Package ‘bpcs’

December 9, 2020

Title  Bayesian Paired Comparison Analysis with Stan
Version  1.0.0
Description  Models for the analysis of paired comparison data using Stan. The models include Bayesian versions of the Bradley-Terry model, including random effects (1 level), generalized model for predictors, order effect (home advantage) and the variations for the Davidson (1970) model to handle ties. Additionally, we provide a number of functions to facilitate inference and obtaining results with these models. References: Bradley and Terry (1952) <doi:10.2307/2334029>; Davidson (1970) <doi:10.1080/01621459.1970.10481082>; Carpenter et al. (2017) <doi:10.18637/jss.v076.i01>.

URL  https://github.com/davidissamattos/bpcs,  
       https://davidissamattos.github.io/bpcs/
BugReports  https://github.com/davidissamattos/bpcs/issues
License  MIT + file LICENSE
Encoding  UTF-8
LazyData  true
RoxygenNote  7.1.1
Biarch  true
Depends  R (>= 4.0.0)
Imports  coda, dplyr, tidyr, stringr, ggplot2, gtools, methods, Rcpp  
        (>= 0.12.0), RcppParallel (>= 5.0.1), rstan (>= 2.20.0),  
        rstantools (>= 2.1.0), tibble, tidyselect, HDInterval,  
        shinystan, loo, magrittr, badger, stats, rlang, knitr
LinkingTo  BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0),  
          RcppParallel (>= 5.0.1), rstan (>= 2.20.0), StanHeaders (>=  
          2.20.0)
SystemRequirements  GNU make
Suggests  rmarkdown, testthat, covr, bayesplot, kableExtra
VignetteBuilder  knitr
NeedsCompilation  yes
R topics documented:

R topics documented:

bpcs-package .......................................................... 3
bpc ........................................................................... 3
brasil_soccer_league ...................................................... 6
check_if_there_are_na ...................................................... 7
check_if_there_are_ties ...................................................... 8
check_numeric_predictor_matrix ..................................... 8
check_predictors_df_contains_all_players ............................ 9
check_result_column ...................................................... 9
check_z_column .......................................................... 10
compute_scores ......................................................... 10
compute_ties ............................................................. 11
create_array_of_par_names ............................................. 12
create_bpc_object ....................................................... 12
create_cluster_index .................................................... 13
create_cluster_index_with_existing_lookup_table ................. 14
create_index ............................................................ 15
create_index_cluster_lookuptable ................................... 15
create_index_lookuptable .............................................. 16
create_index_predictors_with_lookup_table ....................... 17
create_index_with_existing_lookup_table .......................... 17
create_predictors_lookup_table ..................................... 18
create_predictor_matrix_with_player_lookup_table ................ 18
expand_aggregated_data ............................................... 19
get_hpdi_parameters ................................................... 20
get_loo ................................................................. 21
get_model_parameters ................................................ 21
get_probabilities ....................................................... 22
get_rank_of_players .................................................. 23
get_sample_posterior .................................................. 24
get_stanfit ............................................................. 24
get_stanfit_summary ................................................... 25
get_waic ................................................................. 26
HPDI_from_stanfit ....................................................... 27
HPD_higher_from_column ............................................. 27
HPD_lower_from_column .............................................. 28
inv_logit ............................................................... 28
launch_shinystan ....................................................... 29
logit ................................................................. 30

Author  David Issa Mattos [aut, cre] (<https://orcid.org/0000-0002-2501-9926>),
Erika Martins Silva Ramos [aut]
(<https://orcid.org/0000-0001-7393-1410>)
Maintainer  David Issa Mattos <issamattos.david@gmail.com>
Repository  CRAN
Date/Publication  2020-12-09 09:20:16 UTC
**bpcs-package**

`match_cluster_names_to_cluster_lookup_table` .............................................. 30  
`match_player_names_to_lookup_table` .......................................................... 31  
`optimization_algorithms` .............................................................................. 31  
`predict.bpc` ................................................................................................. 32  
`print.bpc` ..................................................................................................... 33  
`replace_parameter_index_with_names` ............................................................ 34  
`sample_stanfit` ............................................................................................ 35  
`summary.bpc` ............................................................................................... 35  
`tennis_agresti` .............................................................................................. 36  

**Index**

bpcs-package  

**bpcs - A package for Bayesian Paired Comparison analysis with Stan**

**Description**

bpcs - A package for Bayesian Paired Comparison analysis with Stan

**References**

https://mc-stan.org

**bpc**  

**Bayesian Paired comparison regression models in Stan**

**Description**

This is the main function of the package. This function utilizes precompiled stan models to sample the posterior distribution of the specified model with the input data. For more information and larger examples of usage see the vignettes.

**Usage**

```r
bpc(  
data,  
player0,  
player1,  
player0_score = NULL,  
player1_score = NULL,  
result_column = NULL,  
z_player1 = NULL,  
cluster = NULL,  
predictors = NULL,  
model_type,
```
data: A data frame containing the observations. The other parameters specify the name of the columns.

player0: A string with name of the column containing the players 0. This column should be of string/character type and not be of factor type.

player1: A string with name of the column containing the players 0. This column should be of string/character type and not be of factor type.

player0_score: A string with name of the column containing the scores of players 0.

player1_score: A string with name of the column containing the scores of players 1.

result_column: A string with name of the column containing the winners. 0 for player 0, 1 for player 1, and 2 for ties.

z_player1: A string with the name of the column containing the order effect for player 1. E.g. if player1 has the home advantage this column should have 1 otherwise it should have 0.

cluster: A string with the name of the column containing the cluster for the observation. To be used with a random effects model. This column should contain strings.

predictors: A data frame that contains the players predictors values when using a generalized model. Only numeric values are accepted. Booleans are accepted but will be cast into integers. The first column should be for the player name, the others will be the predictors. The column names will be used as name for the predictors.

model_type: We first add a base model 'bt' or 'davidson' and then additional options with `-`

  - 'bt' for the Bradley Terry model. Ref: Bradley-Terry 1952
  - 'davidson' the Davidson model to handle ties. Ref: Davidson 1970
  - 'bt-ordereffect' for the Bradley-Terry with order effect, for home advantage. Ref: Davidson 1977
  - 'davidson-ordereffect' for the Davidson model with order effect, for home advantage, and ties. Ref: Davidson 1977
  - 'bt-generalized': for the generalized Bradley Terry model for subject specific predictors. Ref: Springall 1973
  - 'davidson-generalized' for the generalized Davidson model for subject specific predictors
  - 'bt-U': for the Bradley-Terry with random effects. Ref: Bockenholt 2001
  - 'davidson-U': For Davidson model with random effects
• 'bt-orderseffect-U' for Bradley-Terry with order effects and random effects, use similar syntax for other variations by appending the correct options

**solve_ties**
A string for the method of handling ties.
- 'random' for converting ties randomly,
- 'remove' for removing the tie occurrences
- 'none' to ignore ties. This requires a model capable of handling ties

**win_score**
A string that indicates if which score should win
- 'higher' score is winner
- 'lower' score is winner

**priors**
A list with the parameters for the priors.
- 'prior_lambda_mu' Mean value of the lambda parameter in the all models. For the generalized this is also the prior for the B the parameter for lambda ~ normal(mu, std)
- 'prior_lambda_std' Standard deviation of the lambda parameter in the all models. lambda ~ normal(mu, std)
- 'prior_nu_mu' Mean value of the nu parameter in the Davidson models. nu ~ normal(mu, std)
- 'prior_nu_std' Standard deviation ofnu parameter in the Davidson models. nu ~ normal(mu, std)
- 'prior_gm_mu' Mean value of the gm in the ordered effect model. gm ~ normal(mu, std). Default = 0
- 'prior_gm_std' Standard deviation of the gm parameter in the ordered effect model. gm ~ normal(mu, std). Default =
- 'prior_U_std' Standard deviation of the U parameter in the random effects model. U ~ normal(0, std). Default = 3.0

**chains**
Number of chains passed to Stan sampling. Positive integer, default=4. For more information consult Stan documentation

**iter**
Number of iterations passed to Stan sampling. Positive integer, default =2000. For more information consult Stan documentation

**warmup**
Number of iteration for the warmup passed to Stan sampling. Positive integer, default 1000. For more information consult Stan documentation

**show_chain_messages**
Hide chain messages from Stan

**seed**
a random seed for Stan

**Value**
An object of the class bpc. This object should be used in conjunction with the several auxiliary functions from the package

**References**


Examples

#For the simple Bradley-Terry model
bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')

brasil_soccer_league

This is a dataset with the results matches fromo the first league of the Brazilian soccer championship from 2017-2019. It was reduced and translatedfrom the adaduque/Brasileirao_Dataset repository

Description

This is a dataset with the results matches fromo the first league of the Brazilian soccer championship from 2017-2019. It was reduced and translatedfrom the adaduque/Brasileirao_Dataset repository

Usage

brasil_soccer_league

Format

Data frame that contains 1140 matches and 9 Columns from the Brazilian soccer championship

- Time: time of the day in 24h format
- DayWeek: day of the week
- Date: date YY-MM-DD
- HomeTeam: name of the team playing home
- VisitorTeam: name of the team playing visitor
check_if_there_are_na

- Round: Round number of the championship
- Stadium: Name of the stadium where the game was played
- ScoreHomeTeam: number of goals for the home team
- ScoreVisitorTeam: number of goals for the visitor

Source

https://github.com/adaoduque/Brasileirao_Dataset

---

check_if_there_are_na  Check for NA in the specific columns and returns T or F is there is at least 1 NA in those columns

Description

Check for NA in the specific columns and returns T or F is there is at least 1 NA in those columns

Usage

```r
check_if_there_are_na(
  d,
  player0,
  player1,
  player0_score = NULL,
  player1_score = NULL,
  result_column = NULL
)
```

Arguments

d a data frame
player0 the name of column for player0
player1 the name of column for player1
player0_score the name of column for player0 scores
player1_score the name of column for player1 scores
result_column the name of column for results

Value

TRUE (there are NA) or FALSE (no NA)
check_if_there_are_ties

*Check if a data frame column contains ties*

**Description**

Check if a data frame column contains ties

**Usage**

check_if_there_are_ties(d_column)

**Arguments**

- **d_column**: a column with the values for the ties

**Value**

T (there are ties) or F (no ties)

---

check_numeric_predictor_matrix

*Check if all values in the predictor matrix are numeric and not NA. Note that TRUE will be cast to 1 and FALSE will be cast to 0*

**Description**

Check if all values in the predictor matrix are numeric and not NA. Note that TRUE will be cast to 1 and FALSE will be cast to 0

**Usage**

check_numeric_predictor_matrix(predictor_matrix)

**Arguments**

- **predictor_matrix**: a predictor matrix generated by the create_predictor_matrix_with_player_lookup_table function

**Value**

TRUE (correct) or FALSE (with problems)
check_predictors_df_contains_all_players

*Check if the predictor df contains all players and only those*

**Description**
Check if the predictor df contains all players and only those

**Usage**
```r
check_predictors_df_contains_all_players(predictor_df, lookup_table)
```

**Arguments**
- `predictor_df`: the predictor input data frame
- `lookup_table`: a lookup table of the players

**Value**
TRUE (correct) or FALSE (with problems)

---

check_result_column

*Check if a data frame column contains only the values 1 0 and 2. Used to check the format of the results*

**Description**
Check if a data frame column contains only the values 1 0 and 2. Used to check the format of the results

**Usage**
```r
check_result_column(d_column)
```

**Arguments**
- `d_column`: a column from a data frame

**Value**
TRUE (correct) or FALSE (with problems)
### check_z_column

*Description*

Check if a data frame column contains only the values 1 or 0. For the z column

*Usage*

```r
check_z_column(d_column)
```

*Arguments*

- `d_column`: a column of a data frame to be tested

*Value*

TRUE (correct) or FALSE (with problems)

### compute_scores

*Description*

Giving a player0 an player1 scores, this functions adds one column to the data frame containing who won (0=player0 1=player1 2=tie) and another if it was a tie. The ties column superseeds the y column. If it was tie the y column does not matter y column: (0= player0 1=player1 2=tie) ties column (0=not tie, 1=tie)

*Usage*

```r
compute_scores(
  d, 
  player0_score, 
  player1_score, 
  solve_ties = "random",  
  win_score = "higher"
)
```
compute_ties

Arguments

- **d**: dataframe
- **player0_score**: name of the column in data
- **player1_score**: name of the column in data
- **solve_ties**: Method to solve the ties, either randomly allocate, or do nothing, or remove the row from the dataset ('random', 'none', 'remove').
- **win_score**: decides if who wins is the one that has the highest score or the lowest score

Value

A dataframe with column 'y' that contains the results of the comparison and a ties column indicating if there was ties

Description

Giving a result column we create a new column with ties (0 and 1 if it has)

Usage

```R
compute_ties(d, result_column)
```

Arguments

- **d**: data frame
- **result_column**: column where the result is

Value

Dataframe with a column called ties
create_array_of_par_names

Create an array with the parameter name and to what player/cluster it refers to in the order stan presents

Description

Create an array with the parameter name and to what player/cluster it refers to in the order stan presents

Usage

create_array_of_par_names(par, lookup_table, cluster_lookup_table = NULL)

Arguments

par               name of the parameter
lookup_table     lookup table of the players
cluster_lookup_table
                  a lookup table of the clusters

Value

a data. frame where we change the names in the variable column to the corresponding parameter_name from the lookup table

create_bpc_object

Defines the class bpc and creates the bpc object. To create we need to receive some defined parameters (the arguments from the bpc function), a lookup table and a the stanfit object generated from the rstan sampling procedure

Description

Defines the class bpc and creates the bpc object. To create we need to receive some defined parameters (the arguments from the bpc function), a lookup table and a the stanfit object generated from the rstan sampling procedure
Usage

create_bpc_object(
  stanfit,
  lookup_table,
  model_type,
  standata,
  call_arg,
  cluster_lookup_table = NULL,
  predictors_df = NULL,
  predictors_lookup_table = NULL,
  predictors_matrix = NULL
)

Arguments

stanfit Stanfit object returned by rstan::sampling
lookup_table lookup_table dataframe. Two columns one Index the other Names where each
each index will match a string in the names
model_type the type of the model used to call stan (string)
standata a list with the data used to call the rstan::sampling procedure
call_arg a list with the arguments called from the bpc function
cluster_lookup_table a lookup table with we have random effects
predictors_df the data frame of the predictors for a generalized model
predictors_lookup_table a lookup table for generalized models
predictors_matrix a matrix of predictors for generalized models

Value

a bpc object

create_cluster_index Create two columns with the indexes for the names of the players Here
we create a new lookup table. Should be used when sampling the
parameters

Description

Create two columns with the indexes for the names of the players Here we create a new lookup
table. Should be used when sampling the parameters

Usage

create_cluster_index(d, cluster)
create_cluster_index_with_existing_lookup_table

Arguments

- d: A data frame containing the observations. The other parameters specify the name of the columns.
- cluster: The name of the column of data that contains player0.

Value

A dataframe with the additional columns 'cluster_index'.
**create_index**

Create two columns with the indexes for the names of the players. Here we create a new lookup table. Should be used when sampling the parameters.

**Description**

Create two columns with the indexes for the names of the players. Here we create a new lookup table. Should be used when sampling the parameters.

**Usage**

```r
create_index(d, player0, player1)
```

**Arguments**

- `d`: A data frame containing the observations. The other parameters specify the name of the columns.
- `player0`: The name of the column of data `d` contains player0.
- `player1`: The name of the column of data `d` contains player0.

**Value**

A dataframe with the additional columns `player0_index` and `player1_index` that contains the indexes.

**create_index_cluster_lookuptable**

Create a lookup table of names and indexes. Note that the indexes will be created in the order they appear. For string this does not make much difference but for numbers the index might be different than the actual number that appears in names.

**Description**

Create a lookup table of names and indexes. Note that the indexes will be created in the order they appear. For string this does not make much difference but for numbers the index might be different than the actual number that appears in names.

**Usage**

```r
create_index_cluster_lookuptable(d, cluster)
```
create_index_lookuptable

**Arguments**

d A data frame containing the observations. The other parameters specify the name of the columns

cluster A string with the name of the cluster variable

**Value**

A dataframe of a lookup table with columns Names and Index

---

**Description**

Create a lookup table of names and indexes Note that the indexes will be created in the order they appear. For string this doesn't make much difference but for numbers the index might be different than the actual number that appears in names

**Usage**

create_index_lookuptable(d, player0, player1)

**Arguments**

d A data frame containing the observations. The other parameters specify the name of the columns

player0 The name of the column of data contains player0

player1 The name of the column of data contains player0

**Value**

A dataframe of a lookup table with columns Names and Index
create_index_predictors_with_lookup_table

Receives one column with player names and returns a data frame with the relevant index columns based on a given lookup table. To be used with the predictors data frame.

**Description**

Receives one column with player names and returns a data frame with the relevant index columns based on a given lookup table. To be used with the predictors data frame.

**Usage**

```r
create_index_predictors_with_lookup_table(d, player, lookup_table)
```

**Arguments**

- `d`: a data frame of the predictors
- `player`: The name of the column of data data contains the player
- `lookup_table`: a lookup table data frame

**Value**

A dataframe with the additional column `player_index`.

create_index_with_existing_lookup_table

Create two columns with the indexes for the names. Here we use an existing lookup table. Should be used in predicting.

**Description**

Create two columns with the indexes for the names. Here we use an existing lookup table. Should be used in predicting.

**Usage**

```r
create_index_with_existing_lookup_table(d, player0, player1, lookup_table)
```

**Arguments**

- `d`: A data frame containing the observations. The other parameters specify the name of the columns
- `player0`: The name of the column of data data contains player0
- `player1`: The name of the column of data data contains player0
- `lookup_table`: lookup_table a lookup table data frame
**create_predictor_matrix_with_player_lookup_table**

**Value**
A dataframe with the additional columns 'player0_index' and 'player1_index' that contains the indexes

**create_predictors_lookup_table**
Receives a vector with predictors strings (the column names) and returns a predictor_lookup_table

**Description**
Receives a vector with predictors strings (the column names) and returns a predictor_lookup_table

**Usage**
create_predictors_lookup_table(predictors_columns)

**Arguments**
predictors_columns

  a vector with strings containing the columns for the predictors

**Value**
A matrix to be used in stan

**create_predictor_matrix_with_player_lookup_table**
Receives a predictor dataframe, a string with the column of the player, a vector of strings with the columns for the predictors and a lookup table and returns an ordered matrix for Stan To be used with the predictors data frame

**Description**
Receives a predictor dataframe, a string with the column of the player, a vector of strings with the columns for the predictors and a lookup table and returns an ordered matrix for Stan To be used with the predictors data frame

**Usage**
create_predictor_matrix_with_player_lookup_table(
  d,
  player,
  predictors_columns,
  lookup_table
)
Arguments

d a data frame of the predictors
player The name of the column of data data contains the player
predictors_columns a vector with strings containing the columns for the predictors
lookup_table a lookup table data frame

Value

A matrix to be used in stan

Description

Expand aggregated data Several datasets for the Bradley-Terry Model aggregate the number of wins for each player in a different column. The models we provide are intended to be used in a long format. A single result for each contest. This function expands datasets that have aggregated data into this long format.

Usage

expand_aggregated_data(d, player0, player1, wins0, wins1, keep)

Arguments

d a data frame
player0 string with column name of player0
player1 string with column name of player1
wins0 string with column name of the number of wins of player 0
wins1 string with column name of the number of wins of player 1
keep an array of strings with the name of columns we want to keep in the new data frame (and repeat in every expanded row)

Value

a data frame with the expanded dataset. It will have the columns player1, player0, y, the keep columns, and a rowid column (to make each row unique)
Results

# Creating a simple data frame with only one row to illustrate how the function works
df1 <- tibble::tribble(~player0, ~player1, ~wins0, ~wins1, ~cluster, 'A', 'B', 4, 3, 'c1')
df2 <- expand_aggregated_data(df1, 'player0', 'player1', 'wins0', 'wins1', keep=c('cluster'))
print(df2)

get_hpdi_parameters

Return the mean and the HPDI of the parameters of the model

Description

Return a data frame with the mean and with high and low 95% hpd interval for all parameters of the model

Usage

get_hpdi_parameters(bpc_object)

Arguments

bpc_object a bpc object

Value

a data frame containing a column with the parameters, a column with mean and two columns with higher and lower hpd

Examples

m <- bpc(data = tennis_agresti, player0 = 'player0', player1 = 'player1', result_column = 'y', model_type = 'bt', solve_ties = 'none')
hpdi <- get_hpdi_parameters(m)
print(hpdi)
get_loo

Tiny wrapper for the PSIS-LOO-CV method from the loo package.

Description

This is used to evaluate the fit of the model using entropy criteria

Usage

get_loo(bpc_object)

Arguments

bpc_object a bpc object

Value

a loo object

References


Examples

m<-bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
l<-get_loo(m)
print(l)

get_model_parameters

Return all the name of parameters in a model from a bpc_object. Here we exclude the log_lik and the lp__ since they are not parameters of the model

Description

Return all the name of parameters in a model from a bpc_object. Here we exclude the log_lik and the lp__ since they are not parameters of the model
get_probabilities

Usage

get_model_parameters(bpc_object)

Arguments

bpc_object a bpc object

Value

a vector with the name of the parameters

get_probabilities

Get the empirical win/draw probabilities based on the ability/strength parameters. Instead of calculating from the probability formula given from the model we create a predictive posterior distribution for all pair combinations and calculate the posterior wins/loose/draw The function returns the mean value of win/loose/draw for the player i. To calculate for player j the probability is 1-p_i

Description

Get the empirical win/draw probabilities based on the ability/strength parameters. Instead of calculating from the probability formula given from the model we create a predictive posterior distribution for all pair combinations and calculate the posterior wins/loose/draw The function returns the mean value of win/loose/draw for the player i. To calculate for player j the probability is 1-p_i

Usage

get_probabilities(bpc_object, n = 1000)

Arguments

bpc_object a bpc object

n number of samples to draw from the posterior

Value

a list with data frame table with the respective probabilities and a matrix with the corresponding posterior
get_rank_of_players

Examples

m<-bpc(data = tennis_agresti,
player0 = 'player0",
player1 = 'player1",
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
prob<-get_probabilities(m)
print(prob$Table)

get_rank_of_players(bpc_object, n = 1000)

Arguments

bp_object a bpc object
n Number of times we will sample the posterior

Value

a data frame. This data frame contains the median of the rank, the mean, the standard deviation and column with a list containing all the posterior values for the rank

Examples

m<-bpc(data = tennis_agresti,
player0 = 'player0",
player1 = 'player1",
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
rank_m<-get_rank_of_players(m,n=100)
rank_table <- dplyr::select(rank_m,-MeanRank,-StdRank,-PosteriorRank)
print(rank_table)

get_rank_of_players Generate a ranking of the ability based on sampling the posterior distribution of the ranks.

Description

To print this object you should remove the last column PosteriorRank since it contain the whole posterior distribution for each case

Usage

get_rank_of_players(bpc_object, n = 1000)

Arguments

bp_object a bpc object
n Number of times we will sample the posterior

Value

a data frame. This data frame contains the median of the rank, the mean, the standard deviation and column with a list containing all the posterior values for the rank

Examples

m<-bpc(data = tennis_agresti,
player0 = 'player0",
player1 = 'player1",
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
rank_m<-get_rank_of_players(m,n=100)
rank_table <- dplyr::select(rank_m,-MeanRank,-StdRank,-PosteriorRank)
print(rank_table)
get_sample_posterior  Get the posterior samples for a parameter of the model.

Description
Return a data frame with the posterior samples for the parameters of the model

Usage
get_sample_posterior(bpc_object, par = "lambda", n = 1000)

Arguments
- bpc_object: a bpc object
- par: name of the parameters to predict
- n: how many times are we sampling? Default 1000

Value
Return a data frame with the posterior samples for the parameters. One column for each parameter, one row for each sample

Examples
```r
m<-bpc(data = tennis_agresti, 
player0 = 'player0', 
player1 = 'player1', 
result_column = 'y', 
model_type = 'bt', 
solve_ties = 'none')
s <- get_sample_posterior(m, par='lambda', n=100)
print(head(s))
```

get_stanfit  Retrieve the stanfit object generated by rstan.

Description
This object can be used with any other function or package that uses stanfit objects from rstan

Usage
get_stanfit(bpc_object)
**get_stanfit_summary**

**Arguments**

- **bpc_object**  
  a bpc object

**Value**

- a stanfit object

**Examples**

```r
m <- bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
stanfit <- get_stanfit(m)
print(class(stanfit))
```

---

**get_stanfit_summary**  
Get stanfit summary table of all parameters excluding log_lik.

**Description**

Important to investigate the neff and the Rhat from the MCMC. This excludes the log_lik parameter.

**Usage**

```r
get_stanfit_summary(bpc_object)
```

**Arguments**

- **bpc_object**  
  a bpc object

**Value**

- a data frame with the summary including quantiles, Rhat and neff

**Examples**

```r
m <- bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
s <- get_stanfit_summary(m)
```
get_waic

print(s)

tiny wrapper for the WAIC method from the loo package.

Description

This is used to evaluate the fit of the model using the Watanabe-Akaike Information criteria.

Usage

general(bpc_object)

Arguments

bpc_object: a bpc object

Value

A loo object

References


Examples

```r
m <- bpc(data = tennis_agresti, 
player0 = 'player0', 
player1 = 'player1', 
result_column = 'y', 
model_type = 'bt', 
solve_ties = 'none')
waic <- get_waic(m)
print(waic)
```
**HPDI_from_stanfit**

*Calculate HPDI for all parameters from a stanfit object Here we use the coda package*

---

**Description**

Calculate HPDI for all parameters from a stanfit object Here we use the coda package

**Usage**

`HPDI_from_stanfit(stanfit)`

**Arguments**

- `stanfit` a stanfit object retrieved from a bpc object

**Value**

a data frame with the HPDI calculated from the coda package

**References**


---

**HPD_higher_from_column**

*Returns the higher value of the HPD interval for a data frame column*

---

**Description**

Returns the higher value of the HPD interval for a data frame column

**Usage**

`HPD_higher_from_column(column, credMass = 0.95)`

**Arguments**

- `column` the data to calculate the HPDI
- `credMass` Credibility mass for the interval (area contained in the interval)

**Value**

the value of the higher HPD interval for that column
References


---

**HPD_lower_from_column**  
*Returns the lower value of the HPD interval for a data frame column*

Description

Returns the lower value of the HPD interval for a data frame column

Usage

```
HPD_lower_from_column(column, credMass = 0.95)
```

Arguments

- `column`: the data to calculate the HPDI
- `credMass`: Credibility mass for the interval (area contained in the interval)

Value

the value of the lower HPD interval for that column

References


---

**inv_logit**  
*Inverse logit function*

Description

Inverse logit function

Usage

```
inv_logit(x)
```

Arguments

- `x`: is a real -inf to inf
launch_shinystan

Value

a value between 0 and 1

References

https://en.wikipedia.org/wiki/Logit

Examples

inv_logit(5)
inv_logit(-5)
inv_logit(0)

launch_shinystan

Tiny wrapper to launch a shinystan app to investigate the MCMC.

Description

It launches a shinystan app automatically in the web browser

Usage

launch_shinystan(bpc_object)

Arguments

bpc_object a bpc object

Examples

m<-bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
launch_shinystan(m)
### logit

**Logit function**

**Description**
Logit function

**Usage**

```r
logit(x)
```

**Arguments**

- `x`  
  `p` is a probability 0 to 1

**Value**

A value between `-inf` and `inf`

**References**


**Examples**

```r
logit(0.5)
logit(0.2)
```

### match_cluster_names_to_cluster_lookup_table

**Receives a column with cluster names and returns a data frame with the relevant index column based on a given cluster lookup table**

**Description**

Receives a column with cluster names and returns a data frame with the relevant index column based on a given cluster lookup table

**Usage**

```r
match_cluster_names_to_cluster_lookup_table(d, cluster, cluster_lookup_table)
```

**Arguments**

- `d`  
  A data frame
- `cluster`  
  The name of the column of data data contains player0
- `cluster_lookup_table`  
  A lookup table for the cluster
**match_player_names_to_lookup_table**

Receives two columns with player names and returns a data frame with the relevant index columns based on a given lookup table.

**Description**

Receives two columns with player names and returns a data frame with the relevant index columns based on a given lookup table.

**Usage**

```r
match_player_names_to_lookup_table(d, player0, player1, lookup_table)
```

**Arguments**

- `d` : a data frame
- `player0` : The name of the column of data data contains player0
- `player1` : The name of the column of data data contains player1
- `lookup_table` : a lookup table data frame

**Value**

A dataframe with the additional columns `player0_index` and `player1_index` that contains the indexes

---

**optimization_algorithms**

Dataset containing an example of the performance of different optimization algorithms against different benchmark functions. This is a reduced version of the dataset presented at the paper: "Statistical Models for the Analysis of Optimization Algorithms with Benchmark Functions.". For details on how the data was collected we refer to the paper.

**Description**

Dataset containing an example of the performance of different optimization algorithms against different benchmark functions. This is a reduced version of the dataset presented at the paper: "Statistical Models for the Analysis of Optimization Algorithms with Benchmark Functions.". For details on how the data was collected we refer to the paper.
Usage

optimization_algorithms

Format

This is the expansion of the data where each row contains 1 match only

- Algorithm: name of algorithm
- Benchmark: name of the benchmark problem
- TrueRewardDifference: Difference between the minimum function value obtained by the algorithm and the known global minimum
- Ndimensions: Number of dimensions of the benchmark problem
- MaxFevalPerDimensions: Maximum allowed budget for the algorithm per dimensions of the benchmark problem
- simNumber: id of the simulation. Indicates the repeated measures of each algorithm in each benchmark

Source


predict.bpc

Predict results for new data.

Description

This S3 function receives the bpc model and a data frame containing the same columns as the one used to fit the model. It returns another data frame with with the same columns of the new data and n additional columns representing a posterior predictive distribution. See the vignettes for a larger examples with the usage of this function

Usage

## S3 method for class 'bpc'
predict(object, newdata, predictors = NULL, n = 100, return_matrix = F, ...)

Arguments

object         a bpc object
newdata        a data frame that contains columns with the same names as used to fit the data in the model.
predictors: A data frame that contains the players predictors values when using a generalized model. Should be set only if using the generalized models. Only numeric values are accepted. Booleans are accepted but will be cast into integers. The first column should be for the player name, the others will be the predictors. The column names will be used as name for the predictors.

n: number of time we will iterate and get the posterior. default is 100 so we dont get too many.

return_matrix: should we return only a matrix with the predictive values. Default F. Use this to combine with predictive posterior plots in bayesplot. This parameter also ignores the n parameter above since it passes all the predictions from stan.

Value

A dataframe or a matrix depending on the return_matrix parameter.

Examples

```r
m<-bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
predict(m,newdata=tennis_agresti)
```

---

print.bpc  

Print method for the bpc object.

Description

This S3 functions only prints the mean and the HDPI values of all the parameters in the model.

Usage

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

Arguments

- `x`: a bpc object
- `digits`: number of decimal digits in the table
- `...`: additional parameters for the generic print function
Examples

```r
m <- bpc(data = tennis_agresti, 
player0 = 'player0', 
player1 = 'player1', 
result_column = 'y', 
model_type = 'bt', 
solve_ties = 'none') 
# print(m)
```

replace_parameter_index_with_names

*Replace the name of the parameter from index to name using a lookup_table Receives a data frame and returns a dataframe.*

Description

Replace the name of the parameter from index to name using a lookup_table Receives a data frame and returns a dataframe.

Usage

```r
replace_parameter_index_with_names(
  d, 
  column, 
  par, 
  lookup_table, 
  cluster_lookup_table = NULL, 
  predictors_lookup_table = NULL
)
```

Arguments

d dataframe
column name of the column
par name of the parameter
lookup_table lookup table of the players
cluster_lookup_table a lookup table of the predictors
predictors_lookup_table a lookup table for the predictors

Value

a data.frame where we change the names in the variable column to the corresponding parameter_name from the lookup table
sample_stanfit

Return a data frame by resampling the posterior from a stanfit. Here we select a parameter, retrieve all the posterior from the stanfit and then we resample this posterior n times.

Usage

```r
sample_stanfit(stanfit, par, n = 100)
```

Arguments

- `stanfit`: stanfit object
- `par`: parameter name
- `n`: number of samples

Value

A dataframe containing the samples of the parameter. Each column is a parameter (in order of the index), each row is a sample.

References


summary.bpc

Summary of the model bpc model.

Description

- Table 1: Contains the parameter estimates and respective HPD interval
- Table 2: Contains the posterior probability for the combination of all players
- Table 3: Contains the ranking of the players' abilities based on the posterior distribution of the ranks

Usage

```r
## S3 method for class 'bpc'
summary(object, digits = 2, ...)
```
Arguments

object  bpc object
digits  number of decimal digits in the table
...  additional parameters for the generic summary function

Examples

m <- bpc(data = tennis_agresti,
player0 = 'player0',
player1 = 'player1',
result_column = 'y',
model_type = 'bt',
solve_ties = 'none')
summary(m)

Description

This is the expansion of the tennis data from Agresti (2003) p.449 This data refers to matches for several women tennis players during 1989 and 1990

Usage

tennis_agresti

Format

This is the expansion of the data where each row contains 1 match only

- player0: name of player0
- player1: name of player1
- y: corresponds to the result of the match: 0 if player0 won, 1 if player1 won.
- id: is a column to make each row unique in the data. It does not have any particular interpretation

Source

Index

* data
  brasil_soccer_league, 6
  optimization_algorithms, 31
  tennis_agresti, 36

bpc, 3
bpcs-package, 3
brasil_soccer_league, 6

check_if_there_are_na, 7
check_if_there_are_ties, 8
check_numeric_predictor_matrix, 8
check_predictors_df_contains_all_players, 9
check_result_column, 9
check_z_column, 10
compute_scores, 10
compute_ties, 11
create_array_of_par_names, 12
create_bpc_object, 12
create_cluster_index, 13
create_cluster_index_with_existing_lookup_table, 14
create_index, 15
create_index_cluster_lookup_table, 15
create_index_lookup_table, 16
create_index_predictors_with_lookup_table, 17
create_index_with_existing_lookup_table, 17
create_predictor_matrix_with_player_lookup_table, 18
create_predictors_lookup_table, 18

expand_aggregated_data, 19

get_hpdi_parameters, 20
get_loo, 21
get_model_parameters, 21
get_probabilities, 22
get_rank_of_players, 23
get_sample_posterior, 24
get_stanfit, 24
get_stanfit_summary, 25
get_waic, 26

HPD_higher_from_column, 27
HPD_lower_from_column, 28
HPDI_from_stanfit, 27

inv_logit, 28
launch_shinystan, 29
logit, 30

match_cluster_names_to_cluster_lookup_table, 30
match_player_names_to_lookup_table, 31
optimization_algorithms, 31
predict.bpc, 32
print.bpc, 33
replace_parameter_index_with_names, 34

sample_stanfit, 35
summary.bpc, 35
tennis_agresti, 36