Package ‘rhoR’

September 13, 2020

Title  Rho for Inter Rater Reliability
Maintainer  Cody L Marquart <cody.marquart@wisc.edu>
Version  1.3.0.3

Description  Rho is used to test the generalization of inter rater reliability
(IRR) statistics. Calculating rho starts by generating a large number of
simulated, fully-coded data sets: a sizable collection of hypothetical
populations, all of which have a kappa value below a given threshold -- which
indicates unacceptable agreement. Then kappa is calculated on a sample from
each of those sets in the collection to see if it is equal to or higher than
the kappa in then real sample. If less than five percent of the distribution
of samples from the simulated data sets is greater than actual observed kappa,
the null hypothesis is rejected and one can conclude that if the two raters had
coded the rest of the data, we would have acceptable agreement (kappa above the
threshold).

Depends  R (>= 3.0.0)
License  GPL-3 | file LICENSE
URL  https://rhor.qe-libs.org
BugReports  https://gitlab.com/epistemic-analytics/qe-packages/rhoR/-/issues
LazyData  true
RoxygenNote  7.1.0

LinkingTo  Rcpp, RcppArmadillo
Imports  Rcpp, stats, utils, methods
Suggests  testthat (>= 2.1.0), knitr, rmarkdown, microbenchmark

Collate  'RcppExports.R' 'baserate.R' 'baserateCT.R' 'baserateSet.R'
'calcKappa.R' 'calcRho.R' 'calculations.R' 'checkBRPKcombo.R'
'codeSet.R' 'contingencyTable.R' 'contingencyToSet.R'
'createRandomSet.R' 'createSimulatedCodeSet.R' 'genPKcombo.R'
'genPcombo.R' 'generateKPs.R' 'getBootPvalue.R' 'getHandSet.R'
'getHandSetIndices.R' 'getR.R' 'getTestSet.R' 'kappa.R'
'kappaCT.R' 'kappaSet.R' 'prset.R' 'rho.R' 'rho.file.R'
'rhoCT.R' 'rhoK.R' 'rhoMin.R' 'rhoR.R' 'rhoSet.R' 'utils.R'
'zzz.R'
NeedsCompilation: yes

Author: Brendan Eagan [aut],
       Brad Rogers [aut],
       Rebecca Pozen [aut],
       Cody L Marquart [cre, aut] (<https://orcid.org/0000-0002-3387-6792>),
       David Williamson Shaffer [aut]

Repository: CRAN

Date/Publication: 2020-09-13 14:20:03 UTC

R topics documented:

as.code.set .......................................................... 3
as.contingency.table ............................................. 3
baserate ............................................................. 4
baserateCT .......................................................... 5
baserateSet ......................................................... 5
codeSet ............................................................. 6
contingencyTable ................................................... 7
contingency_table .................................................. 7
createSimulatedCodeSet ........................................... 8
generateKPs_c ....................................................... 9
getBootPvalue_c .................................................... 10
getHandCT ........................................................... 10
getHandSet .......................................................... 11
getHandSetIndices .................................................. 11
getHand_kappa ....................................................... 12
getTestSet .......................................................... 12
kappa ................................................................. 13
kappaCT ............................................................. 14
kappaSet ............................................................. 14
kappa_ct ............................................................ 15
random_contingency_table ......................................... 15
recall ................................................................. 16
rho ................................................................. 16
rho.file ............................................................. 18
rhoCT ............................................................... 19
rhoK ................................................................. 21
rhoMin .............................................................. 22
rhoR ................................................................. 23
rhoSet .............................................................. 24
sample_contingency_table ......................................... 25
$.rating.set ........................................................ 25

Index: 26
as.code.set

Convert codeset to contingency table

Description

Convert codeset to contingency table

Usage

as.code.set(x)

Arguments

x matrix contingency table (2x2)

Value

2-column matrix representation of the contingency table

as.contingency.table

Convert a codeset to a contingency table

Description

Convert a codeset to a contingency table

Usage

as.contingency.table(x)

Arguments

x codeset

Value

contingency table as a 2x2 matrix
Calculate Baserate

Description

This function calculates the baserate of the first rater, second rater, and the average of both the raters.

Usage

baserate(data)

Arguments

data The testSet or contingencyTable for which the baserate is calculated.

Details

A baserate is the percentage, as a decimal, that a positive code appears in data (either a codeSet or contingencyTable) for a given rater. It is assumed that the first rater is more experienced and thus provides a better estimation of the actual baserate for a given code, so the first rater’s baserate is often used as if it is the actual baserate. If the raters are assumed to have the same experience level, the average baserate may give a better estimation. If the second rater is more experienced, the second rater’s baserate may give a better estimation. Functions assume that the first rater is the more experienced rater and thus uses the first rater’s baserate as the overall baserate estimation.

Value

A list of the format:

- firstBaserate The percentage of the data for which a positive code, or a 1, appears in the first rater
- secondBaserate The percentage of the data for which a positive code, or a 1, appears in the second rater
- averageBaserate The average of the firstBaserate and secondBaserate.

See Also

baserateSet and baserateCT

Examples

# Given a code set
baserate(data = codeSet)

# Given a contingency Table
baserate(data = contingencyTable)
baserateCT  

*Calculate Baserate (CT)*

**Description**

This function calculates the baserate of the first rater, second rater, and the average of both the raters. Called by `baserate`.

**Usage**

```
baserateCT(CT)
```

**Arguments**

- **CT**  
  The *contingencyTable* for which the baserate is calculated

**Value**

A list of the format:

- **firstBaserate**  
  The percentage of the data for which a positive code, or a 1, appears in the first rater
- **secondBaserate**  
  The percentage of the data for which a positive code, or a 1, appears in the second rater
- **averageBaserate**  
  The average of the firstBaserate and secondBaserate.

**See Also**

- `baserate` and `baserateSet`

baserateSet  

*Calculate Baserate (Set)*

**Description**

This function will calculate the baserate of the first rater, second rater, and the average of both the raters. Called by `baserate`.

**Usage**

```
baserateSet(set)
```

**Arguments**

- **set**  
  The *codeSet* for which the baserate is calculated
Value

A list of the format:

- **firstBaserate**: The percentage that a positive code, or a 1, appears in the first rater
- **secondBaserate**: The percentage that a positive code, or a 1, appears in the second rater
- **averageBaserate**: The average percentage that a positive code, or a 1, appears in either of the two raters

See Also

- `baserate` and `baserateCT`

---

### codeSet
codeSet

**Description**

A codeSet is a Nx2 binary matrix in which the first column corresponds to the first rater and the second column corresponds to the second rater.

**Usage**

codeSet

**Format**

The codeSet is an object of class matrix with n rows and two columns.

**Examples**

```r
#An example codeSet
firstRater = c(1, 1, 1, rep(0, 36))
secondRater = c(1, 1, 0, 1, 1, rep(0, 34))
exampleSet = cbind(firstRater, secondRater)

#This set is included in the package under the variable name "codeSet".
```
**Description**

A contingency Table is a 2x2 matrix that contains the counts of all combinations of positive and negative ratings made by two raters.

**Usage**

contingencyTable

**Format**

The contingency Table is an object of class matrix with two rows and two columns. The ordering of the combination vector input to the matrix is as follows: c(Rater1Positive & Rater2Positive, Rater1Negative & Rater2Positive, Rater1Positive & Rater2Negative, Rater1Negative & Rater2Negative).

**Examples**

```r
#An example contingencyTable
c = matrix(c(3,2,1,3), nrow = 2, ncol = 2)
#This contingencyTable is included in the package under the variable name "contingencyTable".
```

**Description**

Create a contingency table using the provided precision, recall, baserate, and length.

**Usage**

contingency_table(precision, rec, length, baserate)

**Arguments**

- precision: double
- rec: double
- length: int
- baserate: double
createSimulatedCodeSet

Create Simulated codeSet

Description

Creates a simulated codeSet with the given parameters

Usage

createSimulatedCodeSet(
  length, 
  baserate, 
  kappaMin, 
  kappaMax, 
  precisionMin, 
  precisionMax, 
  ...,
  tries = 50
)

Arguments

length the length of the simulated codeSet to be created
baserate the baserate of the simulated codeSet
kappaMin the minimum kappa of the simulated codeSet
kappaMax the maximum kappa of the simulated codeSet
precisionMin the minimum precision of the simulated codeSet
precisionMax the maximum precision of the simulated codeSet
... Parameters passed to createRandomSet (e.g. type = "set" or type = "ct")
tries the maximum number of tries to generate a valid set, smaller set lengths may require an increased number of tries

Details

codeSets are generated by first picking a random kappa within its range and a random precision within its range. If the random kappa, random precision, and baserate are not mathematically possible, then the precision is resampled from a range of mathematically possible values within its range. A unique simulated codeSet is then constructed given these parameters.

Value

A codeSet that fulfills the given parameters
Description

generate_kp_list

Usage

generate_kp_list(
    numNeeded,
    baserate,
    kappaMin,
    kappaMax,
    precisionMin,
    precisionMax,
    distributionType = 0L,
    distributionLength = 10000L
)

Arguments

- numNeeded: int
- baserate: double
- kappaMin: double
- kappaMax: double
- precisionMin: double
- precisionMax: double
- distributionType: int 0 - normal (default), 1 - bell
- distributionLength: long

Value

matrix of kappa and precision values (column 1 as precision)
**getBootPvalue_c**

**Description**

returns the percentage of the time that the distribution was greater or equal to the observed kappa if the result is less than the mean of the distribution, than the p value is 1 else return the number of times that the distribution is greater than the result as a percentage of the total number of items in the distribution

**Usage**

```r
getBootPvalue_c(distribution, result)
```

**Arguments**

- `distribution`: vector of calculated kappas
- `result`: double calculated kappa to compare against

**Value**

double calculated p-value

---

**getHandCT**

**Get Handset**

**Description**

This function is to get a handset of a set and calculate the kappa

**Usage**

```r
getHandCT(full.ct, handSetLength, handSetBaserate, as_kappa = TRUE)
```

**Arguments**

- `full.ct`: This is the set to take a handset of
- `handSetLength`: This is the length of the handset to take
- `handSetBaserate`: This is the minimum baserate to inflate the handset to
- `as_kappa`: If FALSE then return the handSet, if TRUE (default) return the kappa of the handSet

**Value**

The function returns the handSet if returnSet is TRUE or the kappa of the handSet if not
**getHandSet**

*Get Handset*

**Description**

This function is to get a handset of a set and calculate the kappa.

**Usage**

```r
getHandSet(set, handSetLength, handSetBaserate, returnSet = FALSE)
```

**Arguments**

- **set**: This is the set to take a handset of.
- **handSetLength**: This is the length of the handset to take.
- **handSetBaserate**: This is the minimum baserate to inflate the handset to.
- **returnSet**: If TRUE, then return the handSet if FALSE, return the kappa of the handSet.

**Value**

The function returns the handSet if returnSet is TRUE or the kappa of the handSet if not.

**getHandSetIndices**

*Generate a Handset*

**Description**

Generate a vector representing indices of set, using the handSetBaserate to determine the minimum number of indices that are positive.

**Usage**

```r
getHandSetIndices(set, handSetLength = 20, handSetBaserate = 0.2)
```

**Arguments**

- **set**: matrix of two columns.
- **handSetLength**: number of indices to find.
- **handSetBaserate**: number between 0 and 1 to use as a minimum number of positive indices.

**Value**

vector of indices from set.
**getHand_kappa**

**Description**
This function returns kappa calculated from a Handset taken from a larger Contingency Table.

**Usage**
```r
getchand_kappa(ct, handSetLength, handSetBaserate)
```

**Arguments**
- `ct`: KPs matrix of kappa (column 1) and precision (column 2) values.
- `handSetLength`: The length of the testSet (ignored unless data is an observed kappa value).
- `handSetBaserate`: Base rate to inflate the sampled contingency table to.

**Value**
- Kappa as double.

---

**getTestSet**

**Get Test Set**

**Description**
This function gets a testSet from a larger codeSet given certain sampling parameters.

**Usage**
```r
gittestset(set, testSetLength, testSetBaserateInflation = 0)
```

**Arguments**
- `set`: The codeSet from which the testSet is taken.
- `testSetLength`: The length of the testSet to be taken.
- `testSetBaserateInflation`: The minimum guaranteed baserate of the testSet. Default to 0.
**kappa**

**Details**

A testSet is a codeSet that is a subset of a larger codeSet with a given set of properties. A testSet is constructed by sampling (without replacement) P rows from rows in the larger codeSet where the first rater’s code was 1, and then appending an additional sample (without replacement) of R rows taken at random from the larger codeSet excluding rows included in the first P rows sampled. P is computed as the minbaserate * length of the testset. R is computed as testSetLength - P. The result of this sampling procedure is to create a sample with a minimum baserate regardless of the baserate of the larger codeSet. If testSetBaserateInflation is set to zero, the function selects rows at random.

**Value**

A codeSet with the properties specified

---

**kappa** *Calculate kappa*

**Description**

This function calculates Cohen’s kappa on a contingencyTable or a codeSet

**Usage**

kappa(data)

**Arguments**

data A contingencyTable or a codeSet

**Value**

The kappa of the contingencyTable or codeSet

**See Also**

kappaSet and kappaCT

**Examples**

#Given a code set
kappa(data = codeSet)

#Given a contingency Table
kappa(data = contingencyTable)
### kappaCT

*Calculate kappa (contingency Table)*

**Description**

This function calculates Cohen’s kappa on a *contingencyTable*. Called by *kappa*.

**Usage**

`kappaCT(ct)`

**Arguments**

- `ct`: A *contingencyTable*

**Value**

The kappa of the *contingencyTable*

**See Also**

- *kappa* and *kappaSet*

### kappaSet

*Calculate kappa (Set)*

**Description**

This function calculates Cohen’s kappa for a given *codeSet*. Called by *kappa*.

**Usage**

`kappaSet(set)`

**Arguments**

- `set`: A *codeSet*

**Value**

The kappa of the *codeSet*

**See Also**

- *kappa* and *kappaCT*
kappa_ct

---

### Description

Calculate kappa from a contingency table.

### Usage

```r
kappa_ct(ct)
```

### Arguments

- `ct` [TBD]

---

### Description

random_contingency_table

### Usage

```r
random_contingency_table(
    setLength,
    baserate,
    kappaMin,
    kappaMax,
    minPrecision = 0,
    maxPrecision = 1
)
```

### Arguments

- `setLength` [TBD]
- `baserate` [TBD]
- `kappaMin` [TBD]
- `kappaMax` [TBD]
- `minPrecision` [TBD]
- `maxPrecision` [TBD]
**recall**

**Description**
recall

**Usage**
recall(kappa, BR, P)

**Arguments**
kappa        double
BR          double
P           double

**Value**
Recall calculated from provided kappa, BR, and P

---

**rho**

**Description**
This function calculates rho for a testSet, contingencyTable, or an observed kappa value with associated set parameters (testSetLength and OcSBaserate).

**Usage**
rho(
x,
  OcSBaserate = NULL,
  testSetLength = NULL,
  testSetBaserateInflation = 0,
  OcSLength = 10000,
  replicates = 800,
  ScSKappaThreshold = 0.9,
  ScSKappaMin = 0.4,
  ScSPrecisionMin = 0.6,
  ScSPrecisionMax = 1
)
Arguments

\( x \)  
The observed kappa value, \texttt{testSet} or \texttt{contingencyTable} that will be tested with \texttt{rho}.

\( \text{OcSBaserate} \)  
The \texttt{baserate} of the observed \texttt{codeSet} (defaults to \texttt{baserate} of \texttt{testSet} or \texttt{contingencyTable}).

\( \text{testSetLength} \)  
The length of the \texttt{testSet} (ignored unless \texttt{data} is an observed kappa value).

\( \text{testSetBaserateInflation} \)  
The minimum \texttt{baserate} from the sampling procedure.

\( \text{OcSLength} \)  
The length of the observed \texttt{codeSet}.

\( \text{replicates} \)  
The number of simulated \texttt{codeSets} to use in the null hypothesis distribution for \texttt{rho}; similar to replicates in a Monte Carlo study.

\( \text{ScSKappaThreshold} \)  
The maximum kappa value used to generate simulated \texttt{codeSets} in the null hypothesis distribution for \texttt{rho}.

\( \text{ScSKappaMin} \)  
The minimum kappa value used to generate simulated \texttt{codeSets} in the null hypothesis distribution for \texttt{rho}.

\( \text{ScSPrecisionMin} \)  
The minimum precision to be used for generation of simulated \texttt{codeSets} in the null hypothesis distribution for \texttt{rho}.

\( \text{ScSPrecisionMax} \)  
The maximum precision to be used for generation of simulated \texttt{codeSets} in the null hypothesis distribution for \texttt{rho}.

Details

\texttt{rho} is a Monte Carlo rejective method of interrater reliability statistics, implemented here for Cohen’s Kappa. \texttt{rho} constructs a collection of data sets in which kappa is below a specified threshold, and computes the empirical distribution on kappa based on the specified sampling procedure. \texttt{rho} returns the percent of the empirical distribution greater than or equal to an observed kappa. As a result, \texttt{rho} quantifies the type 1 error in generalizing from an observed test set to a true value of agreement between two raters.

\texttt{rho} starts with an observed kappa value, calculated on a subset of a \texttt{codeSet}, known as an observed \texttt{testSet}, and a \texttt{kappa threshold} which indicates what is considered significant agreement between raters.

It then generates a collection of fully-coded, simulated \texttt{codeSets} (ScS), further described in \texttt{createSimulatedCodeSet}, all of which have a kappa value below the kappa threshold and similar properties as the original \texttt{codeSet}.

Then, kappa is calculated on a \texttt{testSet} sampled from each of the ScSs in the collection to create a null hypothesis distribution. These \texttt{testSets} mirror the observed \texttt{testSets} in their size and sampling method. How these \texttt{testSets} are sampled is further described in \texttt{getTestSet}.

The null hypothesis is that the observed \texttt{testSet}, was sampled from a data set, which, if both raters were to code in its entirety, would result in a level of agreement below the kappa threshold.

For example, using an alpha level of 0.05, if the observed kappa is greater than 95 percent of the kappas in the null hypothesis distribution, the null hypothesis is rejected. Then one can conclude that the two raters would have acceptable agreement had they coded the entire data set.
rho.file

Value

rho for the given parameters
rho and kappa for the given data and parameters (unless kappa is given)

See Also

rho

Examples

# Given an observed kappa value
rho(x = 0.88, OcSBaserate = 0.2, testSetLength = 80)

# Given a test Set
rho(x = codeSet)

# Given a contingency Table
rho(x = contingencyTable)

Description

This function calculates rho and kappa for a given testSet as defined by the file and columns (col1, col2), and returns a list containing both values. Called by rho.

Usage

rho.file(
  x,
  col1,
  col2,
  OcSBaserate = NULL,
  testSetBaserateInflation = 0,
  OcSLength = 10000,
  replicates = 800,
  ScSKappaThreshold = 0.9,
  ScSKappaMin = 0.4,
  ScSPrecisionMin = 0.6,
  ScSPrecisionMax = 1
)

Rho using a file
Arguments

- **x**: The observed kappa value, `testSet` or `contingencyTable` that will be tested with `rho`.
- **col1**: The first column from file.
- **col2**: The second column from file.
- **OcSBaserate**: The `baserate` of the observed `codeSet` (defaults to `baserate` of `testSet` or `contingencyTable`).
- **testSetBaserateInflation**: The minimum `baserate` from the sampling procedure.
- **OcSLength**: The length of the observed `codeSet`.
- **replicates**: The number of simulated `codeSets` to use in the null hypothesis distribution for `rho`; similar to replicates in a Monte Carlo study.
- **ScSKappaThreshold**: The maximum kappa value used to generate simulated `codeSets` in the null hypothesis distribution for `rho`.
- **ScSKappaMin**: The minimum kappa value used to generate simulated `codeSets` in the null hypothesis distribution for `rho`.
- **ScSPrecisionMin**: The minimum precision to be used for generation of simulated `codeSets` in the null hypothesis distribution for `rho`.
- **ScSPrecisionMax**: The maximum precision to be used for generation of simulated `codeSets` in the null hypothesis distribution for `rho`.

Value

- **rho** for the given parameters

A list of the format:

- **rho**: The rho of the `codeSet`
- **kappa**: The Cohen’s Kappa of the `codeSet`

See Also

- **rho**

---

**rhoCT**

*Rho (contingency Table)*

Description

This function calculates rho and kappa for a given `contingencyTable`, and returns a list containing both values. Called by **rho**.
Usage

rhoCT(
  x,
  OcSBaserate = NULL,
  testSetBaserateInflation = 0,
  OcSLength = 10000,
  replicates = 800,
  ScSKappaThreshold = 0.9,
  ScSKappaMin = 0.4,
  ScSPrecisionMin = 0.6,
  ScSPrecisionMax = 1
)

Arguments

x                   The observed kappa value, testSet or contingencyTable that will be tested with rho

OcSBaserate        The baserate of the observed codeSet (defaults to baserate of testSet or contingencyTable)

testSetBaserateInflation
                   The minimum baserate from the sampling procedure

OcSLength           The length of the observed codeSet

replicates          The number of simulated codeSets to use in the null hypothesis distribution for rho; similar to replicates in a Monte Carlo study

ScSKappaThreshold   The maximum kappa value used to generate simulated codeSets in the null hypothesis distribution for rho

ScSKappaMin         The minimum kappa value used to generate simulated codeSets in the null hypothesis distribution for rho

ScSPrecisionMin     The minimum precision to be used for generation of simulated codeSets in the null hypothesis distribution for rho

ScSPrecisionMax     The maximum precision to be used for generation of simulated codeSets in the null hypothesis distribution for rho

Value

rho for the given parameters

A list of the format:

   rho  The rho of the contingencyTable
   kappa The Cohen’s Kappa of the contingencyTable

See Also

rho
Description
This function calculates rho for an observed kappa value with associated set parameters (testSetLength and OcSBaserate). Called by rho. A p-value is returned and if this value is less than 0.05, it is said that the handset does generalize to the entire set.

Usage
rhoK(x, OcSBaserate, testSetLength, testSetBaserateInflation = 0, OcSLength = 10000, replicates = 800, ScSKappaThreshold = 0.9, ScSKappaMin = 0.4, ScSPrecisionMin = 0.6, ScSPrecisionMax = 1, method = "standard")

Arguments
x The observed kappa value, testSet or contingencyTable that will be tested with rho
OcSBaserate The baserate of the observed codeSet (defaults to baserate of testSet or contingencyTable)
testSetLength The length of the testSet (ignored unless data is an observed kappa value)
testSetBaserateInflation The minimum baserate from the sampling procedure
OcSLength The length of the observed codeSet
replicates The number of simulated codeSets to use in the null hypothesis distribution for rho; similar to replicates in a Monte Carlo study
ScSKappaThreshold The maximum kappa value used to generate simulated codeSets in the null hypothesis distribution for rho
ScSKappaMin The minimum kappa value used to generate simulated codeSets in the null hypothesis distribution for rho
ScSPrecisionMin The minimum precision to be used for generation of simulated codeSets in the null hypothesis distribution for rho
ScSPrecisionMax
The maximum precision to be used for generation of simulated codeSets in the null hypothesis distribution for rho

method
set to "c" to calculate using the C++ implmentation. Defaults to "standard"

Value
rho for the given parameters
rho for the given parameters

See Also
rho

Usage
rhoMin(baserate, alpha = 0.05, inc = 10, printInc = FALSE, ...)

Arguments
baserate  A baserate
alpha     The threshold of significance for rho (similar to an alpha level for a p value), defaulted to 0.05
inc       An integer indicating by how much the testSetLength should increase each iteration
printInc  A boolean indicating whether to print out each increment value with it's corresponding significance for rho
...
Any additional parameters passed into rho

Value
The minimum length of testSet, to the nearest multiple of inc, greater than the minimum length, that would give a value where rho less than alpha becomes mathematically possible.
Examples

```r
# Add testSetBaserateInflation as an additional parameter
rhoMin(0.2, testSetBaserateInflation = 0.33)
```

```r
# Add testSetBaserateInflation as well as changing inc and selecting printInc
rhoMin(0.2, inc = 5, printInc = TRUE, testSetBaserateInflation = 0.33)
```

rhoR: A package for computing rho

Description

Rho is used to test the generalization of inter rater reliability (IRR) statistics, in this case Cohen’s Kappa.

Rho is a Monte Carlo rejective method of interrater reliability statistics, implemented here for Cohen’s Kappa. Rho constructs a collection of data sets in which kappa is below a specified threshold, and computes the empirical distribution on kappa based on the specified sampling procedure. Rho returns the percent of the empirical distribution greater than or equal to an observed kappa. As a result, Rho quantifies the type I error in generalizing from an observed test set to a true value of agreement between two raters.

Rho starts with an observed kappa value, calculated on a subset of a `codeSet`, known as an observed `testSet`, and a `kappa threshold` which indicates what is considered significant agreement between raters.

It then generates a collection of fully-coded, simulated `codeSets` (ScS), further described in `createSimulatedCodeSet`, all of which have a kappa value below the kappa threshold and similar properties as the original `codeSet`.

Then, kappa is calculated on a `testSet` sampled from each of the ScSs in the collection to create a null hypothesis distribution. These `testSets` mirror the observed `testSet` in their size and sampling method. How these `testSets` are sampled is further described in `testSet`.

The null hypothesis is that the observed `testSet` was sampled from a data set, which, if both raters were to code in its entirety, would result in a level of agreement below the kappa threshold.

For example, using an alpha level of 0.05, if the observed kappa is greater than 95 percent of the kappas in the null hypothesis distribution, the null hypothesis is rejected. Then one can conclude that the two raters would have acceptable agreement had they coded the entire data set.

rho

Use `rho` `rhoK` `rhoSet` `rhoCT`

kappa

Use `kappa` `kappaSet` `kappaCT`
rhoSet

**Use** rhoMin

### Description

This function calculates rho and kappa for a given `testSet`, and returns a list containing both values. Called by `rho`.

### Usage

```r
rhoSet(
  x,
  OcSBaserate = NULL,
  testSetBaserateInflation = 0,
  OcSLength = 10000,
  replicates = 800,
  ScSKappaThreshold = 0.9,
  ScSKappaMin = 0.4,
  ScSPrecisionMin = 0.6,
  ScSPrecisionMax = 1
)
```

### Arguments

- **x**: The observed kappa value, `testSet` or `contingencyTable` that will be tested with rho
- **OcSBaserate**: The baserate of the observed codeSet (defaults to baserate of `testSet` or `contingencyTable`)
- **testSetBaserateInflation**: The minimum baserate from the sampling procedure
- **OcSLength**: The length of the observed codeSet
- **replicates**: The number of simulated `codeSets` to use in the null hypothesis distribution for rho; similar to replicates in a Monte Carlo study
- **ScSKappaThreshold**: The maximum kappa value used to generate simulated `codeSets` in the null hypothesis distribution for rho
- **ScSKappaMin**: The minimum kappa value used to generate simulated `codeSets` in the null hypothesis distribution for rho
- **ScSPrecisionMin**: The minimum precision to be used for generation of simulated `codeSets` in the null hypothesis distribution for rho
- **ScSPrecisionMax**: The maximum precision to be used for generation of simulated `codeSets` in the null hypothesis distribution for rho
Value

- rho for the given parameters
- A list of the format:
  - rho  The rho of the codeSet
  - kappa  The Cohen's Kappa of the codeSet

See Also

- rho

Description

ds sample_contingency_table

Usage

```r
sample_contingency_table(xx, n, forR = TRUE)
```

Arguments

- `xx`  contingency table matrix
- `n`  int size of the contingency table
- `forR`  bool if true, add 1 to the results accounting for R indices starting at 1

$.rating.set

Helper function to return special values on a rating set

Description

- Helper function to return special values on a rating set

Usage

```r
## S3 method for class 'rating.set'
x$i
```

Arguments

- `x`  Set or Contingency.Table
- `i`  Value to search for
Index

* datasets
  codeSet, 6
  contingencyTable, 7
  $.rating.set, 25
  as.code.set, 3
  as.contingency.table, 3
  baserate, 4, 5, 6, 8, 12, 17, 19–22, 24
  baserateCT, 4, 5, 6
  baserateSet, 4, 5, 5
  codeSet, 4, 5, 6, 8, 12–14, 17, 19–21, 23–25
  codeSets, 17, 19–24
  contingency_table, 7
  contingencyTable, 4, 5, 7, 13, 14, 16, 17, 19–21, 24
  createSimulatedCodeSet, 8, 17, 23
  generate_kp_list (generateKPs_c), 9
  generateKPs_c, 9
  getBootPvalue_c, 10
  getHand_kappa, 12
  getHandCT, 10
  getHandSet, 11
  getHandSetIndices, 11
  getTestSet, 12, 17
  kappa, 13, 14, 23
  kappa_ct, 15
  kappaCT, 13, 14, 14, 23
  kappaSet, 13, 14, 14, 23
  random_contingency_table, 15
  recall, 16
  rho, 16, 18–25
  rho.file, 18
  rhoCT, 19, 23
  rhoK, 21, 23
  rhoMin, 22, 24
  rhoR, 23
  rhoSet, 23, 24
  sample_contingency_table, 25
  testSet, 4, 12, 16–21, 23, 24
  testSets, 17, 23