().

Value

A ranger model object with additional slots:

- ranger.arguments: Arguments used to fit the model.
- importance: List with global importance data frame (predictors ranked by importance), importance plot, and local importance scores (per-observation difference in accuracy between permuted and non-permuted predictors, based on out-of-bag data).
- performance: Model performance metrics including R-squared (out-of-bag and standard), pseudo R-squared, RMSE, and NRMSE.
- residuals: Model residuals with normality diagnostics (residuals_diagnostics()) and spatial autocorrelation (moran_multithreshold()).

See Also

Other main_models: rf_spatial()

Examples

```
data(
  plants_df,
  plants_response,
  plants_predictors,
  plants_distance
)

m <- rf(
  data = plants_df,
  dependent.variable.name = plants_response,
  predictor.variable.names = plants_predictors,
  distance.matrix = plants_distance,
  distance.thresholds = c(100, 1000, 2000),
  ranger.arguments = list(
    num.trees = 50,
    min.node.size = 20
  ),
```

```
  verbose = FALSE,
  n.cores = 1
)

class(m)
#variable importance
m$importance$per.variable
m$importance$per.variable.plot

#model performance
m$performance

#autocorrelation of residuals
m$residuals$autocorrelation$per.distance
m$residuals$autocorrelation$plot

#model predictions
m$predictions$values

#predictions for new data (using stats::predict)
y <- stats::predict(
  object = m,
  data = plants_df[1:5, ],
  type = "response"
)$predictions

#alternative: pass arguments via ranger.arguments list
args <- list(
  data = plants_df,
  dependent.variable.name = plants_response,
  predictor.variable.names = plants_predictors,
  distance.matrix = plants_distance,
  distance.thresholds = c(100, 1000, 2000),
  num.trees = 50,
  min.node.size = 20,
  num.threads = 1
)

m <- rf(
  ranger.arguments = args,
  verbose = FALSE
)
```

`rf_compare` *Compares models via spatial cross-validation*

Description

Uses `rf_evaluate()` to compare the performance of several models on independent spatial folds via spatial cross-validation.

Usage

```
rf_compare(
  models = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  metrics = c("r.squared", "pseudo.r.squared", "rmse", "nrmse", "auc"),
  distance.step = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
  fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 0.8),
  line.color = "gray30",
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`models`	Named list with models resulting from `rf()`, `rf_spatial()`, `rf_tuning()`, or `rf_evaluate()`. Example: `models = list(a = model.a, b = model.b)`. Default: `NULL`
`xy`	Data frame or matrix with two columns containing coordinates and named "x" and "y". Default: `NULL`
`repetitions`	Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: `30`
`training.fraction`	Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: `0.75`
`metrics`	Character vector, names of the performance metrics selected. The possible values are: "r.squared" (`cor(obs, pred) ^ 2`), "pseudo.r.squared" (`cor(obs, pred)`), "rmse" (`sqrt(sum((obs - pred)^2)/length(obs))`), "nrmse" (`rmse/(quantile(obs, 0.75) - quantile(obs, 0.25))`). Default: `c("r.squared", "pseudo.r.squared", "rmse", "nrmse")`
`distance.step`	Numeric, argument `distance.step` of `thinning_til_n()`. distance step used during the selection of the centers of the training folds. These fold centers are selected by thinning the data until a number of folds equal or lower than `repetitions` is reached. Its default value is 1/1000th the maximum distance within records in `xy`. Reduce it if the number of training folds is lower than expected.
`distance.step.x`	Numeric, argument `distance.step.x` of `make_spatial_folds()`. Distance step used during the growth in the x axis of the buffers defining the training folds. Default: `NULL` (1/1000th the range of the x coordinates).

distance.step.y
: Numeric, argument distance.step.x of make_spatial_folds(). Distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).

fill.color
: Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F", direction = -1)

line.color
: Character string, color of the line produced by ggplot2::geom_smooth(). Default: "gray30"

seed
: Integer, random seed to facilitate reproduciblity. If set to a given number, the results of the function are always the same. Default: 1.

verbose
: Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE

n.cores
: Integer, number of cores to use for parallel execution. Creates a socket cluster with parallel::makeCluster(), runs operations in parallel with foreach and %dopar%, and stops the cluster with parallel::clusterStop() when the job is done. Default: parallel::detectCores() - 1

cluster
: A cluster definition generated with parallel::makeCluster(). If provided, overrides n.cores. When cluster = NULL (default value), and model is provided, the cluster in model, if any, is used instead. If this cluster is NULL, then the function uses n.cores instead. The function does not stop a provided cluster, so it should be stopped with parallel::stopCluster() afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the model argument, or using the %>% pipe. Default: NULL

Value

A list with three slots:

- comparison.df: Data frame with one performance value per spatial fold, metric, and model.
- spatial.folds: List with the indices of the training and testing records for each evaluation repetition.
- plot: Violin-plot of comparison.df.

See Also

rf_evaluate()

Other model_workflow: rf_evaluate(), rf_importance(), rf_repeat(), rf_tuning()

Examples

```
if(interactive()){

  data(
    plants_rf,
    plants_rf_spatial,
```

```
  plants_xy
)

comparison <- rf_compare(
  models = list(
    `Non spatial` = plants_rf,
    Spatial = plants_rf_spatial
  ),
  repetitions = 5,
  xy = plants_xy,
  metrics = "rmse",
  n.cores = 1
)

}
```

`rf_evaluate` *Evaluates random forest models with spatial cross-validation*

Description

Evaluates the performance of random forest on unseen data over independent spatial folds.

Usage

```
rf_evaluate(
  model = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  metrics = c("r.squared", "pseudo.r.squared", "rmse", "nrmse", "auc"),
  distance.step = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
  grow.testing.folds = FALSE,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`model`	Model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.
`xy`	Data frame or matrix with two columns containing coordinates and named "x" and "y". If `NULL`, the function will throw an error. Default: `NULL`

repetitions
: Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: 30

training.fraction
: Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: 0.75

metrics
: Character vector, names of the performance metrics selected. The possible values are: "r.squared" (cor(obs, pred) ^ 2), "pseudo.r.squared" (cor(obs, pred)), "rmse" (sqrt(sum((obs - pred)^2)/length(obs))), "nrmse" (rmse/(quantile(obs, 0.75) - quantile(obs, 0.25))), and "auc" (only for binary responses with values 1 and 0). Default: c("r.squared", "pseudo.r.squared", "rmse", "nrmse")

distance.step
: Numeric, argument distance.step of thinning_til_n(). distance step used during the selection of the centers of the training folds. These fold centers are selected by thinning the data until a number of folds equal or lower than repetitions is reached. Its default value is 1/1000th the maximum distance within records in xy. Reduce it if the number of training folds is lower than expected.

distance.step.x
: Numeric, argument distance.step.x of make_spatial_folds(). Distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).

distance.step.y
: Numeric, argument distance.step.x of make_spatial_folds(). Distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).

grow.testing.folds
: Logic. By default, this function grows contiguous training folds to keep the spatial structure of the data as intact as possible. However, when setting grow.testing.folds = TRUE, the argument training.fraction is set to 1 - training.fraction, and the training and testing folds are switched. This option might be useful when the training data has a spatial structure that does not match well with the default behavior of the function. Default: FALSE

seed
: Integer, random seed to facilitate reproduciblity. If set to a given number, the results of the function are always the same. Default: 1.

verbose
: Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE

n.cores
: Integer, number of cores to use for parallel execution. Creates a socket cluster with parallel::makeCluster(), runs operations in parallel with foreach and %dopar%, and stops the cluster with parallel::clusterStop() when the job is done. Default: parallel::detectCores() - 1

cluster
: A cluster definition generated with parallel::makeCluster(). If provided, overrides n.cores. When cluster = NULL (default value), and model is provided, the cluster in model, if any, is used instead. If this cluster is NULL, then the function uses n.cores instead. The function does not stop a provided cluster, so it should be stopped with parallel::stopCluster() afterwards. The cluster definition is stored in the output list under the name "cluster" so it can

be passed to other functions via the model argument, or using the %>% pipe. Default: NULL

Details

The evaluation algorithm works as follows: the number of repetitions and the input dataset (stored in model$ranger.arguments$data) are used as inputs for the function thinning_til_n(), that applies thinning() to the input data until as many cases as repetitions are left, and as separated as possible. Each of these remaining records will be used as a "fold center". From that point, the fold grows, until a number of points equal (or close) to training.fraction is reached. The indices of the records within the grown spatial fold are stored as "training" in the output list, and the remaining ones as "testing". Then, for each spatial fold, a "training model" is fitted using the cases corresponding with the training indices, and predicted over the cases corresponding with the testing indices. The model predictions on the "unseen" data are compared with the observations, and the performance measures (R squared, pseudo R squared, RMSE and NRMSE) computed.

Value

A model of the class "rf_evaluate" with a new slot named "evaluation", that is a list with the following slots:

- training.fraction: Value of the argument training.fraction.
- spatial.folds: Result of applying make_spatial_folds() on the data coordinates. It is a list with as many slots as repetitions are indicated by the user. Each slot has two slots named "training" and "testing", each one having the indices of the cases used on the training and testing models.
- per.fold: Data frame with the evaluation results per spatial fold (or repetition). It contains the ID of each fold, it's central coordinates, the number of training and testing cases, and the training and testing performance measures: R squared, pseudo R squared (cor(observed, predicted)), rmse, and normalized rmse.
- per.model: Same data as above, but organized per fold and model ("Training", "Testing", and "Full").
- aggregated: Same data, but aggregated by model and performance measure.

See Also

Other model_workflow: rf_compare(), rf_importance(), rf_repeat(), rf_tuning()

Examples

```
if(interactive()){

data(
  plants_rf,
  plants_xy
)

plants_rf <- rf_evaluate(
  model = plants_rf,
```

```
  xy = plants_xy,
  repetitions = 5,
  n.cores = 1
)

plot_evaluation(plants_rf, notch = FALSE)

print_evaluation(plants_rf)

get_evaluation(plants_rf)

}
```

`rf_importance` *Contribution of each predictor to model transferability*

Description

Evaluates the contribution of the predictors to model transferability via spatial cross-validation. The function returns the median increase or decrease in a given evaluation metric (R2, pseudo R2, RMSE, nRMSE, or AUC) when a variable is introduced in a model, by comparing and evaluating via spatial cross-validation models with and without the given variable. This function was devised to provide importance scores that would be less sensitive to spatial autocorrelation than those computed internally by random forest on the out-of-bag data. This function is experimental.

Usage

```
rf_importance(
  model = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  metric = c("r.squared", "pseudo.r.squared", "rmse", "nrmse", "auc"),
  distance.step = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
 fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 1, end = 0.9),
  line.color = "white",
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`model`	Model fitted with `rf()` and/or `rf_spatial()`. The function doesn't work with models fitted with `rf_repeat()`. Default: `NULL`

xy
: Data frame or matrix with two columns containing coordinates and named "x" and "y". If NULL, the function will throw an error. Default: NULL

repetitions
: Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: 30

training.fraction
: Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: 0.75

metric
: Character, nams of the performance metric to use. The possible values are: "r.squared" (cor(obs, pred) ^ 2), "pseudo.r.squared" (cor(obs, pred)), "rmse" (sqrt(sum((obs - pred)^2)/length(obs))), "nrmse" (rmse/(quantile(obs, 0.75) - quantile(obs, 0.25))), and "auc" (only for binary responses with values 1 and 0). Default: "r.squared"

distance.step
: Numeric, argument distance.step of thinning_til_n(). distance step used during the selection of the centers of the training folds. These fold centers are selected by thinning the data until a number of folds equal or lower than repetitions is reached. Its default value is 1/1000th the maximum distance within records in xy. Reduce it if the number of training folds is lower than expected.

distance.step.x
: Numeric, argument distance.step.x of make_spatial_folds(). Distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).

distance.step.y
: Numeric, argument distance.step.x of make_spatial_folds(). Distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).

fill.color
: Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. viridis::viridis(100)). Default: viridis::viridis(100, option = "F", direction = -1, alpha = 0.8, end = 0.9)

line.color
: Character string, color of the line produced by ggplot2::geom_smooth(). Default: "white"

seed
: Integer, random seed to facilitate reproduciblity. If set to a given number, the results of the function are always the same. Default: 1.

verbose
: Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE

n.cores
: Integer, number of cores to use for parallel execution. Creates a socket cluster with parallel::makeCluster(), runs operations in parallel with foreach and %dopar%, and stops the cluster with parallel::clusterStop() when the job is done. Default: parallel::detectCores() - 1

cluster
: A cluster definition generated with parallel::makeCluster(). If provided, overrides n.cores. When cluster = NULL (default value), and model is provided, the cluster in model, if any, is used instead. If this cluster is NULL, then the function uses n.cores instead. The function does not stop a provided cluster, so it should be stopped with parallel::stopCluster() afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the model argument, or using the %>% pipe. Default: NULL

Value

The input model with new data in its "importance" slot. The new importance scores are included in the data frame `model$importance$per.variable`, under the column names "importance.cv" (median contribution to transferability over spatial cross-validation repetitions), "importance.cv.mad" (median absolute deviation of the performance scores over spatial cross-validation repetitions), "importance.cv.percent" ("importance.cv" expressed as a percent, taking the full model's performance as baseline), and "importance.cv.mad" (median absolute deviation of "importance.cv"). The plot is stored as "cv.per.variable.plot".

See Also

Other model_workflow: `rf_compare()`, `rf_evaluate()`, `rf_repeat()`, `rf_tuning()`

Examples

```
if(interactive()){
  data(plants_rf)

  m_importance <- rf_importance(
    model = plants_rf,
    repetitions = 5
  )
}
```

`rf_repeat` *Fits several random forest models on the same data*

Description

Fits several random forest models on the same data in order to capture the effect of the algorithm's stochasticity on the variable importance scores, predictions, residuals, and performance measures. The function relies on the median to aggregate performance and importance values across repetitions. It is recommended to use it after a model is fitted (`rf()` or `rf_spatial()`), tuned (`rf_tuning()`), and/or evaluated (`rf_evaluate()`). This function is designed to be used after fitting a model with `rf()` or `rf_spatial()`, tuning it with `rf_tuning()` and evaluating it with `rf_evaluate()`.

Usage

```
rf_repeat(
  model = NULL,
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
```

```
  xy = NULL,
  ranger.arguments = NULL,
  scaled.importance = FALSE,
  repetitions = 10,
  keep.models = TRUE,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

model	A model fitted with rf(). If provided, the data and ranger arguments are taken directly from the model definition (stored in model$ranger.arguments). Default: NULL
data	Data frame with a response variable and a set of predictors. Default: NULL
dependent.variable.name	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument case.weights of ranger is populated by the function case_weights(). Default: NULL
predictor.variable.names	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL
distance.matrix	Squared matrix with the distances among the records in data. The number of rows of distance.matrix and data must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
distance.thresholds	Numeric vector with neighborhood distances. All distances in the distance matrix below each value in dustance.thresholds are set to 0 for the computation of Moran's I. If NULL, it defaults to seq(0, max(distance.matrix), length.out = 4). Default: NULL
xy	(optional) Data frame or matrix with two columns containing coordinates and named "x" and "y". It is not used by this function, but it is stored in the slot ranger.arguments$xy of the model, so it can be used by rf_evaluate() and rf_tuning(). Default: NULL
ranger.arguments	Named list with ranger arguments (other arguments of this function can also go here). All ranger arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of ranger if you are not familiar with the arguments of this function.
scaled.importance	Logical. If TRUE, and 'importance = "permutation', the function scales 'data' with scale and fits a new model to compute scaled variable importance scores. Default: FALSE

repetitions	Integer, number of random forest models to fit. Default: 10
keep.models	Logical, if TRUE, the fitted models are returned in the models slot. Set to FALSE if the accumulation of models is creating issues with the RAM memory available. Default: TRUE.
seed	Integer, random seed to facilitate reproduciblity. If set to a given number, the results of the function are always the same. Default: 1.
verbose	Logical, ff TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with parallel::makeCluster(), runs operations in parallel with foreach and %dopar%, and stops the cluster with parallel::clusterStop() when the job is done. Default: parallel::detectCores() - 1
cluster	A cluster definition generated with parallel::makeCluster(). If provided, overrides n.cores. When cluster = NULL (default value), and model is provided, the cluster in model, if any, is used instead. If this cluster is NULL, then the function uses n.cores instead. The function does not stop a provided cluster, so it should be stopped with parallel::stopCluster() afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the model argument, or using the %>% pipe. Default: NULL

Value

A ranger model with several new slots:

- ranger.arguments: Stores the values of the arguments used to fit the ranger model.
- importance: A list containing a data frame with the predictors ordered by their importance, a ggplot showing the importance values, and local importance scores.
- performance: out-of-bag performance scores: R squared, pseudo R squared, RMSE, and normalized RMSE (NRMSE).
- pseudo.r.squared: computed as the correlation between the observations and the predictions.
- residuals: residuals, normality test of the residuals computed with residuals_test(), and spatial autocorrelation of the residuals computed with moran_multithreshold().

See Also

Other model_workflow: rf_compare(), rf_evaluate(), rf_importance(), rf_tuning()

Examples

```
if(interactive()){

  data(plants_rf)

  m_repeat <- rf_repeat(
    model = plants_rf,
```

```
  repetitions = 5,
  n.cores = 1
)

#performance scores across repetitions
m_repeat$performance
print_performance(m_repeat)

#variable importance
plot_importance(m_repeat)

#response curves
plot_response_curves(
  model = m_repeat,
  variables = "climate_bio1_average",
  quantiles = 0.5
)

}
```

`rf_spatial` *Fits spatial random forest models*

Description

Fits spatial random forest models using different methods to generate, rank, and select spatial predictors acting as proxies of spatial processes not considered by the non-spatial predictors. The end goal is providing the model with information about the spatial structure of the data to minimize the spatial correlation (Moran's I) of the model residuals and generate honest variable importance scores.

Usage

```
rf_spatial(
  model = NULL,
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  xy = NULL,
  ranger.arguments = NULL,
  scaled.importance = TRUE,
 method = c("mem.moran.sequential", "mem.effect.sequential", "mem.effect.recursive",
    "hengl", "hengl.moran.sequential", "hengl.effect.sequential",
    "hengl.effect.recursive", "pca.moran.sequential", "pca.effect.sequential",
    "pca.effect.recursive"),
 max.spatial.predictors = NULL,
```

```
  weight.r.squared = NULL,
  weight.penalization.n.predictors = NULL,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

model	A model fitted with rf(). If used, the arguments data, dependent.variable.name, predictor.variable.names, distance.matrix, distance.thresholds, ranger.arguments, and scaled.importance are taken directly from the model definition. Default: NULL
data	Data frame with a response variable and a set of predictors. Default: NULL
dependent.variable.name	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument case.weights of ranger is populated by the function case_weights(). Default: NULL
predictor.variable.names	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL
distance.matrix	Squared matrix with the distances among the records in data. The number of rows of distance.matrix and data must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
distance.thresholds	Numeric vector with distances in the same units as distance.matrix Distances below each distance threshold are set to 0 on separated copies of the distance matrix to compute Moran's I at different neighborhood distances. If NULL, it defaults to seq(0, max(distance.matrix)/2, length.out = 4) (defined by default_distance_thresholds()). Default: NULL
xy	(optional) Data frame or matrix with two columns containing coordinates and named "x" and "y". It is not used by this function, but it is stored in the slot ranger.arguments$xy of the model, so it can be used by rf_evaluate() and rf_tuning(). Default: NULL
ranger.arguments	Named list with ranger arguments (other arguments of this function can also go here). All ranger arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of ranger if you are not familiar with the arguments of this function.
scaled.importance	Logical. If TRUE, and 'importance = "permutation', the function scales 'data' with scale and fits a new model to compute scaled variable importance scores. Default: TRUE
method	Character, method to build, rank, and select spatial predictors. One of:

- "hengl"
- "hengl.moran.sequential" (experimental)
- "hengl.effect.sequential" (experimental)
- "hengl.effect.recursive" (experimental)
- "pca.moran.sequential" (experimental)
- "pca.effect.sequential" (experimental)
- "pca.effect.recursive" (experimental)
- "mem.moran.sequential"
- "mem.effect.sequential"
- "mem.effect.recursive"

`max.spatial.predictors`
: Integer, maximum number of spatial predictors to generate. Useful when memory problems arise due to a large number of spatial predictors, Default: `NULL`

`weight.r.squared`
: Numeric between 0 and 1, weight of R-squared in the selection of spatial components. See Details, Default: `NULL`

`weight.penalization.n.predictors`
: Numeric between 0 and 1, weight of the penalization for adding an increasing number of spatial predictors during selection. Default: `NULL`

`seed`
: Integer, random seed to facilitate reproducibility. Default: `1`.

`verbose`
: Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: `TRUE`

`n.cores`
: Integer, number of cores to use for parallel execution. Creates a socket cluster with `parallel::makeCluster()`, runs operations in parallel with `foreach` and `%dopar%`, and stops the cluster with `parallel::clusterStop()` when the job is done. Default: `parallel::detectCores() - 1`

`cluster`
: A cluster definition generated with `parallel::makeCluster()`. If provided, overrides `n.cores`. When `cluster = NULL` (default value), and `model` is provided, the cluster in `model`, if any, is used instead. If this cluster is `NULL`, then the function uses `n.cores` instead. The function does not stop a provided cluster, so it should be stopped with `parallel::stopCluster()` afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the `model` argument, or using the `%>%` pipe. Default: `NULL`

Details

The function uses three different methods to generate spatial predictors ("hengl", "pca", and "mem"), two methods to rank them in order to define in what order they are introduced in the model ("effect" and "moran), and two methods to select the spatial predictors that minimize the spatial correlation of the model residuals ("sequential" and "recursive"). All method names but "hengl" (that uses the complete distance matrix as predictors in the spatial model) are named by combining a method to generate the spatial predictors, a method to rank them, and a method to select them, separated by a point. Examples are "mem.moran.sequential" or "mem.effect.recursive". All combinations are not possible, since the ranking method "moran" cannot be used with the selection method "recursive" (because the logics behind them are very different, see below). Methods to generate spatial predictors:

- "hengl": named after the method RFsp presented in the paper "Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables", by Hengl et al. (2018), where the authors propose to use the distance matrix among records as predictors in spatial random forest models (RFsp method). In this function, all methods starting with "hengl" use either the complete distance matrix, or select columns of the distance matrix as spatial predictors.
- "mem": Generates Moran's Eigenvector Maps, that is, the eigenvectors of the double-centered weights of the distance matrix. The method is described in "Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM)", by Dray et al. (2006), and "Statistical methods for temporal and space–time analysis of community composition data", by Legendre and Gauthier (2014).
- "pca": Computes spatial predictors from the principal component analysis of a weighted distance matrix (see weights_from_distance_matrix()). This is an experimental method, use with caution.

Methods to rank spatial predictors (see rank_spatial_predictors()):

- "moran": Computes the Moran's I of each spatial predictor, selects the ones with positive values, and ranks them from higher to lower Moran's I.
- "effect": If a given non-spatial random forest model is defined as y = p1 + ... + pn, being p1 + ... + pn the set of predictors, for every spatial predictor generated (spX) a spatial model y = p1 + ... + pn + spX is fitted, and the Moran's I of its residuals is computed. The spatial predictors are then ranked by how much they help to reduce spatial autocorrelation between the non-spatial and the spatial model.

Methods to select spatial predictors:

- "sequential" (see select_spatial_predictors_sequential()): The spatial predictors are added one by one in the order they were ranked, and once all spatial predictors are introduced, the best first n predictors are selected. This method is similar to the one employed in the MEM methodology (Moran's Eigenvector Maps) described in the paper "Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM)", by Dray et al. (2006), and "Statistical methods for temporal and space–time analysis of community composition data", by Legendre and Gauthier (2014). This method generally introduces tens of predictors into the model, but usually offers good results.
- "recursive" (see select_spatial_predictors_recursive()): This method tries to find the smallest combination of spatial predictors that reduce the spatial correlation of the model's residuals the most. The algorithm goes as follows: 1. The first ranked spatial predictor is introduced into the model; 2. the remaining predictors are ranked again using the "effect" method, using the model in 1. as reference. The first spatial predictor in the resulting ranking is then introduced into the model, and the steps 1. and 2. are repeated until spatial predictors stop having an effect in reducing the Moran's I of the model residuals. This method takes longer to compute, but generates smaller sets of spatial predictors. This is an experimental method, use with caution.

Once ranking procedure is completed, an algorithm is used to select the minimal subset of spatial predictors that reduce the most the Moran's I of the residuals: for each new spatial predictor introduced in the model, the Moran's I of the residuals, it's p-value, a binary version of the p-value (0 if <

0.05 and 1 if >= 0.05), the R-squared of the model, and a penalization linear with the number of spatial predictors introduced (computed as `(1 / total spatial predictors) * introduced spatial predictors`) are rescaled between 0 and 1. Then, the optimization criteria is computed as `max(1 - Moran's I, p-value binary) + (weig`. The predictors from the first one to the one with the highest optimization criteria are then selected as the best ones in reducing the spatial correlation of the model residuals, and used along with `data` to fit the final spatial model.

Value

A ranger model with several new slots:

- `ranger.arguments`: Values of the arguments used to fit the ranger model.
- `importance`: A list containing the vector of variable importance as originally returned by ranger (scaled or not depending on the value of 'scaled.importance'), a data frame with the predictors ordered by their importance, and a ggplot showing the importance values.
- `performance`: With the out-of-bag R squared, pseudo R squared, RMSE and NRMSE of the model.
- `residuals`: residuals, normality test of the residuals computed with `residuals_test()`, and spatial autocorrelation of the residuals computed with `moran_multithreshold()`.
- `spatial`: A list with four slots:
  - `method`: Character, method used to generate, rank, and select spatial predictors.
  - `names`: Character vector with the names of the selected spatial predictors. Not returned if the method is "hengl".
  - `optimization`: Criteria used to select the spatial predictors. Not returned if the method is "hengl".
  - `plot`: Plot of the criteria used to select the spatial predictors. Not returned if the method is "hengl".

See Also

Other main_models: `rf()`

Examples

```
if (interactive()) {
  data(
    plants_df,
    plants_response,
    plants_predictors,
    plants_distance,
    plants_rf
  )

  #subset to speed up example
  idx <- 1:100
  plants_df <- plants_df[idx, ]
  plants_distance <- plants_distance[idx, idx]

  #fit spatial model from scratch
```

```
  m_spatial <- rf_spatial(
    data = plants_df,
    dependent.variable.name = plants_response,
    predictor.variable.names = plants_predictors,
    distance.matrix = plants_distance,
    distance.thresholds = c(100, 1000, 2000),
    method = "mem.moran.sequential",
    ranger.arguments = list(num.trees = 30),
    n.cores = 1
  )

  plot_residuals_diagnostics(m_spatial)

  #optimization of MEM selection
  plot_optimization(m_spatial)

  #from non-spatial to spatial model
  m_spatial <- rf_spatial(
    model = plants_rf
    )

}
```

`rf_tuning` *Tuning of random forest hyperparameters via spatial cross-validation*

Description

Finds the optimal set of random forest hyperparameters `num.trees`, `mtry`, and `min.node.size` via grid search by maximizing the model's R squared, or AUC, if the response variable is binomial, via spatial cross-validation performed with `rf_evaluate()`.

Usage

```
rf_tuning(
  model = NULL,
  num.trees = NULL,
  mtry = NULL,
  min.node.size = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

model	A model fitted with rf(). If provided, the training data is taken directly from the model definition (stored in model$ranger.arguments). Default: NULL
num.trees	Numeric integer vector with the number of trees to fit on each model repetition. Default: c(500, 1000, 2000).
mtry	Numeric integer vector, number of predictors to randomly select from the complete pool of predictors on each tree split. Default: floor(seq(1, length(predictor.variable.names length.out = 4))
min.node.size	Numeric integer, minimal number of cases in a terminal node. Default: c(5, 10, 20, 40)
xy	Data frame or matrix with two columns containing coordinates and named "x" and "y". If NULL, the function will throw an error. Default: NULL
repetitions	Integer, number of independent spatial folds to use during the cross-validation. Default: 30.
training.fraction	Proportion between 0.2 and 0.9 indicating the number of records to be used in model training. Default: 0.75
seed	Integer, random seed to facilitate reproduciblity. If set to a given number, the results of the function are always the same. Default: 1.
verbose	Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with parallel::makeCluster(), runs operations in parallel with foreach and %dopar%, and stops the cluster with parallel::clusterStop() when the job is done. Default: parallel::detectCores() - 1
cluster	A cluster definition generated with parallel::makeCluster(). If provided, overrides n.cores. When cluster = NULL (default value), and model is provided, the cluster in model, if any, is used instead. If this cluster is NULL, then the function uses n.cores instead. The function does not stop a provided cluster, so it should be stopped with parallel::stopCluster() afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the model argument, or using the %>% pipe. Default: NULL

Value

A model with a new slot named tuning, with a data frame with the results of the tuning analysis.

See Also

rf_evaluate()

Other model_workflow: rf_compare(), rf_evaluate(), rf_importance(), rf_repeat()

Examples

```
if(interactive()){
  data(
    plants_rf,
    plants_xy
  )

  plants_rf_tuned <- rf_tuning(
    model = plants_rf,
    num.trees = c(25, 50),
    mtry = c(5, 10),
    min.node.size = c(10, 20),
    xy = plants_xy,
    repetitions = 5,
    n.cores = 1
  )

  plot_tuning(plants_rf_tuned)
}
```

---

root_mean_squared_error

RMSE and normalized RMSE

Description

Computes the rmse or normalized rmse (nrmse) between two numeric vectors of the same length representing observations and model predictions.

Usage

```
root_mean_squared_error(
  o,
  p,
  normalization = c("rmse", "all", "mean", "sd", "maxmin", "iq")
)
```

Arguments

o	Numeric vector with observations, must have the same length as p.
p	Numeric vector with predictions, must have the same length as o.
normalization	character, normalization method, Default: "rmse" (see Details).

Details

The normalization methods go as follows:

- `"rmse"`: RMSE with no normalization.
- `"mean"`: RMSE dividied by the mean of the observations (rmse/mean(o)).
- `"sd"`: RMSE dividied by the standard deviation of the observations (rmse/sd(o)).
- `"maxmin"`: RMSE divided by the range of the observations (rmse/(max(o) - min(o))).
- `"iq"`: RMSE divided by the interquartile range of the observations (rmse/(quantile(o, 0.75) - quantile(o, 0.25)))

Value

Named numeric vector with either one or 5 values, as selected by the user.

See Also

Other utilities: `.vif_to_df()`, `auc()`, `beowulf_cluster()`, `objects_size()`, `optimization_function()`, `prepare_importance_spatial()`, `rescale_vector()`, `setup_parallel_execution()`, `standard_error()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

Examples

```
root_mean_squared_error(
  o = runif(10),
  p = runif(10)
)
```

---

select_spatial_predictors_recursive

Finds optimal combinations of spatial predictors

Description

Selects spatial predictors following these steps:

1. Gets the spatial predictors ranked by `rank_spatial_predictors()` and fits a model of the form `y ~ predictors + best_spatial_predictor_1`. The Moran's I of the residuals of this model is used as reference value for the next step.
2. The remaining spatial predictors are introduced again into `rank_spatial_predictors()`, and the spatial predictor with the highest ranking is introduced in a new model of the form `y ~ predictors + best_spatial_predictor_1 + best_spatial_predictor_2`.
3. Steps 1 and 2 are repeated until the Moran's I doesn't improve for a number of repetitions equal to the 20 percent of the total number of spatial predictors introduced in the function.

This method allows to select the smallest set of spatial predictors that have the largest joint effect in reducing the spatial correlation of the model residuals, while maintaining the model's R-squared as high as possible. As a consequence of running `rank_spatial_predictors()` on each iteration, this method includes less spatial predictors in the final model than the sequential method implemented in `select_spatial_predictors_sequential()` would do, while minimizing spatial correlation and maximizing the R squared of the model as much as possible.

Usage

```
select_spatial_predictors_recursive(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  ranger.arguments = NULL,
  spatial.predictors.df = NULL,
  spatial.predictors.ranking = NULL,
  weight.r.squared = 0.25,
  weight.penalization.n.predictors = 0,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`data` Data frame with a response variable and a set of predictors. Default: `NULL`

`dependent.variable.name`
Character string with the name of the response variable. Must be in the column names of `data`. Default: `NULL`

`predictor.variable.names`
Character vector with the names of the predictive variables. Every element of this vector must be in the column names of `data`. Default: `NULL`

`distance.matrix`
Squared matrix with the distances among the records in `data`. The number of rows of `distance.matrix` and `data` must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: `NULL`

`distance.thresholds`
Numeric vector with neighborhood distances. All distances in the distance matrix below each value in `dustance.thresholds` are set to 0 for the computation of Moran's I. If `NULL`, it defaults to seq(0, max(distance.matrix), length.out = 4). Default: `NULL`

`ranger.arguments`
Named list with ranger arguments (other arguments of this function can also go here). All ranger arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of ranger if you are not familiar with the arguments of this function.

`spatial.predictors.df`
Data frame of spatial predictors.

`spatial.predictors.ranking`
: Ranking of predictors returned by `rank_spatial_predictors()`.

`weight.r.squared`
: Numeric between 0 and 1, weight of R-squared in the optimization index. Default: `0.25`

`weight.penalization.n.predictors`
: Numeric between 0 and 1, weight of the penalization for the number of spatial predictors added in the optimization index. Default: `0`

`n.cores`
: Integer, number of cores to use. Default: `parallel::detectCores() - 1`

`cluster`
: A cluster definition generated by `parallel::makeCluster()`. Default: `NULL`

Details

The algorithm works as follows. If the function `rank_spatial_predictors()` returns 10 ranked spatial predictors (sp1 to sp10, being sp7 the best one), `select_spatial_predictors_recursive()` is going to first fit the model `y ~ predictors + sp7`. Then, the spatial predictors sp2 to sp9 are again ranked with `rank_spatial_predictors()` using the model `y ~ predictors + sp7` as reference (at this stage, some of the spatial predictors might be dropped due to lack of effect). When the new ranking of spatial predictors is ready (let's say they are sp5, sp3, and sp4), the best one (sp5) is included in the model `y ~ predictors + sp7 + sp5`, and the remaining ones go again to `rank_spatial_predictors()` to repeat the process until spatial predictors are depleted.

Value

A list with two slots: `optimization`, a data frame with the index of the spatial predictor added on each iteration, the spatial correlation of the model residuals, and the R-squared of the model, and `best.spatial.predictors`, that is a character vector with the names of the spatial predictors that minimize the Moran's I of the residuals and maximize the R-squared of the model.

See Also

Other spatial_analysis: `filter_spatial_predictors()`, `mem()`, `mem_multithreshold()`, `moran()`, `moran_multithreshold()`, `pca()`, `pca_multithreshold()`, `rank_spatial_predictors()`, `residuals_diagnostics()`, `residuals_test()`, `select_spatial_predictors_sequential()`

Examples

```
if (interactive()) {
  data(
    plants_df,
    plants_response,
    plants_predictors,
    plants_distance,
    plants_rf
  )

  #subset to speed up example
  idx <- 1:20
  plants_df <- plants_df[idx, ]
  plants_distance <- plants_distance[idx, idx]
```

```
#generate spatial predictors
mems <- mem_multithreshold(
  distance.matrix = plants_distance,
  distance.thresholds = 100
)

#rank them from higher to lower moran
mems.rank <- rank_spatial_predictors(
  ranking.method = "moran",
  spatial.predictors.df = mems,
  reference.moran.i = plants_rf$residuals$autocorrelation$max.moran,
  distance.matrix = plants_distance,
  distance.thresholds = 100,
  n.cores = 1
)

#select best subset via sequential addition
selection <- select_spatial_predictors_recursive(
  data = plants_df,
  dependent.variable.name = plants_response,
  predictor.variable.names = plants_predictors,
  distance.matrix = plants_distance,
  distance.thresholds = 0,
  spatial.predictors.df = mems,
  spatial.predictors.ranking = mems.rank,
  ranger.arguments = list(num.trees = 30),
  n.cores = 1
)

#names of selected spatial predictors
selection$best.spatial.predictors

#optimization plot
plot_optimization(selection$optimization)
}
```

`select_spatial_predictors_sequential` *Sequential introduction of spatial predictors into a model*

Description

Selects spatial predictors by adding them sequentially into a model while monitoring the Moran's I of the model residuals and the model's R-squared. Once all the available spatial predictors have been added to the model, the function identifies the first `n` predictors that minimize the spatial correlation of the residuals and maximize R-squared, and returns the names of the selected spatial predictors and a data frame with the selection criteria.

Usage

```
select_spatial_predictors_sequential(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  ranger.arguments = NULL,
  spatial.predictors.df = NULL,
  spatial.predictors.ranking = NULL,
  weight.r.squared = 0.75,
  weight.penalization.n.predictors = 0.25,
  verbose = FALSE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`data`	Data frame with a response variable and a set of predictors. Default: `NULL`
`dependent.variable.name`	Character string with the name of the response variable. Must be in the column names of `data`. Default: `NULL`
`predictor.variable.names`	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of `data`. Default: `NULL`
`distance.matrix`	Squared matrix with the distances among the records in `data`. The number of rows of `distance.matrix` and `data` must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: `NULL`
`distance.thresholds`	Numeric vector with neighborhood distances. All distances in the distance matrix below each value in `dustance.thresholds` are set to 0 for the computation of Moran's I. If `NULL`, it defaults to seq(0, max(distance.matrix), length.out = 4). Default: `NULL`
`ranger.arguments`	Named list with ranger arguments (other arguments of this function can also go here). All ranger arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of ranger if you are not familiar with the arguments of this function.
`spatial.predictors.df`	Data frame of spatial predictors.
`spatial.predictors.ranking`	Ranking of the spatial predictors returned by `rank_spatial_predictors()`.
`weight.r.squared`	Numeric between 0 and 1, weight of R-squared in the optimization index. Default: `0.75`

weight.penalization.n.predictors
: Numeric between 0 and 1, weight of the penalization for the number of spatial predictors added in the optimization index. Default: 0.25

verbose
: Logical, ff TRUE, messages and plots generated during the execution of the function are displayed, Default: FALSE

n.cores
: Integer, number of cores to use. Default: parallel::detectCores() - 1

cluster
: A cluster definition generated by parallel::makeCluster(). Default: NULL

Details

The algorithm works as follows: If the function rank_spatial_predictors returns 10 spatial predictors (sp1 to sp10, ordered from best to worst), select_spatial_predictors_sequential is going to fit the models y ~ predictors + sp1, y ~ predictors + sp1 + sp2, until all spatial predictors are used in y ~ predictors + sp1 ... sp10. The model with lower Moran's I of the residuals and higher R-squared (computed on the out-of-bag data) is selected, and its spatial predictors returned.

Value

A list with two slots: optimization, a data frame with the index of the spatial predictor added on each iteration, the spatial correlation of the model residuals, and the R-squared of the model, and best.spatial.predictors, that is a character vector with the names of the spatial predictors that minimize the Moran's I of the residuals and maximize the R-squared of the model.

See Also

Other spatial_analysis: filter_spatial_predictors(), mem(), mem_multithreshold(), moran(), moran_multithreshold(), pca(), pca_multithreshold(), rank_spatial_predictors(), residuals_diagnostics(), residuals_test(), select_spatial_predictors_recursive()

Examples

```
if(interactive()){
data(
  plants_df,
  plants_response,
  plants_predictors,
  plants_distance,
  plants_rf
)

#subset to speed up example
idx <- 1:20
plants_df <- plants_df[idx, ]
plants_distance <- plants_distance[idx, idx]

#generate spatial predictors
mems <- mem_multithreshold(
  distance.matrix = plants_distance,
  distance.thresholds = 100
)
```

```
#rank them from higher to lower moran
mems.rank <- rank_spatial_predictors(
  ranking.method = "moran",
  spatial.predictors.df = mems,
  reference.moran.i = plants_rf$residuals$autocorrelation$max.moran,
  distance.matrix = plants_distance,
  distance.thresholds = 100,
  n.cores = 1
)

#select best subset via sequential addition
selection <- select_spatial_predictors_sequential(
  data = plants_df,
  dependent.variable.name = plants_response,
  predictor.variable.names = plants_predictors,
  distance.matrix = plants_distance,
  distance.thresholds = 0,
  spatial.predictors.df = mems,
  spatial.predictors.ranking = mems.rank,
  ranger.arguments = list(num.trees = 30),
  n.cores = 1
)

#names of selected spatial predictors
selection$best.spatial.predictors

#optimization plot
plot_optimization(selection$optimization)
}
```

`setup_parallel_execution` *Setup parallel execution with automatic backend detection*

Description

Internal helper to manage parallel backend setup with support for user-managed backends, external clusters, and internal clusters.

Usage

```
setup_parallel_execution(cluster = NULL, n.cores = parallel::detectCores() - 1)
```

Arguments

`cluster`	A cluster object from parallel::makeCluster(), or NULL
`n.cores`	Number of cores for internal cluster creation

Value

A list with:

- cluster: The cluster object to pass to child functions (or NULL)
- mode: One of "user_backend", "external_cluster", "internal_cluster", "sequential"
- cleanup: A function to call in on.exit() for proper cleanup

See Also

Other utilities: `.vif_to_df()`, `auc()`, `beowulf_cluster()`, `objects_size()`, `optimization_function()`, `prepare_importance_spatial()`, `rescale_vector()`, `root_mean_squared_error()`, `standard_error()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

`standard_error` *Standard error of the mean of a numeric vector*

Description

Computes the standard error of the mean of a numeric vector as `round(sqrt(var(x)/length(x)), 3)`

Usage

```
standard_error(x)
```

Arguments

`x`	A numeric vector.

Details

The function removes `NA` values before computing the standard error, and rounds the result to 3 decimal places.

Value

A numeric value.

See Also

Other utilities: `.vif_to_df()`, `auc()`, `beowulf_cluster()`, `objects_size()`, `optimization_function()`, `prepare_importance_spatial()`, `rescale_vector()`, `root_mean_squared_error()`, `setup_parallel_execution()`, `statistical_mode()`, `thinning()`, `thinning_til_n()`

Examples

```
standard_error(x = runif(10))
```

statistical_mode *Statistical mode of a vector*

Description

Computes the mode of a numeric or character vector

Usage

```
statistical_mode(x)
```

Arguments

x	Numeric or character vector.

Value

Statistical mode of x.

See Also

Other utilities: .vif_to_df(), auc(), beowulf_cluster(), objects_size(), optimization_function(), prepare_importance_spatial(), rescale_vector(), root_mean_squared_error(), setup_parallel_execution(), standard_error(), thinning(), thinning_til_n()

Examples

```
statistical_mode(x = c(10, 9, 10, 8))
```

the_feature_engineer *Suggest variable interactions and composite features for random forest models*

Description

Suggests candidate variable interactions and composite features able to improve predictive accuracy over data not used to train the model via spatial cross-validation with rf_evaluate(). For a pair of predictors a and b, interactions are build via multiplication (a * b), while composite features are built by extracting the first factor of a principal component analysis performed with pca(), after rescaling a and b between 1 and 100. Interactions and composite features are named a..x..b and a..pca..b respectively.

Candidate variables a and b are selected from those predictors in predictor.variable.names with a variable importance above importance.threshold (set by default to the median of the importance scores).

For each interaction and composite feature, a model including all the predictors plus the interaction or composite feature is fitted, and it's R squared (or AUC if the response is binary) computed via spatial cross-validation (see `rf_evaluate()`) is compared with the R squared of the model without interactions or composite features.

From all the potential interactions screened, only those with a positive increase in R squared (or AUC when the response is binomial) of the model, a variable importance above the median, and a maximum correlation among themselves and with the predictors in `predictor.variable.names` not higher than `cor.threshold` (set to 0.5 by default) are selected. Such a restrictive set of rules ensures that the selected interactions can be used right away for modeling purposes without increasing model complexity unnecessarily. However, the suggested variable interactions might not make sense from a domain expertise standpoint, so please, examine them with care.

The function returns the criteria used to select the interactions, and the data required to use these interactions a model.

Usage

```
the_feature_engineer(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  xy = NULL,
  ranger.arguments = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  importance.threshold = 0.75,
  cor.threshold = 0.75,
  point.color = viridis::viridis(100, option = "F", alpha = 0.8),
  seed = NULL,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

Arguments

`data`	Data frame with a response variable and a set of predictors. Default: `NULL`
`dependent.variable.name`	Character string with the name of the response variable. Must be in the column names of `data`. If the dependent variable is binary with values 1 and 0, the argument `case.weights` of `ranger` is populated by the function `case_weights()`. Default: `NULL`
`predictor.variable.names`	Character vector with the names of the predictive variables, or object of class `"variable_selection"` produced by `auto_vif()` and/or `auto_cor()`. Every element of this vector must be in the column names of `data`. Default: `NULL`
`xy`	Data frame or matrix with two columns containing coordinates and named "x" and "y". If not provided, the comparison between models with and without variable interactions is not done.

`ranger.arguments`
: Named list with ranger arguments (other arguments of this function can also go here). All ranger arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of ranger if you are not familiar with the arguments of this function.

`repetitions`
: Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: `30`

`training.fraction`
: Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: `0.75`

`importance.threshold`
: Numeric between 0 and 1, quantile of variable importance scores over which to select individual predictors to explore interactions among them. Larger values reduce the number of potential interactions explored. Default: `0.75`

`cor.threshold`
: Numeric, maximum Pearson correlation between any pair of the selected interactions, and between any interaction and the predictors in `predictor.variable.names`. Default: `0.75`

`point.color`
: Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. `viridis::viridis(100)`). Default: `viridis::viridis(100, option = "F", alpha = 0.8)`

`seed`
: Integer, random seed to facilitate reproduciblity. If set to a given number, the results of the function are always the same. Default: `NULL`

`verbose`
: Logical. If `TRUE`, messages and plots generated during the execution of the function are displayed. Default: `TRUE`

`n.cores`
: Integer, number of cores to use for parallel execution. Creates a socket cluster with `parallel::makeCluster()`, runs operations in parallel with `foreach` and `%dopar%`, and stops the cluster with `parallel::clusterStop()` when the job is done. Default: `parallel::detectCores() - 1`

`cluster`
: A cluster definition generated with `parallel::makeCluster()`. If provided, overrides `n.cores`. When `cluster = NULL` (default value), and `model` is provided, the cluster in `model`, if any, is used instead. If this cluster is `NULL`, then the function uses `n.cores` instead. The function does not stop a provided cluster, so it should be stopped with `parallel::stopCluster()` afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the `model` argument, or using the `%>%` pipe. Default: `NULL`

Value

A list with seven slots:

- `screening`: Data frame with selection scores of all the interactions considered.
- `selected`: Data frame with selection scores of the selected interactions.
- `df`: Data frame with the computed interactions.

- `plot`: List of plots of the selected interactions versus the response variable. The output list can be plotted all at once with `patchwork::wrap_plots(p)` or `cowplot::plot_grid(plotlist = p)`, or one by one by extracting each plot from the list.
- `data`: Data frame with the response variable, the predictors, and the selected interactions, ready to be used as `data` argument in the package functions.
- `dependent.variable.name`: Character, name of the response.
- `predictor.variable.names`: Character vector with the names of the predictors and the selected interactions.

See Also

Other preprocessing: `auto_cor()`, `auto_vif()`, `case_weights()`, `default_distance_thresholds()`, `double_center_distance_matrix()`, `is_binary()`, `make_spatial_fold()`, `make_spatial_folds()`, `weights_from_distance_matrix()`

Examples

```
if (interactive()) {
  data(
    plants_df,
    plants_response,
    plants_predictors,
    plants_xy,
    plants_rf
  )

  #get five most important predictors from plants_rf to speed-up example
  predictors <- get_importance(plants_rf)[1:5, "variable"]

  #subset to speed-up example
  idx <- 1:30
  plants_df <- plants_df[idx, ]
  plants_xy <- plants_xy[idx, ]

  #data subsetted to speed-up example runtime
  y <- the_feature_engineer(
    data = plants_df,
    dependent.variable.name = plants_response,
    predictor.variable.names = predictors,
    xy = plants_xy,
    repetitions = 5,
    n.cores = 1,
    ranger.arguments = list(
      num.trees = 30
    ),
    verbose = TRUE
  )

  #all tested interactions
  y$screening
```

```
  #selected interaction (same as above in this case)
  y$selected

  #new column added to data
  head(y$data[, y$selected$interaction.name])
}
```

---

thinning *Applies thinning to pairs of coordinates*

Description

Resamples a set of points with x and y coordinates to impose a minimum distance among nearby points.

Usage

```
thinning(xy, minimum.distance = NULL)
```

Arguments

xy	A data frame with columns named "x" and "y" representing geographic coordinates.
minimum.distance	Numeric, minimum distance to be set between nearby points, in the same units as the coordinates of xy.

Details

Generally used to remove redundant points that could produce pseudo-replication, and to limit sampling bias by disaggregating clusters of points.

Value

A data frame with the same columns as xy with points separated by the defined minimum distance.

See Also

thinning_til_n()

Other utilities: .vif_to_df(), auc(), beowulf_cluster(), objects_size(), optimization_function(), prepare_importance_spatial(), rescale_vector(), root_mean_squared_error(), setup_parallel_execution(), standard_error(), statistical_mode(), thinning_til_n()

Examples

```
data(plants_xy)

y <- thinning(
  xy = plants_xy,
  minimum.distance = 10
)

if (interactive()) {
  plot(
    plants_xy[, c("x", "y")],
    col = "blue",
    pch = 15
  )

  points(
    y[, c("x", "y")],
    col = "red",
    pch = 15
  )
}
```

`thinning_til_n` *Applies thinning to pairs of coordinates until reaching a given n*

Description

Resamples a set of points with x and y coordinates by increasing the distance step by step until a given sample size is obtained.

Usage

```
thinning_til_n(
  xy,
  n = 30,
  distance.step = NULL
)
```

Arguments

`xy`	A data frame with columns named "x" and "y" representing geographic coordinates. Default: `NULL`
`n`	Integer, number of samples to obtain. Must be lower than `nrow(xy)`. Default: `30`
`distance.step`	Numeric, distance step used during the thinning iterations. If `NULL`, the one percent of the maximum distance among points in `xy` is used. Default: `NULL`

Value

A data frame with the same columns as xy with a row number close to n.

See Also

thinning()

Other utilities: .vif_to_df(), auc(), beowulf_cluster(), objects_size(), optimization_function(), prepare_importance_spatial(), rescale_vector(), root_mean_squared_error(), setup_parallel_execution(), standard_error(), statistical_mode(), thinning()

Examples

```
data(plants_xy)

y <- thinning_til_n(
  xy = plants_xy,
  n = 10
)

if (interactive()) {
  plot(
    plants_xy[, c("x", "y")],
    col = "blue",
    pch = 15
  )

  points(
    y[, c("x", "y")],
    col = "red",
    pch = 15,
    cex = 1.5
  )
}
```

---

weights_from_distance_matrix

Transforms a distance matrix into a matrix of weights

Description

Transforms a distance matrix into weights (1/distance.matrix) normalized by the row sums. Used to compute Moran's I values and Moran's Eigenvector Maps. Allows to apply a threshold to the distance matrix before computing the weights.

Usage

```
weights_from_distance_matrix(
  distance.matrix = NULL,
  distance.threshold = 0
)
```

Arguments

`distance.matrix`
: Distance matrix. Default: `NULL`.

`distance.threshold`
: Numeric, positive, in the range of values of `distance.matrix`. Distances below this value in the distance matrix are set to 0., Default: `0`.

Value

A weighted distance matrix.

See Also

Other preprocessing: `auto_cor()`, `auto_vif()`, `case_weights()`, `default_distance_thresholds()`, `double_center_distance_matrix()`, `is_binary()`, `make_spatial_fold()`, `make_spatial_folds()`, `the_feature_engineer()`

Examples

```
data(plants_distance)

y <- weights_from_distance_matrix(
  distance.matrix = plants_distance
)

y[1:5, 1:5]
```

%>% *Pipe operator*

Description

See `magrittr::%>%` for details.

Usage

```
lhs %>% rhs
```

Arguments

lhs	A value or the magrittr placeholder.
rhs	A function call using the magrittr semantics.

Value

The result of calling rhs(lhs).

Index