Package 'funkycells'

August 9, 2023

Title Functional Data Analysis for Multiplexed Cell Images

Version 1.1.1

Description Compare variables of interest between (potentially large numbers of) spatial interactions and meta-variables. Spatial variables are summarized using K, or other, functions, and projected for use in a modified random forest model. The model allows comparison of functional and non-functional variables to each other and to noise, giving statistical significance to the results. Included are preparation, modeling, and interpreting tools along with example datasets, as described in VanderDoes et al., (2023) <doi:10.1101/2023.07.18.549619>.

License GPL (>= 3)

URL https://github.com/jrvanderdoes/funkycells, https://jrvanderdoes.github.io/funkycells/

BugReports https://github.com/jrvanderdoes/funkycells/issues

Depends R (>= 2.10)

Imports fda, ggplot2, rpart, spatstat.explore, spatstat.geom, stats, stringr, tidyr

Suggests knitr, pROC, rmarkdown, scales, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Jeremy VanderDoes [aut, cre, cph] (<https://orcid.org/0009-0001-9885-3073>), Jack Hywood [aut] (<https://orcid.org/0000-0002-2028-2629>), Gregory Rice [aut]

Maintainer Jeremy VanderDoes <jeremy.vanderdoes@gmail.com>

Repository CRAN

Date/Publication 2023-08-09 16:00:06 UTC

R topics documented:

computePseudoROCCurves 2
funkyForest 3
funkyModel 4
getCountData 6
getKFunction 7
getKsPCAData 9
plotPP 10
plot_K_functions 11
predict_funkyForest 13
simulateMeta 14
simulatePP 16
TNBC 17
TNBC_meta 18
TNBC_pheno 18

Index **20**

`computePseudoROCCurves` *Compute Pseudo-ROC Curves*

Description

An receiver operating characteristic (ROC) curve is a curve showing the performance of a classification model at all classification thresholds. True ROC can only be computed for two-options, but we can consider each classification, i.e. prediction, correct or incorrect and overlay the curves. Note this means the lines may cover each other and be difficult to see.

Usage

```
computePseudoROCCurves(trueOutcomes, modelPercents)
```

Arguments

`trueOutcomes`	Vector of the true results
`modelPercents`	Data.frame with columns named after the true outcomes, giving the percent of selecting that outcome. This is what is returned predict.RandomForest_PC with type='all' in object `PredPerc[-1]` (first column is the predictions).

Details

This function requires the package 'pROC' to be installed.

Value

ggplot object containing the ROC curves.

Examples

```
percents <- data.frame(c(0.980, 0.675, 0.878, 0.303, 0.457, 0.758,
                         0.272, 0.524, 0.604, 0.342, 0.214, 0.569,
                         0.279, 0.128, 0.462, 0.098, 0.001, 0.187),
                       c(0.005, 0.160, 0.100, 0.244, 0.174, 0.143,
                         0.652, 0.292, 0.040, 0.312, 0.452, 0.168,
                         0.173, 0.221, 0.281, 0.029, 0.005, 0.057),
                       c(0.015, 0.165, 0.022, 0.453, 0.369, 0.099,
                         0.076, 0.084, 0.156, 0.346, 0.334, 0.263,
                         0.548, 0.651, 0.257, 0.873, 0.994, 0.756))
colnames(percents) <- c('0','1','2')
proc <- computePseudoROCCurves(c(0, 0, 0, 0, 0, 0,
                                 1, 1, 1, 1, 1, 1,
                                 2, 2, 2, 2, 2, 2),
                               percents)
```

`funkyForest` *Compute a Modified Random Forest Model*

Description

This function creates a modified random forest model for principal component and meta-data. This can be useful to get a final model, but we recommend use of randomForest_CVPC in general, which includes the final model.

Usage

```
funkyForest(
  data,
  outcome = colnames(data)[1],
  unit = colnames(data)[2],
  nTrees = 500,
  varImpPlot = TRUE,
  metaNames = NULL,
  keepModels = TRUE,
  varSelPercent = 0.8,
  method = "class"
)
```

Arguments

`data`	Data.frame of outcome and predictors. The predictors include groups of variables which are finite projections of a higher dimensional variables as well as single meta-variables.

	Any replicate data, i.e. repeated observations, should already be handled. The unit column is needed just to drop data (so pre-removing and giving NULL works). Typically use the results from getKsPCAData, potentially with meta-variables attached.
outcome	(Optional) String indicating the outcome column name in data. Default is the first column of data.
unit	(Optional) String indicating the unit column name in data. Default is the second column of data.
nTrees	(Optional) Numeric indicating the number of trees to use in the random forest model. Default is 500.
varImpPlot	(Optional) Boolean indicating if variable importance plots should also be returned with the model. Default is TRUE.
metaNames	(Optional) Vector with the column names of data that correspond to metavariables. Default is NULL.
keepModels	(Optional) Boolean indicating if the individual models should be kept. Can get large in size. Default is TRUE as it is needed for predictions.
varSelPercent	(Optional) Numeric in (0,1) indicating (approx) percentage of variables to keep for each tree. Default is 0.8.
method	(Optional) Method for rpart tree to build random forest. Default is "class". Currently this is the only tested method. This will be expanded in future releases.

Value

A list with entries

1. varImportanceData: Data.frame for variable importance information.
2. (Optional) model: List of CART that builds the random forest model.
3. (Optional) varImportancePlot: Variable importance plots.

Examples

```
ff <- funkyForest(
  data = TNBC[, c(1:8, ncol(TNBC))],
  outcome = "Class", unit = "Person",
  metaNames = c("Age")
  )
```

`funkyModel` *Fit a Modified Random Forest Model with Bounds and Alignment*

Description

The function fits a modified random forest model to principal components of spatial interactions as well as meta-data. Additionally permutation and cross-validation is employed to improve understanding of the data.

Usage

```
funkyModel(
  data,
  K = 10,
  outcome = colnames(data)[1],
  unit = colnames(data)[2],
  metaNames = NULL,
  synthetics = 100,
  alpha = 0.05,
  silent = FALSE,
  rGuessSims = 500,
  subsetPlotSize = 25,
  nTrees = 500,
  method = "class"
)
```

Arguments

data	Data.frame of outcome and predictors. The predictors include groups of variables which are finite projections of a higher dimensional variables as well as single meta-variables. Any replicate data, i.e. repeated observations, should already be handled. The unit column is needed just to drop data (so pre-removing and giving NULL works). Typically use the results from getKsPCAData, potentially with meta-variables attached.
K	(Optional) Numeric indicating the number of folds to use in K-fold cross-validation. The default is 10.
outcome	(Optional) String indicating the outcome column name in data. Default is the first column of data.
unit	(Optional) String indicating the unit column name in data. Default is the second column of data.
metaNames	(Optional) Vector indicating the meta-variables to be considered. Default is NULL.
synthetics	(Optional) Numeric indicating the number of synthetics for variables (one set of sythethics for functional variables and one for each meta-variable). If 0 are used, the data cannot be aligned properly. Default is 100.
alpha	(Optional) Numeric in (0,1) indicating the significance used throughout the analysis. Default is 0.05.
silent	(Optional) Boolean indicating if output should be suppressed when the function is running. Default is FALSE.
rGuessSims	(Optional) Numeric value indicating the number of simulations used for guessing and creating the guess estimate on the plot. Default is 500.
subsetPlotSize	(Optional) Numeric indicating the number of top variables to include in a subset graph. If this is larger than the total number then no subset graph will be produced. Default is 25.

nTrees	(Optional) Numeric indicating the number of trees to use in the random forest model. Default is 500.
method	(Optional) Method for rpart tree to build random forest. Default is "class". Currently this is the only tested method. This will be expanded in future releases.

Value

List with the following items:

1. model: The funkyForest Model fit on the entire given data.
2. VariableImportance: Data.frame with the results of variable importance indices from the models and CV. The columns are var, est, sd, and cvSD.
3. AccuracyEstimate: Data.frame with model accuracy estimates: out-of-bag accuracy (OOB), biased estimate (bias), and random guess (guess). The columns are OOB, bias, and guess.
4. NoiseCutoff: Numeric indicating noise cutoff (vertical line).
5. InterpolationCutoff: Vector of numerics indicating the interpolation cutoff (curved line).
6. AdditionalParams: List of additional parameters for reference: Alpha and subsetPlotSize.
7. viPlot: ggplot2 object for vi plot with standardized results. It displays ordered underlying functions and meta-variables with point estimates, sd, noise cutoff, and interpolation cutoff all based on variable importance values.
8. subset_viPlot: (Optional) ggplot2 object for vi plot with standardized results and only top subsetPlotSize variables. It displays ordered underlying functions and meta-variables with point estimates, sd, noise cutoff, and interpolation cutoff all based on variable importance values.

Examples

```
# Parameters are reduced beyond recommended levels for speed
fm <- funkyModel(
  data = TNBC[, c(1:8, ncol(TNBC))],
  outcome = "Class", unit = "Person",
  metaNames = c("Age"),
  nTrees = 5, synthetics = 10,
  silent = TRUE
)
```

---

getCountData *Get Agent Count Data*

Description

This function gets the average percent agent counts per replicate, if there are replicates (i.e. replicate is not NULL), then the agent percents are calculated for each replicate and these percentages are averaged.

Usage

```
getCountData(
  agent_data,
  outcome,
  unit,
  replicate = NULL,
  type = "type",
  data_append = NULL
)
```

Arguments

agent_data	Data.frame of agent data information, with columns as defined in subsequent parameters
outcome	String of the column name in data indicating the outcome or response.
unit	String of the column name in data indicating a unit or base thing. Note this unit may have replicates.
replicate	(Optional) String of the column name in data indicating the replicate id. Default is NULL.
type	(Optional) String of the column name in data indicating the type. Default is type.
data_append	(Optional) Data.frame with outcome, patient that the results can be appended to if desired. Default is NULL.

Value

List with two elements:

- dat: Data.frame with outcome, unit, data_append, and the count data. Columns of the count data are named after the type and are given in the next list entry.
- agents: Vector of the the types, i.e. the column names for the new count data. This can be treated as meta data for funkyForest.

Examples

```
data_ct <- getCountData(TNBC_pheno[TNBC_pheno$Phenotype %in% c('Tumor','B'),],
                        outcome="Class", unit="Person",type="Phenotype")
```

---

getKFunction *Get K function*

Description

This function computes the K function between the two agents for each unit, potentially averaging over replicates, or repeated measures.

Usage

```
getKFunction(
  data,
  agents,
  unit,
  replicate = NULL,
  rCheckVals = NULL,
  xRange = NULL,
  yRange = NULL,
  edgeCorrection = "isotropic"
)
```

Arguments

data	Dataframe with column titles for at least x, y, agents, and unit. For consistency (and avoiding errors), use that order. Additionally, replicate can be added.
agents	Two value vector indicating the two agents to use for the K function, the first to the second. These should be in the unit column.
unit	String of the column name in data indicating a unit or base thing. Note this unit may have replicates.
replicate	(Optional) String of the column name in data indicating the unique replicates, or repeated measures.
rCheckVals	(Optional) Numeric vector indicating the radius to check. Note, if note specified, this could take a lot of memory, particularly with many units and replicates.
xRange, yRange	(Optional) Two value numeric vector indicating the min and max x / y values. Note this is re-used for all images. The default just takes the min and max from each image. This allows different sized images, but the edges are defined by some agent location.
edgeCorrection	(Optional) String indicating type of edgeCorrection(s) to apply when computing the K functions. Options include: "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "periodic", "none", "best" or "all" selects all options.

Value

data.frame with the first column being the checked radius and the remaining columns relating to the K function for each unit at those points. If a K function could not be computed, perhaps due to lack of data, an NA is returned for the K function.

Examples

```
KFunction <- getKFunction(
  agents = c("B", "Tumor"), unit = "Person",
  data = TNBC_pheno[TNBC_pheno$Person == 1, -1],
  rCheckVals = seq(0, 50, 1),
  edgeCorrection = "isotropic"
)
```

getKsPCAData *Get K Functions and Compute Principal Components*

Description

This function computes K functions from point process data then converts it into PCs. Note, if there are replicates, i.e. multiple observations per unit, the K functions will be a weighted average based on the number of the first agents.

Usage

```
getKsPCAData(
  data,
  outcome = colnames(data)[1],
  unit = colnames(data)[5],
  replicate = NULL,
  rCheckVals = NULL,
  nPCs = 3,
  agents_df = as.data.frame(expand.grid(unique(data[, 4]), unique(data[, 4]))),
  xRange = NULL,
  yRange = NULL,
  edgeCorrection = "isotropic",
  nbasis = 21,
  silent = FALSE,
  displayTVE = FALSE
)
```

Arguments

data	Data.frame with column titles for at least outcome, x, y, agents, and unit. For consistency (and avoiding errors), use that order. Additionally, replicate can be added.
outcome	(Optional) String of the column name in data indicating the outcome or response. Default is the 1st column.
unit	(Optional) String of the column name in data indicating a unit or base thing. Note this unit may have replicates. Default is the 4th column.
replicate	(Optional) String of the column name in data indicating the replicate id. Default is NULL.
rCheckVals	(Optional) Numeric vector indicating the radius to check. Note, if not specified, this could take a lot of memory, particularly with many units and replicates.
nPCs	(Optional) Numeric indicating the number of principal components.
agents_df	(Optional) Two-column data.frame. The first for agent 1 and the second for agent 2. Both should be in data agents column. This determines which K functions to compute. Default is to compute all, but may be misspecified if the data is in a different order.

xRange, yRange	(Optional) Two value numeric vector indicating the min and max x/y values. Note this is re-used for all replicates. The default just takes the min and max x from each replicate. This allows different sized images, but the edges are defined by some agent location.
edgeCorrection	(Optional) String indicating type of edgeCorrection(s) to apply when computing the K functions. Options include: "border", "bord.modif", "isotropic", "Ripley", "translate", "translation", "periodic", "none", "best" or "all" selects all options.
nbasis	(Optional) Numeric indicating number of basis functions to fit K functions in order to compute PCA. Current implementation uses a b-spline basis.
silent	(Optional) Boolean indicating if progress should be printed.
displayTVE	(Optional) Boolean to indicate if total variance explained (TVE) should be displayed. Default is FALSE.

Value

Data.frame with the outcome, unit and principle components of computed K functions.

Examples

```
dataPCA_pheno <- getKsPCAData(
  data = TNBC_pheno, unit = "Person",
  agents_df = data.frame(rep("B", 2), c("Tumor", "Fake")),
  nPCs = 3,
  rCheckVals = seq(0, 50, 1),
  displayTVE = TRUE
)
```

---

plotPP *Plot Spatial Point Process*

Description

This function is used to plot a spatial point process. This does not split data and instead puts all given data on a single plot.

Usage

```
plotPP(
  data,
  colorGuide = NULL,
  ptSize = 1,
  xlim = c(min(data[, 1]), max(data[, 1])),
  ylim = c(min(data[, 2]), max(data[, 2])),
  dropAxes = FALSE,
  layerBasedOnFrequency = TRUE,
  colors = NULL
)
```

Arguments

data	Data.frame with x, y, and agent type (in that order)
colorGuide	(Optional) String for 'guides(color=)' in ggplot2. Usually NULL or 'none' is sufficient, but ggplot2::guide_legend() can also be used for more custom results. Default is NULL.
ptSize	(Optional) Numeric indicating point size. Default is 1.
xlim	(Optional) Two value numeric vector indicating the size of the region in the x-direction. Default is c(min(x), max(x)).
ylim	(Optional) Two value numeric vector indicating the size of the region in the y-direction. Default is c(min(y), max(y)).
dropAxes	(Optional) Boolean indicating if the x and y axis title and labels should be dropped. Default is FALSE.
layerBasedOnFrequency	(Optional) Boolean indicating if the data should be layer based on the number of agents of the type. Default is TRUE.
colors	(Optional) Vector of colors for the points. Default is NULL, or ggplot2 selected colors.

Value

ggplot2 plot of the spatial point process.

Examples

```
ppplot <- plotPP(
  TNBC_pheno[
    TNBC_pheno$Person == 1,
    c("cellx", "celly", "Phenotype")
  ],
  colorGuide = "none"
)
```

---

plot_K_functions *Compare K Functions Between outcomes*

Description

This function plots K functions from different outcomes for comparison. Group means are included as bold lines. Additionally a reference line for a spatially random process can be included.

Usage

```
plot_K_functions(data, inc.legend = TRUE, inc.noise = FALSE)
```

Arguments

data	Data.frame with named columns r, K, unit, and outcome. The column r indicates the radius of checked K function, K indicates the K function value, unit specifies the unique K function, and outcome indicates the unit outcome.
inc.legend	(Optional) Boolean indicating if the legend should be given. This will also include numbers to indicate if any K functions are missing. The default is TRUE.
inc.noise	(Optional) Boolean indicating if a gray, dashed line should be included to show what spatially random noise would be like. The default is FALSE.

Value

ggplot2 object showing the K function with a superimposed average.

Examples

```
# Example 1
tmp <- getKFunction(TNBC_pheno[TNBC_pheno$Class == 0, -1],
  agents = c("Tumor", "Tumor"),
  unit = "Person",
  rCheckVals = seq(0, 50, 1)
)
tmp1 <- getKFunction(TNBC_pheno[TNBC_pheno$Class == 1, -1],
  agents = c("Tumor", "Tumor"),
  unit = "Person",
  rCheckVals = seq(0, 50, 1)
)
tmp_1 <- tidyr::pivot_longer(data = tmp, cols = K1:K18)
tmp1_1 <- tidyr::pivot_longer(data = tmp1, cols = K1:K15)

data_plot <- rbind(
  data.frame(
    "r" = tmp_1$r,
    "K" = tmp_1$value,
    "unit" = tmp_1$name,
    "outcome" = "0"
  ),
  data.frame(
    "r" = tmp1_1$r,
    "K" = tmp1_1$value,
    "unit" = paste0(tmp1_1$name, "_1"),
    "outcome" = "1"
  )
)

pk1 <- plot_K_functions(data_plot)

# Example 2
tmp <- getKFunction(TNBC_pheno[TNBC_pheno$Class == 0, -1],
  agents = c("Tumor", "B"), unit = "Person",
  rCheckVals = seq(0, 50, 1)
)
```

```
tmp1 <- getKFunction(TNBC_pheno[TNBC_pheno$Class == 1, -1],
  agents = c("Tumor", "B"), unit = "Person",
  rCheckVals = seq(0, 50, 1)
)

tmp_1 <- tidyr::pivot_longer(data = tmp, cols = K1:K18)
tmp1_1 <- tidyr::pivot_longer(data = tmp1, cols = K1:K15)

data_plot <- rbind(
  data.frame(
    "r" = tmp_1$r,
    "K" = tmp_1$value,
    "unit" = tmp_1$name,
    "outcome" = "0"
  ),
  data.frame(
    "r" = tmp1_1$r,
    "K" = tmp1_1$value,
    "unit" = paste0(tmp1_1$name, "_1"),
    "outcome" = "1"
  )
)

pk2 <- plot_K_functions(data_plot)
```

`predict_funkyForest` *Predict a funkyForest*

Description

This function gets the predicted value from a funkyForest model.

Usage

```
predict_funkyForest(model, data_pred, type = "all", data = NULL)
```

Arguments

`model`	funkyForest model. See funkyForest. A list of CART models from rpart. Additionally this is given in funkyModel.
`data_pred`	data.frame of the data to be predicted.
`type`	(Optional) String indicating type of analysis. Options are pred or all. The choice changes the return to best fit intended use.
`data`	(Optional) Data.frame of full data. The data used to fit the model will be extracted (by row name).

Value

The returned data depends on type:

- type='pred': returns a vector of the predictions
- type='all': returns a vector of the predictions

Examples

```
data_pp <- simulatePP(
  agentVarData =
    data.frame(
      "outcome" = c(0, 1),
      "A" = c(0, 0),
      "B" = c(1 / 50, 1 / 50)
    ),
  agentKappaData = data.frame(
    "agent" = c("A", "B"),
    "clusterAgent" = c(NA, "A"),
    "kappa" = c(10, 5)
  ),
  unitsPerOutcome = 5,
  replicatesPerUnit = 1,
  silent = FALSE
)
pcaData <- getKsPCAData(data_pp,
  replicate = "replicate",
  xRange = c(0, 1), yRange = c(0, 1), silent = FALSE
)
RF <- funkyForest(data = pcaData[-2], nTrees = 5) #
pred <- predict_funkyForest(
  model = RF$model, type = "all",
  data_pred = pcaData[-2],
  data = pcaData[-2]
)
```

---

simulateMeta *Simulate Meta Variables*

Description

This function simulates meta-variables with varying distributions to append to some data.

Usage

```
simulateMeta(
  data,
  outcome = colnames(data)[1],
 metaInfo = data.frame(var = c("randUnif", "randBin", "rNorm", "corrUnif", "corrBin",
  "corrNorm"), rdist = c("runif", "rbinom", "rnorm", "runif", "rbinom", "rnorm"),
```

```
   outcome_0 = c("0.5", "0.5", "1", "0.5", "0.6", "1"), outcome_1 = c("0.5", "0.5", "1",
   "0.75", "0.65", "1.5"), outcome_2 = c("0.5", "0.5", "1", "0.95", "0.75", "1.5"))
)
```

Arguments

data	Data.frame with the outcome and unit. Typically this also includes PCA data as it is run after computing the principle components (see examples).
outcome	(Optional) String for column title of the data's outcome. Default is the first column.
metaInfo	(Optional) Data.frame indicating the meta-variables (and properties) to generate. Default has some examples of possible options. The data.frame has a var column, rdist column, and columns for each outcome. The var column names the meta-variables, rdist indicates the distribution (options are runif, rbinom, and rnorm), and the outcome columns indicate mean of the variable for that outcome. In order to allow designation of the expected values, the following rules are imposed on each distribution: • runif: a=0, so b is modified, • rbinom: n=1, so this defines the probability • runif: variance is set to 1

Details

Notes: runif may induce useless information so don't recommend correlating it

Value

Data.frame of the original data with meta-variables appended (as columns) at the end.

Examples

```
data <- simulatePP(
  agentVarData = data.frame(
    "outcome" = c(0, 1, 2),
    "A" = c(0, 0, 0),
    "B" = c(1 / 100, 1 / 500, 1 / 1000)
  ),
  agentKappaData = data.frame(
    "agent" = c("A", "B"),
    "clusterAgent" = c(NA, "A"),
    "kappa" = c(10, 3)
  ),
  unitsPerOutcome = 5,
  replicatesPerUnit = 1
)
pcaData <- getKsPCAData(
  data = data, replicate = "replicate",
  xRange = c(0, 1), yRange = c(0, 1)
```

```
)
pcaMeta <- simulateMeta(pcaData)

## Another simple example
data <- simulateMeta(
  data.frame("outcome" = c(0, 0, 0, 1, 1, 1), "unit" = 1:6)
)
```

`simulatePP` *Simulate a Point Process*

Description

This function simulates a point pattern with optional clustering (visible and invisible). Multiple outcomes, units, and replicates are possible, e.g. a 3 stage disease (outcomes) over 20 people (units) with 3 images each (replicates).

Usage

```
simulatePP(
 agentVarData = data.frame(outcome = c(0, 1, 2), A = c(0, 0, 0), B = c(1/100, 1/500,
  1/500), C = c(1/500, 1/250, 1/100), D = c(1/100, 1/100, 1/100), E = c(1/500, 1/500,
   1/500), F = c(1/250, 1/250, 1/250)),
 agentKappaData = data.frame(agent = c("A", "B", "C", "D", "E", "F"), clusterAgent =
   c(NA, "A", "B", "C", NA, "A"), kappa = c(20, 5, 4, 2, 15, 5)),
 unitsPerOutcome = 20,
 replicatesPerUnit = 5,
 silent = FALSE
)
```

Arguments

`agentVarData`	(Optional) Data.frame describing variances with each agent type. The data.frame has a outcome column and a named column for each agent type. Currently, these names are mandatory.
`agentKappaData`	(Optional) Data.frame describing agent interactions. The data.frame has a agent column giving agent names (matching agentVarData), a clusterAgent column indicating which agent the agent clusters (put NA if the agent doesn't cluster or clusters a hidden agent / self-clusters), and a kappa column directing the number of agents of per replicate.
`unitsPerOutcome`	(Optional) Numeric indicating the number of units per outcome.
`replicatesPerUnit`	(Optional) Numeric indicating the number of replicates, or repeated measures, per unit.
`silent`	(Optional) Boolean indicating if progress output should be printed.

Value

Data.frame containing each point the defined patterns.

The data.frame has columns for outcome, x coordinate, y coordinate, agent type, unit, and replicate id.

Examples

```
data <- simulatePP(
  agentVarData = data.frame(
    "outcome" = c(0, 1),
    "A" = c(0, 0),
    "B" = c(1 / 100, 1 / 500),
    "C" = c(1 / 500, 1 / 250),
    "D" = c(1 / 100, 1 / 100),
    "E" = c(1 / 500, 1 / 500)
  ),
  agentKappaData = data.frame(
    "agent" = c("A", "B", "C", "D", "E"),
    "clusterAgent" = c(NA, "A", "B", "C", NA),
    "kappa" = c(10, 3, 2, 1, 8)
  ),
  unitsPerOutcome = 4,
  replicatesPerUnit = 1
)
```

TNBC *Triple Negative Breast Cancer Data*

Description

A funky model ready set of principle components from K functions based on triple negative breast cancer data from patients. The original data was proteins as coded in T/F values. Additionally, the age meta-variable was added.

Usage

```
TNBC
```

Format

TNBC**:**

A data frame with 33 rows and 1398 columns:

Class Outcome of each patient

Person Person for each image

NA_Si_PC1 through tumerYN_tumerYN_PC3 Principle components of the K functions for the named interactions

age Meta-variable for patient age ...

Source

https://www.angelolab.com/mibi-data

TNBC_meta *Triple Negative Breast Cancer Phenotypes*

Description

Data of meta-variable age related to triple negative breast cancer biopsies from patients.

Usage

```
TNBC_meta
```

Format

TNBC_meta**:**

A data frame with 33 rows and 2 columns:

Person Person for each image

Age Meta-variable for patient age ...

Source

https://www.angelolab.com/mibi-data

TNBC_pheno *Triple Negative Breast Cancer Phenotypes*

Description

Data of triple negative breast cancer biopsies from patients.

Usage

```
TNBC_pheno
```

Format

TNBC_pheno**:**

A data frame with 170,171 rows and 5 columns:

Class Outcome of each patient

Person Person for which each cell is related

cellx, celly The x-y coordinates of the cell

Phenotype The classified phenotype for the cecll ...

Source

https://www.angelolab.com/mibi-data

Index

∗ **datasets**
TNBC, 17
TNBC_meta, 18
TNBC_pheno, 18

computePseudoROCCurves, 2

funkyForest, 3
funkyModel, 4

getCountData, 6
getKFunction, 7
getKsPCAData, 9

plot_K_functions, 11
plotPP, 10
predict_funkyForest, 13

simulateMeta, 14
simulatePP, 16

TNBC, 17
TNBC_meta, 18
TNBC_pheno, 18