Package 'nerfplus'

November 3, 2026

Title Network-Assisted Random Forest+ (NeRF+)

Version 1.0.0

Description Network-assisted Random Forest+ (NeRF+) is a flexible and interpretable machine learning model for incorporating network data alongside node-level covariate information. Briefly, NeRF+ extends a generalization of random forests (RF) called RF+ to the network-assisted regression setting by incorporating both a network cohesion penalty and network embeddings as additional covariates. Using this approach, NeRF+ inherits both the flexibility and interpretability of RFs while allowing researchers to easily incorporate network information in their model to further improve predictive performance. This package provides tools to fit and interpret NeRF+ models, as detailed in Tang, Levina, and Zhu (2025) <doi:10.48550/arXiv.2509.15611>.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Suggests DT, fresh, ggplot2, ggraph, htmltools, igraph, kableExtra, knitr, patchwork, plotly, shiny, shinycssloaders, shinydashboard, shinydashboardPlus, shinyjs, shinyWidgets, testthat (>= 3.0.0), tippy

Config/testthat/edition 3

Imports abind, caret, dplyr, furrr, glmnet, irlba, Matrix, purrr, ranger, Rcpp, rlang, stringr, tibble, tidyr, tidyselect, yardstick

LinkingTo Rcpp, RcppArmadillo

LazyData true

URL https://tiffanymtang.github.io/nerfplus/

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Tiffany Tang [aut, cre] (ORCID: <https://orcid.org/0000-0002-8079-6867>)

Maintainer Tiffany Tang <ttang4@nd.edu>

Repository CRAN

Date/Publication 2026-11-03 14:40:08 UTC

Contents

example_data . . . 2
fit_network_embedding . . . 3
get_feature_importances . . . 4
get_loo . . . 5
interpret_nerfplus . . . 7
nerfplus . . . 9
nerfplus_conformal . . . 12
nerfplus_cv . . . 13
oos_network_embedding . . . 17
predict.nerfplus . . . 18
predict.rnc . . . 19
rnc . . . 20
rnc_cv . . . 22
run_app . . . 24

Index **26**

`example_data` *Example dataset for NeRF+ demonstration*

Description

Example dataset for NeRF+ demonstration

Usage

```
example_data
```

Format

`example_data`**:**

A list with the following named components:

x Training covariate data matrix with 80 rows and 10 columns.

xtest Test covariate data matrix with 40 rows and 10 columns.

y Training response vector with 80 elements.

ytest Test response vector with 40 elements.

A Training adjacency matrix with 80 rows and 80 columns.

A_full Full adjacency matrix with 120 rows and 120 columns.

The covariate data matrix was generated from a standard normal distribution. The adjacency matrix was generated from a stochastic block model with 3 blocks. The responses were generated from a linear model with an additive network block effect and a main covariate effect from the first covariate only.

`fit_network_embedding` *Wrapper to fit various network embedding methods*

Description

Wrapper to fit various network embedding methods

Usage

```
fit_network_embedding(
  A,
  ndim = 2,
  method = c("adjacency", "laplacian", "score"),
  regularization = 0.5,
  varimax_rotation = FALSE
)
```

Arguments

`A`	Adjacency matrix.
`ndim`	Number of dimensions in embedding.
`method`	Embedding type(s), at least one of "adjacency", "laplacian", or score".
`regularization`	Regularization parameter for the adjacency matrix.
`varimax_rotation`	Whether to apply varimax rotation to the embedding.

Value

A list containing the following components:

- `X`: A list of matrices, each corresponding to a method in `method`, where each matrix has `ndim` columns representing the embedding.
- `eval_ls`: A list of eigenvalues corresponding to each method in `method`.
- `method`: The method(s) used for embedding.
- `regularization`: The regularization parameter used for the adjacency matrix.

Examples

```
data(example_data)
embedding_fit <- fit_network_embedding(
  example_data$A, ndim = 2, method = "laplacian"
)
```

`get_feature_importances` *Compute feature importances for a NeRF+ model*

Description

This function computes global and local feature importances given a fitted NeRF+ model.

Usage

```
get_feature_importances(
  object,
  x,
  x_embed = NULL,
  y = NULL,
  A_full,
  nodeids = NULL,
  method = c("permute", "mdi+", "local"),
  B = 10,
  metric = NULL
)
```

Arguments

`object`	A fitted NeRF+ model object.
`x`	A data frame or matrix used to compute the feature importances.
`x_embed`	An optional data frame or matrix of network embeddings corresponding to the samples in `x`. Only needed if training embeddings were manually inputted.
`y`	A vector of responses.
`A_full`	An adjacency matrix representing the network structure for the full set of nodes (training + testing nodes in that order)
`nodeids`	(Optional) vector of node IDs of length n. If provided, node IDs indicate the rows of A, corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.
`method`	A character string indicating the method to use for computing feature importances. Options are: • `"permute"`: Permutation-based global feature importance. • `"mdi+"`: MDI+ global feature importance. • `"local"`: Local feature importance.
`B`	Number of samples for permutation-based feature importance. Ignored if `method` is not `"permute"`.
`metric`	A function to compute the metric used for global feature importances. Defaults to R-squared for regression and AUROC for classification. Ignored if `method` is `"local"`.

Value

If method is "permute" or "mdi+", a tibble with two columns:

- var: The name of the feature.
- importance: The computed feature importance score. If method is "local", an n x p

Examples

```
data(example_data)
nerfplus_out <- nerfplus(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
)
fi_out <- get_feature_importances(
  nerfplus_out, x = example_data$xtest, y = example_data$ytest,
  A_full = example_data$A_full, method = "mdi+"
)
```

---

get_loo *Compute Leave-One-Out (LOO) predictions and parameter changes for a NeRF+ model*

Description

This function computes leave-one-out (LOO) predictions and parameter changes for a fitted NeRF+ model. For each sample, it computes the model's predictions and parameter estimates when that sample is left out of the training data. This is useful for assessing model stability and identifying influential observations.

Usage

```
get_loo(
  object,
  x,
  x_embed = NULL,
  y,
  A = NULL,
  xtest = NULL,
  xtest_embed = NULL,
  ytest = NULL,
  A_full = NULL,
  nodeids = NULL,
```

```
  nodeids_test = NULL,
  metric = NULL,
  return_all = FALSE
)
```

Arguments

object	A fitted NeRF+ model object.
x	A data frame or matrix containing the training data.
x_embed	An optional data frame or matrix of network embeddings corresponding to the training samples. Only needed if training embeddings were manually inputted.
y	A vector of responses for the training data.
A	An adjacency matrix representing the network structure for the training nodes.
xtest	An optional data frame or matrix containing the test data.
xtest_embed	An optional data frame or matrix of network embeddings corresponding to the test samples.
ytest	An optional vector of responses for the test data.
A_full	An optional adjacency matrix representing the network structure for the full set of nodes (training + testing nodes in that order).
nodeids	An optional vector of node IDs of length n. If provided, node IDs indicate the rows of A corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.
nodeids_test	An optional vector of node IDs for the test data.
metric	An optional function to compute the metric used for evaluating prediction changes. If NULL, defaults to RMSE for linear models and AUROC for logistic models.
return_all	Logical indicating whether to return all intermediate results for each tree in the forest. Defaults to FALSE.

Value

A list containing:

loo_params	A list of LOO parameter estimates for each tree.
change_alphas	Matrix of changes in alpha parameters when each sample is left out.
mean_change_alphas	Mean change in alpha parameters across all trees.
mean_change_betas	Mean change in beta parameters across all trees.
loo_preds	Matrix of LOO predictions, where each column corresponds to predictions when a different sample is left out.
loo_preds_test	Matrix of LOO predictions on test data (if provided).
mean_change_error_test	Mean change in prediction error on test data across all trees (if provided).
mean_change_preds_test	Mean change in predictions on test data across all trees (if provided).

Examples

```
data(example_data)
nerfplus_out <- nerfplus(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
)
loo_out <- get_loo(
  nerfplus_out,
  x = example_data$x, y = example_data$y, A = example_data$A,
  xtest = example_data$xtest, ytest = example_data$ytest,
  A_full = example_data$A_full
)
```

---

interpret_nerfplus *Interpret NeRF+ Model*

Description

Wrapper function around `get_feature_importances()` and `get_loo()`. Helpful to format results correctly for input into NeRF+ Interpreter Shiny application.

Usage

```
interpret_nerfplus(
  object,
  x,
  x_embed = NULL,
  y,
  A,
  nodeids = NULL,
  xtest,
  xtest_embed = NULL,
  ytest,
  A_full,
  nodeids_test = NULL,
  methods = c("permute", "mdi+", "local", "loo"),
  B = 10,
  metric = NULL,
  save = FALSE,
  save_dir = "results_nerfplus"
)
```

Arguments

object	A fitted NeRF+ model object.
x	A data frame or matrix containing the training data.
x_embed	An optional data frame or matrix of network embeddings corresponding to the training samples. Only needed if training embeddings were manually inputted.
y	A vector of responses for the training data.
A	An adjacency matrix representing the network structure for the training nodes.
nodeids	An optional vector of node IDs of length n. If provided, node IDs indicate the rows of A corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.
xtest	An optional data frame or matrix containing the test data.
xtest_embed	An optional data frame or matrix of network embeddings corresponding to the test samples.
ytest	An optional vector of responses for the test data.
A_full	An optional adjacency matrix representing the network structure for the full set of nodes (training + testing nodes in that order).
nodeids_test	An optional vector of node IDs for the test data.
methods	A character vector of methods to run. Can include any of "permute", "mdi+", "local", and "loo".
B	Number of permutations to use for permutation feature importance.
metric	A function to compute the metric used for global feature importances. Defaults to R-squared for regression and AUROC for classification.
save	Whether to save the results to disk.
save_dir	Directory to save results to if save = TRUE.

Value

A list with the following components:

object	The fitted NeRF+ model.
data_list	A list containing the data used for interpretation.
fi_results	A list containing the feature importance results.
loo_results	A list containing the sample influence results.

Examples

```
data(example_data)
nerfplus_out <- nerfplus(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
```

```
)
out <- interpret_nerfplus(
  nerfplus_out, x = example_data$x, y = example_data$y, A = example_data$A,
  xtest = example_data$xtest, ytest = example_data$ytest,
  A_full = example_data$A_full
)
```

---

nerfplus *Fit Network-assisted Random Forest+ (NeRF+)*

Description

Fit Network-assisted Random Forest+ (NeRF+)

Usage

```
nerfplus(
  x,
  y,
  A = NULL,
  nodeids = NULL,
  family = c("linear", "logistic"),
  include_raw = TRUE,
  include_netcoh = TRUE,
  embedding = NULL,
 embedding_options = list(ndim = 2, regularization = 0.5, varimax = FALSE, center =
    TRUE, scale = TRUE),
  standardize_x = TRUE,
  normalize_stump = FALSE,
  sample_split = c("none", "oob", "inbag"),
  ntrees = 500,
  mtry = NULL,
  lambda_netcoh,
  lambda_embed = lambda_raw,
  lambda_raw = lambda_stump,
  lambda_stump,
  lambda_l = 0.05,
  parallel = FALSE,
  num.threads = 1,
  ...
)
```

Arguments

x	A numeric matrix or data frame of predictors (features); size n x p. Should be centered so that each column has mean 0.

y — A numeric vector of responses of length n. Should be centered so that the mean is 0.

A — An adjacency matrix representing the network structure.

nodeids — (Optional) vector of node IDs of length n. If provided, node IDs indicate the rows of A, corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.

family — A character string indicating the type of model to fit. Currently, only "linear" and "logistic" are supported.

include_raw — Logical indicating whether to include the raw covariates in the NeRF+ model. Default is TRUE.

include_netcoh — Logical indicating whether to include the individual node effects and network cohesion regularization in the NeRF+ model. Default is TRUE.

embedding — Embedding type(s), at least one of "adjacency", "laplacian", score", or NULL (i.e., do not include any network embedding features). Alternatively, can directly input an n x d matrix of network embedding features corresponding to x.

embedding_options — A list of options for the network embedding. Ignored if embedding = NULL. If provided, the list should contain the following components:

- ndim: Number of dimensions in the embedding (default is 2).
- regularization: Regularization parameter for the adjacency matrix (default is 0.5).
- varimax: Whether to apply varimax rotation to the embedding (default is FALSE).
- center: Whether to center the embedding so that each column has mean 0 (default is TRUE).
- scale: Whether to scale the embedding so that first embedding component column has SD 1 (default is TRUE). All other embedding components are scaled, proportional to their eigenvalues.

standardize_x — Logical indicating whether to standardize the covariates so that each column has mean 0 and SD 1. Default is TRUE.

normalize_stump — Logical indicating whether to normalize the decision stump features by number of samples in children nodes. Default is FALSE.

sample_split — Character string indicating how to split the samples for training the model; one of "none" (default), "oob", or "inbag". If "none", all samples are used for estimating coefficients in NeRF+. If "oob", only out-of-bag samples are used for estimating coefficients in NeRF+. If "inbag", only in-bag samples are used for estimating coefficients in NeRF+.

ntrees — Number of trees in ensemble.

mtry — Number of features to consider at each split. Default is the number of features / 3 for regression and the square root of the number of features for classification.

lambda_netcoh — Regularization parameter for the network cohesion term. Can be either a scalar or a vector of length ntrees, specifying the regularization parameter for each tree. Ignored if include_netcoh = FALSE.

lambda_embed	Regularization parameter for the network embedding features. Default is same as lambda_raw. Can be either a scalar or a vector of length ntrees, specifying the regularization parameter for each tree. Ignored if embedding = NULL.
lambda_raw	Regularization parameter for the raw covariates. Default is same as lambda_stump. Can be either a scalar or a vector of length ntrees, specifying the regularization parameter for each tree. Ignored if include_raw = FALSE.
lambda_stump	Regularization parameter for the decision stump features. Can be either a scalar or a vector of length ntrees, specifying the regularization parameter for each tree.
lambda_l	(Optional) Regularization parameter for the graph Laplacian. Default is 0.05. Can be either a scalar or a vector of length ntrees, specifying the regularization parameter for each tree.
parallel	Logical indicating whether to use parallel processing.
num.threads	Number of threads to use for parallel processing. Default is 1. Ignored if parallel = FALSE.
...	Additional arguments passed to the ranger::ranger() function for fitting the random forest model.

Value

A list containing the following:

- rf_fit: The fitted random forest model object from ranger::ranger().
- nerfplus_fits: A list of fitted NeRF+ models for each tree in the random forest. Each element of the list is a fitted model object that can be used to make predictions.
- tree_infos: A list of tree information objects for each tree in the random forest.
- pre_rf_preprocessing_info: A list containing preprocessing information for the NeRF+ model; output of fit_pre_rf_preprocessing().
- regularization_params: A list containing the regularization parameters used in the NeRF+ model
- model_info: A list containing information about the model, such as family, include_raw, include_netcoh, normalize_stump, and sample_split.
- unordered_factors: A character vector of variable names that are unordered factors.

Examples

```
data(example_data)
nerfplus_out <- nerfplus(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
)
```

`nerfplus_conformal` *Conformal prediction method for NeRF+*

Description

This function generates conformal prediction intervals for a fitted NeRF+ model, using a specified calibration dataset to compute the nonconformity scores (measured as the absolute residuals between the observed responses and the predictions) and a specified test dataset for which predictions and prediction intervals are to be generated.

Usage

```
nerfplus_conformal(
  object,
  x_cal,
  x_cal_embed = NULL,
  y_cal,
  x_test,
  x_test_embed = NULL,
  A_full,
  nodeids_cal = NULL,
  nodeids_test = NULL,
  alpha = 0.05
)
```

Arguments

`object`	A fitted NeRF+ model object.
`x_cal`	A data frame or matrix of calibration data.
`x_cal_embed`	Optional embedding data frame or matrix for the calibration data, whose rows are aligned with those in `x_cal`. If provided, it will be used to augment the input `x_cal` data. Only needed if training embeddings were manually inputted.
`y_cal`	A vector of observed responses for the calibration data.
`x_test`	A data frame or matrix of test data for which predictions are to be made.
`x_test_embed`	Optional embedding data frame or matrix for the test data, whose rows are aligned with those in `x_test`. If provided, it will be used to augment the input `x_test` data. Only needed if training embeddings were manually inputted.
`A_full`	An adjacency matrix representing the network structure for the full set of nodes (training + calibration + testing nodes in that order, unless `nodeids_cal` and `nodeids_test` are provided, in which case the order of nodes in `A_full` should align with the order of node IDs in `nodeids_cal` and `nodeids_test`).
`nodeids_cal`	(Optional) vector of node IDs for the calibration data, of length equal to nrows in `x_cal`. If provided, node IDs indicate the rows of A_full, corresponding to each calibration sample. If not provided, the rows of A_full are assumed to be in the order of (x_train, x_cal, x_test).

nodeids_test	(Optional) vector of node IDs for the test data, of length equal to nrows in x_test. If provided, node IDs indicate the rows of A_full, corresponding to each test sample. If not provided, the rows of A_full are assumed to be in the order of (x_train, x_cal, x_test).
alpha	Significance level for conformal prediction intervals. Default is 0.05 for 95% prediction intervals.

Value

A tibble with columns pred, lower_bound, and upper_bound, containing the predicted values and the corresponding lower and upper bounds of the conformal prediction intervals for each sample in the test data.

Examples

```
data(example_data)
train_idx <- 1:(nrow(example_data$x) / 2)
cal_idx <- (nrow(example_data$x) / 2 + 1):nrow(example_data$x)
x_train <- example_data$x[train_idx, ]
y_train <- example_data$y[train_idx]
x_cal <- example_data$x[cal_idx, ]
y_cal <- example_data$y[cal_idx]
nerfplus_out <- nerfplus(
  x = x_train, y = y_train, A = example_data$A[train_idx, train_idx],
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
)
conformal_out <- nerfplus_conformal(
  nerfplus_out,
  x_cal = x_cal, y_cal = y_cal,
  x_test = example_data$xtest,
  A_full = example_data$A_full,
  alpha = 0.05
)
conformal_out |>
  dplyr::mutate(
   contains_true_y = y_cal >= lower_bound & y_cal <= upper_bound
  ) |>
  dplyr::summarize(
   coverage = mean(contains_true_y)
  )
```

---

nerfplus_cv *Fit Network-assisted Random Forest+ (NeRF+) with Cross-Validation*

Description

Fit Network-assisted Random Forest+ (NeRF+) with Cross-Validation

Usage

```
nerfplus_cv(
  x,
  y,
  A = NULL,
  nodeids = NULL,
  cv = 5,
  cv_foldids = NULL,
  family = c("linear", "logistic"),
  include_raw = TRUE,
  include_netcoh = TRUE,
  embedding = NULL,
 embedding_options = list(ndim = 2, regularization = 0.5, varimax = FALSE, center =
    TRUE, scale = TRUE),
  standardize_x = TRUE,
  normalize_stump = FALSE,
  sample_split = c("none", "oob", "inbag"),
  ntrees = 500,
  ntrees_cv = ntrees,
  mtry = NULL,
  lambdas_netcoh,
  lambdas_embed = NULL,
  lambdas_raw = NULL,
  lambdas_stump,
  lambdas_l = 0.05,
  parallel = FALSE,
  num.threads = 1,
  ...
)
```

Arguments

x	A numeric matrix or data frame of predictors (features); size n x p. Should be centered so that each column has mean 0.
y	A numeric vector of responses of length n. Should be centered so that the mean is 0.
A	An adjacency matrix representing the network structure.
nodeids	(Optional) vector of node IDs of length n. If provided, node IDs indicate the rows of A, corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.
cv	Number of cross-validation folds. Default is 5.
cv_foldids	(Optional) List of length cv, where each component in the list is a vector of sample indices in that fold. If NULL (default), cross-validation folds will be created randomly.

`family` A character string indicating the type of model to fit. Currently, only "linear" and "logistic" are supported.

`include_raw` Logical indicating whether to include the raw covariates in the NeRF+ model. Default is `TRUE`.

`include_netcoh` Logical indicating whether to include the individual node effects and network cohesion regularization in the NeRF+ model. Default is `TRUE`.

`embedding` Embedding type(s), at least one of "adjacency", "laplacian", score", or NULL (i.e., do not include any network embedding features). Alternatively, can directly input an n x d matrix of network embedding features corresponding to `x`.

`embedding_options`
A list of options for the network embedding. Ignored if `embedding = NULL`. If provided, the list should contain the following components:

- `ndim`: Number of dimensions in the embedding (default is 2).
- `regularization`: Regularization parameter for the adjacency matrix (default is 0.5).
- `varimax`: Whether to apply varimax rotation to the embedding (default is FALSE).
- `center`: Whether to center the embedding so that each column has mean 0 (default is TRUE).
- `scale`: Whether to scale the embedding so that first embedding component column has SD 1 (default is TRUE). All other embedding components are scaled, proportional to their eigenvalues.

`standardize_x` Logical indicating whether to standardize the covariates so that each column has mean 0 and SD 1. Default is `TRUE`.

`normalize_stump`
Logical indicating whether to normalize the decision stump features by number of samples in children nodes. Default is `FALSE`.

`sample_split` Character string indicating how to split the samples for training the model; one of "none" (default), "oob", or "inbag". If "none", all samples are used for estimating coefficients in NeRF+. If "oob", only out-of-bag samples are used for estimating coefficients in NeRF+. If "inbag", only in-bag samples are used for estimating coefficients in NeRF+.

`ntrees` Number of trees in ensemble.

`ntrees_cv` Number of trees that will be tuned using cross-validation. Default is `ntrees` (i.e., every tree will be tuned). Reduce this number to speed up the cross-validation process. For all trees that aren't tuned, the hyperparameter will be chosen randomly from the tuned trees.

`mtry` Number of features to consider at each split. Default is the number of features / 3 for regression and the square root of the number of features for classification.

`lambdas_netcoh` Vector of regularization parameters for the network cohesion term.

`lambdas_embed` Vector of regularization parameters for the network embedding features. If `NULL`, the regularization parameter corresponding to the network embedding features will be equal to the regularization parameter for the raw covariates.

lambdas_raw	Vector of regularization parameters for the raw covariate features. If NULL, the regularization parameter for the raw covariates will be equal to the regularization parameter for the decision stump features.
lambdas_stump	Vector of regularization parameters for the decision stump features.
lambdas_l	Vector of regularization parameters for the graph Laplacian.
parallel	Logical indicating whether to use parallel processing.
num.threads	Number of threads to use for parallel processing. Default is 1. Ignored if parallel = FALSE.
...	Additional arguments passed to the ranger::ranger() function for fitting the random forest model.

Value

A list containing the following:

- rf_fit: The fitted random forest model object from ranger::ranger().
- nerfplus_fits: A list of fitted NeRF+ models for each tree in the random forest using the tuned hyperparameters. Each element of the list is a fitted model object that can be used to make predictions.
- cv_losses: A list of ntrees_cv data frames containing the cross-validation losses for each tree and each fold. Each item in the list corresponds to a tree in the random forest. Each row in the data frame corresponds to a different set of hyperparameters.
- best_cv_params: A data frame containing the used hyperparameters for each tree in the random forest.
- tree_infos: A list of tree information objects for each tree in the random forest.
- pre_rf_preprocessing_info: A list containing preprocessing information for the NeRF+ model; output of fit_pre_rf_preprocessing().
- regularization_params: A list containing the regularization parameters used in the NeRF+ model
- model_info: A list containing information about the model, such as family, include_raw, include_netcoh, normalize_stump, and sample_split.
- unordered_factors: A character vector of variable names that are unordered factors.

Examples

```
data(example_data)
nerfplus_cv_out <- nerfplus_cv(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambdas_netcoh = c(0.1, 1),
  lambdas_embed = c(0, 0.1),
  lambdas_raw = c(1, 2),
  lambdas_stump = c(1, 2),
  family = "linear", embedding = "laplacian", sample_split = "none"
)
```

oos_network_embedding *Out-of-sample network embedding method*

Description

This function takes in out-of-sample points (which were not available at training time) and embeds them in a previously trained network embedding space.

Usage

```
oos_network_embedding(embedding_fit, A, A_full)
```

Arguments

embedding_fit	Previously trained network embedding; should be output of fit_network_embedding().
A	Adjacency matrix corresponding to the training data points.
A_full	Adjacency matrix representing the full set of nodes (training + testing nodes in that order).

Value

A list of two:

- X: A list of matrices, each corresponding to a method in embedding_fit$method, where each matrix has ndim columns representing the embedding for the out-of-sample points.
- method: The method(s) used for embedding.

Examples

```
data(example_data)
embedding_fit <- fit_network_embedding(
  example_data$A, ndim = 2, method = "laplacian"
)
oos_embedding <- oos_network_embedding(
  embedding_fit, A = example_data$A, A_full = example_data$A_full
)
```

`predict.nerfplus` *Predict method for a NeRF+ model*

Description

Predict method for a NeRF+ model

Usage

```
## S3 method for class 'nerfplus'
predict(
  object,
  x,
  x_embed = NULL,
  A_full,
  nodeids = NULL,
  type = c("response", "alpha"),
  return_all = FALSE,
  ...
)
```

Arguments

`object`	A fitted NeRF+ model object.
`x`	A data frame or matrix of new data for which predictions are to be made.
`x_embed`	Optional embedding data frame or matrix, whose rows are aligned with those in `x`. If provided, it will be used to augment the input `x` data. Only needed if training embeddings were manually inputted.
`A_full`	An adjacency matrix representing the network structure for the full set of nodes (training + testing nodes in that order). Note: if `nrow(x) == nrow(A_full)`, then `x` is assumed to be the training data.
`nodeids`	(Optional) vector of node IDs of length equal to nrows in `x`. If provided, node IDs indicate the rows of A_full, corresponding to each sample. If not provided, the rows of A_full are assumed to be in the order of (x_train, x).
`type`	Type of prediction to return; one of "response" (default) or "alpha". If "response", the predicted values are returned. If "alpha", the estimated individual node effects are returned.
`return_all`	If `TRUE`, returns a list of predictions for each tree in the ensemble. If `FALSE` (default), returns the average prediction across all trees.
`...`	Additional arguments, currently unused.

Value

If `return_all = FALSE`, this function returns a vector of predicted values (if `type = "response"`) or estimated individual node effects (if `type = "alpha"`). If `return_all = TRUE`, this function returns a list of predicted values or estimated individual node effects, where each element in the list corresponds to a different tree in the ensemble.

Examples

```
data(example_data)
nerfplus_out <- nerfplus(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
)
predicted_y <- predict(
  nerfplus_out, x = example_data$xtest, A_full = example_data$A_full,
  type = "response"
)
estimated_alphas <- predict(
  nerfplus_out, x = example_data$xtest, A_full = example_data$A_full,
  type = "alpha"
)
```

---

predict.rnc *Predict method for rnc (regression with network cohesion) objects*

Description

Predict method for rnc (regression with network cohesion) objects

Usage

```
## S3 method for class 'rnc'
predict(object, x, A_full, nodeids = NULL, ...)
```

Arguments

object	An object of class rnc containing the fitted model.
x	A numeric matrix or data frame of predictors (features) to make predictions for; size n x p.
A_full	An adjacency matrix representing the network structure for the full set of nodes (training + testing nodes in that order)
nodeids	(Optional) vector of node IDs of length n. If provided, node IDs indicate the rows of A_full, corresponding to each sample. If not provided, the rows of A_full are assumed to be in the order of (x_train, x).
...	Not used.

Value

A list of two:

- y: A numeric vector of predictions for the response variable.
- alpha: A numeric vector of node-specific effects (intercepts).

Examples

```
data(example_data)

# demonstrate prediction for RNC fit
rnc_fit <- rnc(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1, lambda_x = 0.5
)
predicted_out <- predict(
  rnc_fit, x = example_data$xtest, A_full = example_data$A_full
)

# demonstrate prediction for RNC CV fit
cv_fit <- rnc_cv(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambdas_netcoh = c(0, 0.1, 1),
  lambdas_x = c(0.1, 0.5)
)
predicted_out <- predict(
  cv_fit, x = example_data$xtest, A_full = example_data$A_full
)
```

---

rnc *Fit regression with network cohesion*

Description

This function fits a regression model with network cohesion regularization. This code has been adapted from the netcoh R package with minor tweaks for increased speed and flexibility. Currently, it can handle linear and logistic regression only.

Usage

```
rnc(
  x,
  y,
  A,
  nodedegrees = NULL,
  nodeids = NULL,
  lambda_netcoh,
```

```
  lambda_x = 0,
  lambda_l = 0.05,
  family = c("linear", "logistic"),
  low_dim = NULL,
  init = NULL,
  newton_maxit = 50,
  newton_tol = 1e-04,
  verbose = FALSE
)
```

Arguments

x	A numeric matrix or data frame of predictors (features); size n x p. Should be centered so that each column has mean 0.
y	A numeric vector of responses of length n. Should be centered so that the mean is 0.
A	An adjacency matrix representing the network structure.
nodedegrees	(Optional) A vector of node degrees. If provided, it is used to compute the graph Laplacian. If not provided, it is computed from the adjacency matrix A.
nodeids	(Optional) vector of node IDs of length n. If provided, node IDs indicate the rows of A, corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.
lambda_netcoh	Regularization parameter for the network cohesion term.
lambda_x	(Optional) Regularization parameter for the (linear) covariates. Default is 0 (i.e., no regularization).
lambda_l	(Optional) Regularization parameter for the graph Laplacian. Default is 0.05.
family	A character string indicating the type of model to fit. Currently, only "linear" and "logistic" are supported.
low_dim	(Optional) If TRUE, the algorithm will use a naive solver for low-dimensional problems. Default is NULL, which will use the navie low-dimensional solver if the number of covariates is <= 1/5 * the number of samples. Only used if family = "linear".
init	(Optional) initial values for the optimization algorithm to fit logistic regression. Ignored for linear regression.
newton_maxit	Maximum number of Newton iterations when fitting logistic regression. Default is 50. Ignored for linear regression.
newton_tol	Tolerance for convergence of Newton iterations when fitting logistic regression. Default is 1e-4. Ignored for linear regression.
verbose	Logical indicating whether to print progress messages.

Value

A list with the following components:

- alpha: n x 1 matrix of coefficients corresponding to the individual node effects

- beta: p x 1 matrix of coefficients corresponding to the covariates
- lambda_netcoh: Regularization parameter for the network cohesion term.
- lambda_x: Regularization parameter for the covariates.
- lambda_l: Regularization parameter for the graph Laplacian.
- nalpha_train: Number of distinct nodes in the training set (i.e., number of rows in A).
- family: The type of model fitted, either "linear" or "logistic".

Examples

```
data(example_data)
rnc_fit <- rnc(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1, lambda_x = 0.5
)
```

---

rnc_cv *Fit regression with network cohesion with cross-validation wrapper*

Description

This function fits a regression model with network cohesion regularization with a cross-validation wrapper. This code has been adapted from the netcoh R package with minor tweaks for increased speed and flexibility. Currently, it can handle linear and logistic regression only.

Usage

```
rnc_cv(
  x,
  y,
  A,
  nodeids = NULL,
  lambdas_netcoh,
  lambdas_x = 0,
  lambdas_l = 0.05,
  lambda_grid = NULL,
  family = c("linear", "logistic"),
  cv = 5,
  cv_foldids = NULL,
  refit = TRUE,
  newton_maxit = 50,
  newton_tol = 1e-04,
  verbose = FALSE
)
```

Arguments

x	A numeric matrix or data frame of predictors (features); size n x p. Should be centered so that each column has mean 0.
y	A numeric vector of responses of length n. Should be centered so that the mean is 0.
A	An adjacency matrix representing the network structure.
nodeids	(Optional) vector of node IDs of length n. If provided, node IDs indicate the rows of A, corresponding to each sample. If not provided, the rows of A are assumed to be in the same order as the rows of x and y.
lambdas_netcoh	Vector or list of regularization parameters for the network cohesion term.
lambdas_x	Vector or list of regularization parameters for the (linear) covariates. Default is 0 (i.e., no regularization).
lambdas_l	(Optional) Vector or list of regularization parameters for the graph Laplacian.
lambda_grid	(Optional) A data frame with columns lambda_netcoh, lambda_x, and lambda_l, where each row specifies a different set of regularization parameters to try. If NULL, a grid of regularization parameters will be generated based upon all possible combination of the provided lambdas_netcoh, lambdas_x, and lambdas_l.
family	A character string indicating the type of model to fit. Currently, only "linear" and "logistic" are supported.
cv	Number of cross-validation folds. Default is 5.
cv_foldids	(Optional) List of length cv, where each component in the list is a vector of sample indices in that fold. If NULL (default), cross-validation folds will be created randomly.
refit	Logical indicating whether or not to refit tuned model on full training set after cross-validation. Default is TRUE.
newton_maxit	Maximum number of Newton iterations when fitting logistic regression. Default is 50. Ignored for linear regression.
newton_tol	Tolerance for convergence of Newton iterations when fitting logistic regression. Default is 1e-4. Ignored for linear regression.
verbose	Logical indicating whether to print progress messages.

Value

A list with the following components:

- lambda_grid: A data frame with the regularization parameters used in the cross-validation.
- best_params: A data frame with the best regularization parameters found during cross-validation.
- cv_errs: A matrix of cross-validation errors, where each row corresponds to a set of regularization parameters and each column corresponds to a fold. If regression, the RMSE is returned. If classification, the AUROC is returned.
- cv_means: A vector of mean cross-validation errors for each set of regularization parameters.
- cv_sds: A vector of standard deviations of cross-validation errors for each set of regularization parameters.
- Other components from the fitted model if refit = TRUE (see output of rnc()).

Examples

```
data(example_data)

# fit RNC with CV
cv_fit <- rnc_cv(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambdas_netcoh = c(0, 0.1, 1),
  lambdas_x = c(0.1, 0.5)
)

# fit RNC with CV, specifying lambda grid manually (equivalent to above)
lambda_grid <- expand.grid(
  lambda_netcoh = c(0, 0.1, 1),
  lambda_x = c(0.1, 0.5),
  lambda_l = 0.05
)
cv_fit <- rnc_cv(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_grid = lambda_grid
)
```

---

run_app *Run Shiny NeRF+ Interpret application*

Description

Run Shiny NeRF+ Interpret application

Usage

```
run_app(
  data_list = NULL,
  object = NULL,
  fi_results = NULL,
  loo_results = NULL,
  max_request_size = 100 * 1024^2,
  ...
)
```

Arguments

`data_list`	A list containing the data used for interpretation. Should have the same format as the `data_list` output from `interpret_nerfplus()`.
`object`	The fitted NeRF+ model. Should have the same format as the `object` output from `interpret_nerfplus()`.
`fi_results`	A list containing the feature importance results. Should have the same format as the `fi_results` output from `interpret_nerfplus()`.

`loo_results` A list containing the sample influence results. Should have the same format as the `loo_results` output from `interpret_nerfplus()`.

`max_request_size` Maximum request size for file uploads. Defaults to 100 MB.

`...` Additional arguments to pass to `shiny::shinyApp()`

Value

A shiny application

Examples

```
if (interactive()) {
  # launch app
  run_app()
}

# or run with arguments
data(example_data)
nerfplus_out <- nerfplus(
  x = example_data$x, y = example_data$y, A = example_data$A,
  lambda_netcoh = 1,
  lambda_embed = 0.1,
  lambda_raw = 2,
  lambda_stump = 3,
  family = "linear", embedding = "laplacian", sample_split = "none"
)
out <- interpret_nerfplus(
  nerfplus_out, x = example_data$x, y = example_data$y, A = example_data$A,
  xtest = example_data$xtest, ytest = example_data$ytest,
  A_full = example_data$A_full
)

if (interactive()) {
  run_app(
    data_list = out$data_list,
    object = out$object,
    fi_results = out$fi_results,
    loo_results = out$loo_results
  )
}
```

Index

∗ **datasets**
example_data, 2

example_data, 2

fit_network_embedding, 3
fit_pre_rf_preprocessing(), *11*, *16*

get_feature_importances, 4
get_loo, 5

interpret_nerfplus, 7
interpret_nerfplus(), *24*, *25*

nerfplus, 9
nerfplus_conformal, 12
nerfplus_cv, 13

oos_network_embedding, 17

predict.nerfplus, 18
predict.rnc, 19

ranger::ranger(), *11*, *16*
rnc, 20
rnc_cv, 22
run_app, 24

shiny::shinyApp(), *25*