Package: heuristica (via r-universe)

September 17, 2024

Title Heuristics Including Take the Best and Unit-Weight Linear

Version 1.0.3.9000

Description Implements various heuristics like Take The Best and unit-weight linear, which do two-alternative choice: which of two objects will have a higher criterion? Also offers functions to assess performance, e.g. percent correct across all row pairs in a data set and finding row pairs where models disagree. New models can be added by implementing a fit and predict function-- see vignette. Take The Best was first described in: Gigerenzer, G. & Goldstein, D. G. (1996) <doi:10.1037/0033-295X.103.4.650>. All of these heuristics were run on many data sets and analyzed in: Gigerenzer, G., Todd, P. M., & the ABC Group (1999). <ISBN:978-0195143812>.

Depends R (>= 3.1.0)

License MIT + file LICENSE

LazyData true

Imports Hmisc

URL https://github.com/jeanimal/heuristica

BugReports https://github.com/jeanimal/heuristica/issues

Suggests devtools, ggplot2, glmnet, knitr, plyr, reshape, reshape2, rmarkdown, testthat

VignetteBuilder knitr

RoxygenNote 7.1.1

Repository https://jeanimal.r-universe.dev

RemoteUrl https://github.com/jeanimal/heuristica

RemoteRef HEAD

RemoteSha 441d5b9e4e479832eb22163907f7087542bd3cc8

Contents

accuracyFromConfusionMatrix3x3 *Accuracy based on a predictPair confusion matrix.*

Description

Given a confusion matrix from pair predict (the output of confusionMatrixFor_Neg1_0_1), calculate an accuracy. By default assumes zeroes are guesses and that half of them are correct. This guessing assumptions helps measures of accuracy converge faster for small samples, but it will artificially reduce the variance of an algorithm's predictions, if that is what you are trying to measure.

Usage

```
accuracyFromConfusionMatrix3x3(confusion_matrix, zero_as_guess = TRUE)
```

Arguments

confusion_matrix	A 3x3 matrix where rows are correct outcomes (-1, 0, 1) and columns are predicted outcomes (-1, 0, 1).
zero_as_guess	Optional parameter which by default treats the 2nd zero column as guesses and assigns half of them to be correct.

Value

A value from 0 to 1 for the proportion correct.

References

Wikipedia's entry on https://en.wikipedia.org/wiki/Confusion_matrix.

See Also

confusionMatrixFor_Neg1_0_1 for generating the confusion matrix.

Examples

```
# Below accuracy is 1 (100% correct) because 4 -1's were correctly predicted,
# and 2 1's were correctly predicted.  (On-diagonal elements are correct
# predictions.)
accuracyFromConfusionMatrix3x3(cbind(c(4,0,0), c(0,0,0), c(0,0,2)))

# 3 wrong and 3 more wrong for 0 accuracy.
accuracyFromConfusionMatrix3x3(cbind(c(0,0,3), c(0,0,0), c(3,0,0)))

# Below is 4 + 5 correct, 1 incorrect, for 9/10 = 0.9 accuracy.
accuracyFromConfusionMatrix3x3(cbind(c(4,0,1), c(0,0,0), c(0,0,5)))
```

```
# Below has 3+1=4 guesses, and 0.5 are assigned correct.
accuracyFromConfusionMatrix3x3(cbind(c(0,0,0), c(3,0,1), c(0,0,0)))
```

`city_population` *Population size of the 83 largest German cities.*

Description

Population size of the 83 German cities that had more than 100,000 inhabitants when this data was collected in 1993 plus cues indicating whether a city has a soccer team, intercity trainline, University, etc. All cues are binary.

Usage

```
city_population
```

Format

A data frame.

Name Name of city

Running_Number Running Number

Population Population size

Soccer_Team 1 indicates that the city has a soccer team, 0 indicates that it does not.

State_Capital 1 indicates that the city is a state capital, 0 indicates that it is not.

Former_East_Germany 1 indicates that the city belongs to former East Germany, 0 that is does not.

Industrial_Belt 1 indicates that the city is an industrial belt, 0 that it is not.

Licence_Plate 1 indicates that the city has a licence plate, 0 that it does not.

Intercity_Trainline 1 indicates that an intercity trainline crosses the city, 0 that it does not.

Exposition_Site 1 indicates that the city is an exposition size, 0 that it is not.

National_Capital 1 indicates that the city is the national capital, 0 that it is not.

University 1 indicates that the city has a University, 0 that it does not.

Details

The data is based on:

Fischer Welt Almanach [Fischer World Almanac]. (1993). Frankfurt, Germany: Fischer.

This is the data set used in simulations by the ABC (Adaptive Behavior and Cognition) research group.

city_population_original

Original, uncorrected Population size of the 83 largest German cities.

Description

In contrast to city_population, this has some transcription errors from the almanac, but it was used in published research, so it is provided for reproducibility.

Usage

```
city_population_original
```

Format

A data frame.

Name Name of city

Running_Number Running Number

Population Population size

Soccer_Team 1 indicates that the city has a soccer team, 0 indicates that it does not.

State_Capital 1 indicates that the city is a state capital, 0 indicates that it is not.

Former_East_Germany 1 indicates that the city belongs to former East Germany, 0 that is does not.

Industrial_Belt 1 indicates that the city is an industrial belt, 0 that it is not.

Licence_Plate 1 indicates that the city has a licence plate, 0 that it does not.

Intercity_Trainline 1 indicates that an intercity trainline crosses the city, 0 that it does not.

Exposition_Site 1 indicates that the city is an exposition size, 0 that it is not.

National_Capital 1 indicates that the city is the national capital, 0 that it is not.

University 1 indicates that the city has a University, 0 that it does not.

collapseConfusionMatrix3x3To2x2

Collapses a 3x3 confusion matrix to a 2x2 confusion matrix.

Description

A 3x3 confusion matrix results from predictPair.

Usage

```
collapseConfusionMatrix3x3To2x2(
  confusion_matrix_3x3,
  guess_handling_fn = distributeGuessAsExpectedValue,
  tie_handling_fn = distributeTies
)
```

Arguments

`confusion_matrix_3x3`
: A 3x3 confusion matrix.

`guess_handling_fn`
: A function to call on the 3x3 confusion matrix to assign a model's guesses– 0 predictions tracked in the 2nd column– to -1 or 1 counts.

`tie_handling_fn`
: A function to call on the 3x3 confusion matrix to distribute ties– 0 correct answers tracked in the 2nd row– to -1 or 1 counts.

Details

The middle column repressents guesses. The middle row represents ties. T

Value

A 2x2 confusion matrix.

`conditionalCueValidityComplete`

Calculate conditional cue validity, which includes reversing and ranks.

Description

Conditional cue validity is the validity of a cue taking into account decisions already made by higher-ranked cues. For a single cue, it is the same as cue validity. For two cues, the higher validity cue will have conditional cue validity = cue validity. However, the remaining cue will have its validity re-calculated on just those pairs of object where cue validity did not discriminate. In the case of binary data, there will be many pairs where the first cue did not discriminate. With real-valued data, there may be no such cases.

Usage

```
conditionalCueValidityComplete(data, criterion_col, cols_to_fit)
```

Arguments

`data`	The matrix or data.frame whose columns are treated as cues.
`criterion_col`	The index of the column used as criterion.
`cols_to_fit`	A vector of indexes of the columns to calculate cue validity for.

Value

A list of vectors with values for each column in cols_to_fit: $cue_validities: The validities based on reversed value, numbers ranging from 0 to 1. It will include NA if the validity cannot be calculated (e.g. higher-validity cues made decisions for all cases in the data set). $cue_ranks: Rank order from 1 to # of cues in cols_to_fit. Will be NA if validity was NA. $cue_directions: 1 if cue is in same direction as criterion, -1 if reversed. Will be NA if validity was NA.

References

Martignon, L., & Hoffrage, U. (2002). Fast, frugal, and fit: Simple heuristics for paired comparisons. Theory and Decision, 52: 29-71.

See Also

`cueValidity` and `cueValidityComplete` for the unconditional version.

Examples

```
# The data below differentiates between cue validity and conditional cue
# validity.  Cue validity of x1 is 1.0.  Cue validity of x2 is 0.5.
# But after you've chosen x1 as the highest-validity cue, only row2
# vs. row3 is undecided  x2 predictions correctly on those, so its
# conditional cue validity is 1.0 (conditional on x1 being first).
data <- cbind(y=c(5,4,3), x1=c(1,0,0), x2=c(0,1,0))
out <- conditionalCueValidityComplete(data, 1, c(2:3))
# This tells you both cues had validity 1-- it returns 1, 1.
out$cue_validities
# This tells you to choose x1 first-- it returns 1, 0.
out$cue_ranks
# This tells you they already point in the correct direction.
out$cue_directions
# For a case with a negative cue direction, try this data:
data2 <- cbind(y=c(5,4,3), x1=c(1,0,0), x2=c(1,0,1))
conditionalCueValidityComplete(data2, 1, c(2:3))
```

confusionMatrixFor_Neg1_0_1 *Confusion matrix for categories -1, 0, 1 (the output of predictPair).*

Description

Measuring accuracy of predicting categories, where in the predictPair paradigm the categories are the relative ranks of a pair of rows. The categories are: -1 means Row1 < Row2 0 means the rows are equal or guess 1 means Row1 > Row2

Usage

```
confusionMatrixFor_Neg1_0_1(ref_data, predicted_data)
```

Arguments

ref_data	A vector with outcome categories from a reference source to be predicted (e.g. the output of correctGreater.)
predicted_data	A vector with outcome categories from a prediction source that is trying to match ref_data (e.g. ttbModel predictions).

Value

A 3x3 matrix of counts. Rows are outcomes of the reference data. Columns are outcomes of predicted data.

References

Wikipedia's entry on https://en.wikipedia.org/wiki/Confusion_matrix.

Examples

```
# Example 1
# Below, the correct outcome is always 1, so only the last row of the
# confusion matrix has non-zero counts.  But the predictor makes a few
# mistakes, so some non-zero counts are off the diagonal.
confusionMatrixFor_Neg1_0_1(c(1,1,1), c(1,-1,-1))
# outputs:
#    -1 0 1
# -1  0 0 0
# 0   0 0 0
# 1   2 0 1
#
# Example 2
# The prediction always matches the reference outcome, so all non-zero
# counts are on the diagonal.
confusionMatrixFor_Neg1_0_1(c(1,1,0,0,-1,-1), c(1,1,0,0,-1,-1))
# outputs:
```

```
#    -1 0 1
# -1  2 0 0
# 0   0 2 0
# 1   0 0 2
#
```

`correctGreater` *Creates function indicating whether row1[col] > row2[col].*

Description

Using rowPairApply, this can generate a column indicating the the correct direction of the criterion in comparing row 1 vs. row2 for all row pairs in test_data. 1 indicates row 1's criterion > row 2's criterion 0 indicates they are equal -1 indicates row 2's criterion is greater By default, the output column is called "CorrectGreater," but you can override the name with output_column_name.

Usage

```
correctGreater(criterion_col, output_column_name = "CorrectGreater")
```

Arguments

`criterion_col`	The integer index of the criterion in test_data.
`output_column_name`	An optional string

Details

This is meant to be used to measure the performance of heuristics wrapped with `heuristics`.

Value

An object that implements createFunction. Users will generally not use this directly– rowPairApply will.

See Also

`heuristics` is the wrapper to get the predicted greater row in the row pair for each heuristic passed in to it.

`rowPairApply` which has an example of using this.

cueAccuracy *Calculate the accuracy of using a cue to predict a criterion.*

Description

cueValidity counts only correct and incorrect inferences, ignoring cases where a cue does not discriminate. Cue accuracy gives those cases a weight of 0.5, the expected accuracy of guessing. It is calculated as (correct + 0.5 * guesses) / (correct + incorrect + guesses).

Usage

```
cueAccuracy(criterion, cue, replaceNanWith = 0.5)
```

Arguments

criterion	A vector of values to be predicted.
cue	A vector of values to predict with. Should have the same length as the criterion.
replaceNanWith	The value to return as cue validity in case it cannot be calculated, e.g. no variance in the values.

Value

The cue accuracy, a value in the range [0,1].

See Also

cueValidity for an alternate measure used in Take The Best.

Examples

```
cueValidity(c(5,1), c(1,0))
cueAccuracy(c(5,1), c(1,0))
# Both return 1.
cueValidity(c(5,2,1), c(1,0,0))
cueAccuracy(c(5,2,1), c(1,0,0))
# Cue validity still returns 1 but cue accuracy returns (2+0.5)/3 = 0.833.
```

`cueValidity` *Calculate the cue validity.*

Description

Calculate the cue validity for a pair of vectors. It is calculated as correct / (correct + incorrect).

Usage

```
cueValidity(criterion, cue, replaceNanWith = 0.5)
```

Arguments

`criterion`	A vector of values to be predicted.
`cue`	A vector of values to predict with. Should have the same length as the criterion.
`replaceNanWith`	The value to return as cue validity in case it cannot be calculated, e.g. no variance in the values.

Value

The cue validity, a value in the range [0,1].

References

Wikipedia's entry on https://en.wikipedia.org/wiki/Cue_validity

See Also

`cueValidityComplete` for more complete output.

`conditionalCueValidityComplete` for a version where validity is conditional on cues already used to make decisions.

`cueAccuracy` for a measure that takes guesses into account.

Examples

```
cueValidity(c(5,1), c(1,0))
# Returns 1.
cueValidity(c(5,2,1), c(1,0,0))
# Also returns 1
cueValidity(c(5,2,1), c(0,0,1))
# Returns 0.
cueValidity(c(5,2,1), c(1,0,1))
# Returns 0.5.
```

`cueValidityAppliedToColumns` *Calculate the cue validity for the cols_to_fit columns.*

Description

This returns only the cue validities, without reversing when a cue points in the wrong direction– e.g. education is negatively associated with number of felonies, so we should use LESS education as a predictor. Use cueValidityComplete for help with that.

Usage

```
cueValidityAppliedToColumns(
  data,
  criterion_col,
  cols_to_fit,
  replaceNanWith = 0.5
)
```

Arguments

`data`	The matrix or data.frame whose columns are treated as cues.
`criterion_col`	The index of the column used as criterion.
`cols_to_fit`	A vector of indexes of the columns to calculate cue validity for.
`replaceNanWith`	The value to return as cue validity in case it cannot be calculated.

Value

A list where $cue_validities has a vector of validities for each of the columns in cols_to_fit.

References

Wikipedia's entry on https://en.wikipedia.org/wiki/Cue_validity

See Also

`cueValidityComplete` for more complete output.

`cueValidityComplete` *Calculate cue validity with reverse, cue directions, and cue ranks.*

Description

This provides a vector of cue_validities and potentially other useful information, particularly if reverse_cues=TRUE. For example, education is negatively associated with number of felonies. If reverse_cues=FALSE, education will get validity < 0.5. If reverse_cues=TRUE, then LESS education will be used as a predictor, resulting in: 1) cue_validity > 0.5 2) cue_direction == -1 To use the cue for prediction, be sure to multiply it by the cue_direction. For ranking, based heuristics, cue_ranks gives the rank order of cues where highest validity = rank 1 (after reversing, if any).

Usage

```
cueValidityComplete(
  data,
  criterion_col,
  cols_to_fit,
  replaceNanWith = 0.5,
  reverse_cues = FALSE,
  ties.method = "random"
)
```

Arguments

`data`	The matrix or data.frame whose columns are treated as cues.
`criterion_col`	The index of the column used as criterion.
`cols_to_fit`	A vector of indexes of the columns to calculate cue validity for.
`replaceNanWith`	The value to return as cue validity in case it cannot be calculated.
`reverse_cues`	Optional parameter to reverse cues as needed. By default, the model will reverse the cue values for cues with cue validity < 0.5, so a cue with validity 0 becomes a cue with validity 1. Set this to FALSE if you do not want that, i.e. the cue stays validity 0.
`ties.method`	An optional parameter passed to rank: A character string sepcifying how ties (in cue validity) are treated.

Value

A list where $cue_validities has a vector of validities for each of the columns in cols_to_fit.

References

Wikipedia's entry on https://en.wikipedia.org/wiki/Cue_validity

`distributeGuessAsExpectedValue` *Distributes guesses of 3x3 confusion matrix to expected value of 1 and -1.*

Description

Given a 3x3 confusion matrix, distributes guesses in column 2 using the expected value. That is, moves half of guess counts (in column 2) to -1 (column 1) and the other half to 1 (column 3).

Usage

```
distributeGuessAsExpectedValue(confusion_matrix_3x3)
```

Arguments

`confusion_matrix_3x3` A 3x3 matrix where the middle column is counts of guesses.

Details

-1 0 1 -1 2 2 2 0 4 4 4 1 6 6 6 becomes -1 0 1 -1 3 0 3 0 6 0 6 1 9 0 9

Value

A 3x3 confusion matrix with 0's in the middle column.

`heuristics` *Wrap fitted heuristics to pass to rowPairApply to call predictPair.*

Description

One or more fitted heuristics can be passed in. They must all have the same cols_to_fit. If they differ on cols_to_fit, then group them in separate heuristics functions.

Usage

```
heuristics(...)
```

Arguments

`...` A list of predictPairInternal implementers, e.g. a fitted ttb model.

Details

Users will generally not use the output directly but instead pass this to rowPairApply.

Value

An object of class heuristics, which implements createFunction. Users will generally not use this directly– rowPairApply will.

See Also

rowPairApply which is what the output of heuristics is normally passed in to.

heuristicsList for a version of this function where you can control the function called (not necessarily predictPairInternal).

predictPairInternal which must be implemented by heuristics in order to use them with the heuristics() wrapper function. This only matters for people implementing their own heuristics.

Examples

```
# Use one fitted ttbModel with column 1 as criterion and columns 2,3 as
# cues.
data <- cbind(y=c(30,20,10,5), x1=c(1,1,0,0), x2=c(1,1,0,1))
ttb <- ttbModel(data, 1, c(2:3))
rowPairApply(data, heuristics(ttb))
# This outputs ttb's predictions for all 6 row pairs of data.
# (It has 6 row pairs because 4*2/2 = 6.)  It gets the predictions
# by calling ttb's predictPairInternal.

# Use the same fitted ttbModel plus a unit weight model with the same
# criterion and cues.
unit <- unitWeightModel(data, 1, c(2,3))
rowPairApply(data, heuristics(ttb, unit))
# This outputs predictions with column names 'ttbModel' and
# 'unitWeightLinearModel'.

# Use the same fitted ttbModel plus another ttbModel that has different
# cols_to_fit.  This has to be put in a separate heuristicsList function.
ttb_just_col_3 <- ttbModel(data, 1, c(3), fit_name="ttb3")
rowPairApply(data, heuristics(ttb), heuristics(unit))
# This outputs predictions with column names 'ttbModel' and
# 'ttb3'.
```

---

heuristicsList *Wrapper for fitted heuristics to generate predictions with rowPairApply.*

Description

A list of fitted heuristics are passed in. They must all implement the fn function passed in, and they must all have the same cols_to_fit. If they differ on these, then group them in separate heuristicsLists.

Usage

```
heuristicsList(list_of_fitted_heuristics, fn)
```

Arguments

list_of_fitted_heuristics	Normally a list of predictProbInternal implementers, e.g. a fitted ttb model.
fn	The function to be called on the heuristics, which is typically predictPairInternal (or the experimental function predictProbInternal) but can be any function with the signature function(object, row1, row2) that is implemented by the heuristics in list_of_fitted_heuristics.

Details

Users will generally not use the output directly– instead just pass this into one of the rowPairApply functions.

Value

An object of class heuristics, which implements createFunction. Users will generally not use this directly– rowPairApply will.

See Also

rowPairApply which is what the output of heuristicsList is normally passed in to.

heuristics for a simpler version of this function with more examples. It is recommended for most uses. (It is hard-coded for fn=predictPairInternal, which is what most people use.)

heuristicsProb for a version of this function tailored for predictProbInternal rather than predictPairInternal.

Examples

```
# Use one fitted ttbModel with column 1 as criterion and columns 2,3 as
# cues.
data <- cbind(y=c(30,20,10,5), x1=c(1,1,0,0), x2=c(1,1,0,1))
ttb <- ttbModel(data, 1, c(2:3))
rowPairApply(data, heuristicsList(list(ttb), predictPairInternal))
# This outputs ttb's predictions for all 6 row pairs of data.
# (It has 6 row pairs because 4*2/2 = 6.)  It gets the predictions
# by calling ttb's predictPairInternal.

# Use the same fitted ttbModel plus a unit weight model with the same
# criterion and cues.
unit <- unitWeightModel(data, 1, c(2,3))
rowPairApply(data, heuristicsList(list(ttb, unit), predictPairInternal))
# This outputs predictions with column names 'ttbModel' and
# 'unitWeightLinearModel'.

# Use the same fitted ttbModel plus another ttbModel that has different
# cols_to_fit.  This has to be put in a separate heuristicsList function.
```

```
ttb_just_col_3 <- ttbModel(data, 1, c(3), fit_name="ttb3")
rowPairApply(data, heuristicsList(list(ttb), predictPairInternal),
  heuristicsList(list(ttb_just_col_3), predictPairInternal))
# This outputs predictions with column names 'ttbModel' and
# 'ttb3'.
```

`heuristicsProb` *Wrap fitted heuristics to pass to rowPairApply to call predictProb.*

Description

One or more fitted heuristics can be passed in. They must all implement predictProbInternal. Users will generally not use the output directly but instead pass this to rowPairApply.

Usage

```
heuristicsProb(...)
```

Arguments

`...`	A list of predictProbInternal implementers, e.g. a fitted ttb model.

Value

An object of class heuristics, which implements createFunction. Users will generally not use this directly– rowPairApply will.

See Also

`rowPairApply` which is what heuristicsProb is passed in to.

`predictProbInternal` which must be implemented by heuristics in order to use them with the heuristicsProb() wrapper function.

Examples

```
## This is typical usage:
data <- cbind(y=c(30,20,10,5), x1=c(1,1,0,0), x2=c(1,1,0,1))
ttb <- ttbModel(data, 1, c(2:ncol(data)))
rowPairApply(data, heuristicsProb(ttb))
## This outputs ttb's predictions for all 6 row pairs of data.
## (It has 6 row pairs because 4*2/2 = 6.)  It gets the predictions
## by calling ttb's predictProbInternal.
```

`highschool_dropout` *Chicago high school dropout rates.*

Description

Chicago high school dropout rates from 1995 and associated variables like average students per teacher and percent low income students. All cues are real-valued but some have N/A values. It includes rows accidentally omitted in prior research.

Usage

```
highschool_dropout
```

Format

A data frame.

Name Name of School

Running_Number Running Number

Included_in_Web_and_Web_Corrected If 1, then this row was accidentally omitted in the ABC studies from 1993

Dropout_Rate Dropout rate in percent, from 0 to 100, counting all students in grades 9 through 12 who left school permanently during the 1993-4 school year

Completeness_of_Data Completeness of data

Enrollment Enrollment as of September 30, 1993

Attendance_Rate Attendance rate in percent, from 0 to 100, averaged over the school year

Graduation_Rate Graduation rate in percent, from 0 to 100, based on freshmen who finished together 4 years later, in 1994

Parental_Involvement_Rate Parental involvement rate in percent, from 0 to 100, counted as parents who had contact with teachers as a percent of students (with no firm state rules on how to measure this)

Limited_English_Students Limited English Students in percent, from 0 to 100, based on the number of students found eligible for bilingual education

Low_Income_Students Low Income Students in percent, from 0 to 100, based on families eligible for free or reduced price lunches or are publicly supported

Average_Class_Size_Student_per_Teacher Calculated as number of students divided by number of teachers on the first day of May

Percent_White_Students Percent white students, from 0 to 100

Percent_Black_Students Percent black students, from 0 to 100

Percent_Hispanic_Students Percent hispanic students, from 0 to 100

Percent_Asian_Students Percent asian students, from 0 to 100

Percent_Minority_Teacher Percent minority teacher, from 0 to 100

Average_Composite_ACT_Score Average composite ACT Score

Reading Reading score on Illinois Goal Assessment Program (IGAP)

Math Math score on IGAP

Science Science score on IGAP

Social_Science Social science score on IGAP

Writing Writing score on IGAP

Details

The data is based on:

Morton, Felicia B. (1995). Charting a School's Course. Chicago. February, pp. 86-95.

Rodkin, Dennis. (1995). 10 Keys for Creating Top High Schools. Chicago. February, pp. 78-85.

This is the data set used in simulations by the ABC (Adaptive Behavior and Cognition) research group.

`logRegModel` *Logistic Regression model using cue differences as predictors*

Description

Create a logistic regression model by specifying columns and a dataset. It fits the model with R's glm function.

Usage

```
logRegModel(
  train_data,
  criterion_col,
  cols_to_fit,
  cue_order_fn = rankByCueValidity,
  suppress_warnings = TRUE,
  fit_name = "logRegModel"
)
```

Arguments

`train_data`	Training/fitting data as a matrix or data.frame.
`criterion_col`	The index of the column in train_data that has the criterion.
`cols_to_fit`	A vector of column indices in train_data, used to fit the criterion.
`cue_order_fn`	Optional argument as a function that orders cues. This only matters for over-specified models (e.g. too many cues for the number of rows), in which case it affects which cues are dropped. The rightmost cues in the order are dropped first, so the function rankByCueValidity means cues with the lowest cueValidity in the training set will be be dropped first. The function must have the signature function(train_data, criterion_col, cols_to_fit).

suppress_warnings
: Optional argument specifying whether glm warnings should be suppressed or not. Default is TRUE.

fit_name
: Optional The name other functions can use to label output. It defaults to the class name.

Details

This version assumes you do not want to include the intercept.

For a discussion of how logistic regression works, see: https://www.r-bloggers.com/what-does-a-generalized-linear-model-do/ Note that our criterion is the probability that row 1 is greater than row 2 when a pair is encountered.

Value

An object of class logRegModel.

---

minModel *Minimalist Model*

Description

Fit the Minimalist heuristic by specifying columns and a dataset. It searches cues in a random order, making a decision based on the first cue that discriminates (has differing values on the two objects).

Usage

```
minModel(
  train_data,
  criterion_col,
  cols_to_fit,
  reverse_cues = TRUE,
  fit_name = "minModel"
)
```

Arguments

train_data
: Training/fitting data as a matrix or data.frame.

criterion_col
: The index of the column in train_data that has the criterion.

cols_to_fit
: A vector of column indices in train_data, used to fit the criterion.

reverse_cues
: Optional parameter to reverse cues as needed. By default, the model will reverse the cue values for cues with cue validity < 0.5, so a cue with validity 0 becomes a cue with validity 1. Set this to FALSE if you do not want that, i.e. the cue stays validity 0.

fit_name
: Optional The name other functions can use to label output. It defaults to the class name.

Value

An object of class minModel, which can be passed to a variety of functions to make predictions, e.g. predictPair and percentCorrectList.

See Also

predictPairProb for prediction.

Examples

```
## Fit column (5,4) to column (1,0), having validity 1.0, and column (0,1),
## validity 0.
train_matrix <- cbind(c(5,4), c(1,0), c(0,1))
min <- minModel(train_matrix, 1, c(2,3))
predictPair(oneRow(train_matrix, 1), oneRow(train_matrix, 2), min)
```

---

oneRow *Convenience function to get one row from a matrix or data frame.*

Description

This simply calls matrix_or_data_frame[row_index,,drop=FALSE] for you but is shorter and helps you avoid forgetting drop=FALSE. The need for drop=FALSE when selecting just one row is explained here: http://www.hep.by/gnu/r-patched/r-faq/R-FAQ_56.html

Usage

```
oneRow(matrix_or_data_frame, row_index)
```

Arguments

matrix_or_data_frame	A matrix or data frame from which you want one row.
row_index	The integer index of the row

Value

The selected row of the data frame.

pairMatrix *Apply a function to all unique pairs of row indices up to num_row.*

Description

Apply a function to all unique pairs of row indices up to num_row.

Usage

```
pairMatrix(num_row, pair_evaluator_fn, also_reverse_row_pairs = FALSE)
```

Arguments

num_row	The number of rows to generate index pairs for.
pair_evaluator_fn	The function you want applied. It should accept a list of two numbers, the index of row 1 and the index of row2.
also_reverse_row_pairs	Optional parameter. When it has its default value of FALSE, it will apply every function only once to any given row pair, e.g. myFunction(1, 2). When it is true, it will also apply the function to every reverse row pair, e.g. myFunction(1, 2) and myFunction(2, 1).

Value

A matrix of the output of the function for all unique row pairs: c(pair_evaluator_fn(c(1,2), pair_evaluator_fn(c(1,3)), etc.)

---

percentCorrect *Percent correct of heuristics' predictPair on test_data.*

Description

Returns overall percent correct for all heuristics. 1. Create predictions using predictPair for all row pairs for all fitted heuristics in the list. 2. Calculate percent correct for each heuristic. Assumes the heuristics passed in have already been fitted to training data and all have the same criterion column.

Usage

```
percentCorrect(test_data, ...)
```

Arguments

test_data	Data to try to predict. Must have same criterion column and cols_to_fit as the data heuristics were fit to.
...	One or more heuristics fitted to data, e.g. the output of ttbModel.

Details

In cases where a heuristic guesses (predictPair outputs 0), percentCorrect will use the expected value, so output will be deterministic and repeatable. That is, if 10 guesses happen across the data set, percentCorrect will always allocate 5 to 1 and 5 to -1.

Value

A one-row data.frame of numbers from 0 to 100, the percent correc of each heuristic. Each column is named with the heuristic's class or the fit name.

See Also

`percentCorrectList` for a version which takes a list of heuristics.

Examples

```
df <- data.frame(y=c(30,20,10,5), name=c("a", "b", "c", "d"),
                 x1=c(1,1,0,0), x2=c(1,1,0,1))
ttb <- ttbModel(df, 1, c(3:4))
sing <- singleCueModel(df, 1, c(3:4))
percentCorrect(df, ttb, sing)
#    ttbModel singleCueModel
#  1     0.75      0.8333333
# TTB gets 75% correct while single cue model gets 83%.

# Now repeatedly sample 2 rows of the data set and see how outcomes are
# affected, tracking with the fit_name.
set.seed(1) # If you want to reproduce the same output as below.
ttb1 <- ttbModel(df[sample(nrow(df), 2),], 1, c(3:4), fit_name="fit1")
ttb2 <- ttbModel(df[sample(nrow(df), 2),], 1, c(3:4), fit_name="fit2")
ttb3 <- ttbModel(df[sample(nrow(df), 2),], 1, c(3:4), fit_name="fit3")
percentCorrect(df, ttb1, ttb2, ttb3)
#        fit1 fit2 fit3
# 1 0.8333333 0.75 0.75
```

---

percentCorrectList *Percent correct of a list of heuristics' predictPair on test_data.*

Description

Returns overall percent correct for all heuristics. 1. Create predictions using predictPair for all row pairs for all fitted heuristics in the list. 2. Calculate percent correct for each heuristic. Assumes the heuristics passed in have already been fitted to training data and all have the same criterion column.

Usage

```
percentCorrectList(test_data, fitted_heuristic_list)
```

Arguments

test_data	Data to try to predict. Must have same criterion column and cols_to_fit as the data heuristics were fit to.
fitted_heuristic_list	A list of one or more heuristics fitted to data, e.g. the output of ttbModel.

Value

A one-row data.frame of numbers from 0 to 100, the percent correc of each heuristic. Each column is named with the heuristic's class or the fit name.

See Also

percentCorrectList for a version which takes heuristics as parameters rather than wrapped in a list.

Examples

```
df <- data.frame(y=c(30,20,10,5), name=c("a", "b", "c", "d"),
                 x1=c(1,1,0,0), x2=c(1,1,0,1))
ttb <- ttbModel(df, 1, c(3:4))
sing <- singleCueModel(df, 1, c(3:4))
percentCorrectList(df, list(ttb, sing))
#    ttbModel singleCueModel
#  1     0.75      0.8333333
# TTB gets 75% correct while single cue model gets 83%.

# Now repeatedly sample 2 rows of the data set and see how outcomes are
# affected, tracking with the fit_name.
set.seed(1) # If you want to reproduce the same output as below.
ttb1 <- ttbModel(df[sample(nrow(df), 2),], 1, c(3:4), fit_name="fit1")
ttb2 <- ttbModel(df[sample(nrow(df), 2),], 1, c(3:4), fit_name="fit2")
ttb3 <- ttbModel(df[sample(nrow(df), 2),], 1, c(3:4), fit_name="fit3")
percentCorrectList(df, list(ttb1, ttb2, ttb3))
#        fit1 fit2 fit3
# 1 0.8333333 0.75 0.75
```

---

percentCorrectListNonSymmetric

percentCorrectList for non-symmetric heuristics

Description

Same as percentCorrectList but for weird heuristics that do not consistently choose the same row. When a symmetric heuristic predicts row1 > row2, then it also predicts row2 < row1. Those can be used with percentCorrectList. All heuristics built into heuristica qualify. They will get the same answers for percentCorrectList and percentCorrectListNonSymmetric. But a non-symmetric heuristic will only get correct answers for percentCorrectListNonSymmetric.

Usage

```
percentCorrectListNonSymmetric(test_data, fitted_heuristic_list)
```

Arguments

test_data	Data to try to predict. Must have same criterion column and cols_to_fit as the data heuristics were fit to.
fitted_heuristic_list	A list of one or more heuristics fitted to data, e.g. the output of ttbModel.

Value

A one-row data.frame of numbers from 0 to 100, the percent correc of each heuristic. Each column is named with the heuristic's class or the fit name.

See Also

percentCorrectList which is faster but wil only be accurate for symmetric heuristics. (percentCorrectListNonSymmetric will be accurate for both symmetric and non-symmetric heuristics, but it's slower.)

---

percentCorrectListReturnMatrix

Percent correct of heuristics' predictPair on test_data, returning a matrix.

Description

Percent correct of heuristics' predictPair on test_data, returning a matrix.

Usage

```
percentCorrectListReturnMatrix(test_data, fitted_heuristic_list)
```

Arguments

test_data	Data to try to predict. Must have same criterion column and cols_to_fit as the data heuristics were fit to.
fitted_heuristic_list	A list of one or more heuristics fitted to data, e.g. the output of ttbModel.

Value

A one-row matrix of numbers from 0 to 100, the percent correct of each heuristic. Each column is named with the heuristic's class or the fit name.

See Also

percentCorrectList for a version that returns a data.frame and includes several examples.

Examples

```
# See examples for percentCorrectList, which returns a data.frame.
```

---

predictPair *Predict which of a pair of rows has a higher criterion.*

Description

Given two rows and a fitted heuristic, returns the heuristic's prediction of whether the criterion of the first row will be greater than that of the 2nd row.

Usage

```
predictPair(row1, row2, object)
```

Arguments

row1	The first row of data. The cues object$cols_to_fit will be passed to the heuristic.
row2	The second row of data. The cues object$cols_to_fit will be passed to the heuristic.
object	The fitted heuristic, e.g. a fitted ttbModel or logRegModel. (More technically, it's any object that implements predictPairInternal.)

Value

A number in the set -1, 0, 1, where 1 means row1 is predicted to have a greater criterion, -1 means row2 is greater, and 0 is a guess or tie.

See Also

rowPairApply to get predictions for all row pairs of a matrix or data.frame.

Examples

```
##Fit column (5,4) to column (1,0), having validity 1.0, and column (0,1),
## validity 0.
train_matrix <- cbind(y=c(5,4), x1=c(1,0), x2=c(0,1))
singlecue <- singleCueModel(train_matrix, 1, c(2,3))
predictPair(oneRow(train_matrix, 1), oneRow(train_matrix, 2), singlecue)
```

predictPairProb *Predict the probability that row1 has a higher criterion than row2.*

Description

Given two rows and a fitted heuristic, returns the heuristic's predicted probability that row1's criterion will be greater than row2's.

Usage

```
predictPairProb(row1, row2, object)
```

Arguments

row1	The first row of cues (will apply cols_to_fit for you, based on object).
row2	The second row (will apply cols_to_fit for you, based on object).
object	The fitted heuristic, e.g. a fitted ttbModel or logRegModel. (More technically, it's any object that implements predictProbInternal.)

Value

A double from 0 to 1, representing the probability that row1's criterion is greater than row2's criterion. 0.5 could be a guess or tie.

See Also

rowPairApply to get predictions for all row pairs of a matrix or data.frame.

Examples

```
train_matrix <- cbind(y=c(5,4), x1=c(1,0), x2=c(0,1))
lreg <- logRegModel(train_matrix, 1, c(2,3))
predictPairProb(oneRow(train_matrix, 1), oneRow(train_matrix, 2), lreg)
```

predictPairSummary *Returns the row indices, correct answer, and predictions for all row pairs.*

Description

This makes it easy to see and evaluate predictions for all row pairs on a data set. It is intended for beginners. Advanced users can get more fine-grained control with rowPairApply.

Usage

```
predictPairSummary(test_data, ...)
```

Arguments

test_data	Data to try to predict. Must have same criterion column and cols_to_fit as the data heuristics were fit to.
...	One or more heuristics already fitted to data, e.g. the output of ttbModel.

Value

A matrix with output for indices, the correct row pair answer, and predictions for each heuristic with as many rows as row pairs in the data. The columns names are Row1, Row2, CorrectGreater, and each heuristic fit_name (which is its class name by default, e.g. ttbModel).

See Also

rowPairApply for full flexibility.

Examples

```
# Get some data and fit it with two models.
train_df <- data.frame(criterion=c(9,8,7,6), a=c(101,101,2,2), b=c(59,58,5,59))
criterion_col <- 1
ttb <- ttbModel(train_df, criterion_col, c(2:3))
lreg <- logRegModel(train_df, criterion_col, c(2:3))

# Generate predictions and correct answers with predictPairSummary.
out <- predictPairSummary(train_df, ttb, lreg)

# Find rows where the models make differing predictions, subsetting on a
# data.frame.
out_df <- data.frame(out)
out_df[out_df$ttbModel != out_df$logRegModel,]
# Outputs:
#   Row1 Row2 CorrectGreater ttbModel logRegModel
#    1    2              1        1          -1
#    3    4              1       -1           1
# So there are only two cases of differing predictions.
# 1) For row 1 vs. 2, TTB predicted 1 and logReg predicted -1.
#    CorrectGreater says 1, so TTB was right.
# 2) For row 3 vs. 4, TTB predicted -1 and logReg predicted 1.
#    CorrectGreater says -1, so logReg was right.

# Note that under the hood, the above predictPairSummary call could be
# done with rowPairApply like this:
out2 <- rowPairApply(train_df, rowIndexes(),
                     correctGreater(criterion_col), heuristics(ttb, lreg))
```

probGreater *Creates function for one column with correct probability row1 is greater.*

Description

Using rowPairApply, this can generate a column with the correct probability that row 1 > row 2 for each row pair in the test_data. It can do this using the criterion column passed in. By default, the output column is called "ProbGreater," but you can override the name with output_column_name.

Usage

```
probGreater(criterion_col, output_column_name = "ProbGreater")
```

Arguments

criterion_col	The integer index of the criterion in test_data.
output_column_name	An optional string

Details

Note this uses a very simplistic "probability" which only looks at the current row pair. It does not look at all sets of row pairs with the same profile.

Value

An object that implements createFunction. Users will generally not use this directly– rowPairApply will.

See Also

heuristicsProb is the wrapper to get the predicted probability that the first row in the row pair is greater, with output for each fitted heuristic passed to it.

rowPairApply which has examples of using this.

regInterceptModel *Linear regression wrapper for hueristica*

Description

A wrapper to create a lm model just specifying columns, generating a model formula for you. This makes it easier to run automated comparisons with other models in heuristica.

Usage

```
regInterceptModel(train_matrix, criterion_col, cols_to_fit)
```

Arguments

train_matrix	A matrix (or data.frame) of data to train (fit) the model with.
criterion_col	The index of the criterion column– "y" in the formula.
cols_to_fit	A vector of column indexes to fit– the "x's" in the formula.

Details

This version assumes you always want to include the intercept.

Value

An object of class regInterceptModel, which is a subclass of lm.

See Also

regModel for a version that excludes the intercept.

predict.lm for prediction.

predictPairProb for predicting between a pair of rows.

---

regModel *Linear regression (no intercept) wrapper for hueristica*

Description

A wrapper to create a lm model just specifying columns, generating a model formula for you __without an intercept__. This makes it easier to run automated comparisons with other models in heuristica.

Usage

```
regModel(train_matrix, criterion_col, cols_to_fit, fit_name = "regModel")
```

Arguments

train_matrix	A matrix (or data.frame) of data to train (fit) the model with.
criterion_col	The index of the criterion column– "y" in the formula.
cols_to_fit	A vector of column indexes to fit– the "x's" in the formula.
fit_name	Optional The name other functions can use to label output. It defaults to the class name.

Details

This version assumes you do NOT want to include the intercept. Excluding the intercept typically has higher out-of-sample accuracy if the goal is predicting rank order because the intercept does not affect the ranking, but estimating it wastes a degree of freedom.

Value

An object of class regModel, which is a subclass of lm.

See Also

`lm` for the regression function being wrapped.

`predictPair` for predicting whether row1 is greater. greater.

`reverseRowsAndReverseColumns`
Reverse rows and columns of data

Description

This matrix [,1] [,2] [1,] 1 2 [2,] 3 4 becomes [,1] [,2] [1,] 4 3 [2,] 2 1

Usage

```
reverseRowsAndReverseColumns(data)
```

Arguments

`data`	A data.frame or matrix.

Value

A data.frame or matrix with rows reversed and columns reversed.

rowIndexes *Wrapper to output two columns, row 1 and row 2.*

Description

Using rowPairApply, this can generate two columns, which by default are called "Row1" and "Row2"

Usage

```
rowIndexes(rowIndexColNames = c("Row1", "Row2"))
```

Arguments

rowIndexColNames	An optional vector of 2 strings for column names.

Value

An object of class rowIndexes, which implements createFunction. Users will generally not use this directly– rowPairApply will.

See Also

createFunction which is what the returned object implements.

rowPairApply which uses createFunction.

rowPairApply *Apply functions to all row pairs.*

Description

Apply functions like heuristic predictions to all row pairs in a matrix or data.frame. This does not accept arbitrary functions– they must be functions designed to be run by rowPairApply.

Usage

```
rowPairApply(test_data, ...)
```

Arguments

test_data	The data to apply the functions to as a matrix or data.frame. Heuristics must have already been fitted to trying data and must include the same criterion_col and cols_to_fit.
...	The functions that generate the functions to apply, such as heuristics(ttb) and correctGreater(col)– see example below.

Value

A matrix of outputs from the functions. The number of rows is based on the number of row pairs in test_data. If the input has N rows, the output will have N x (N-1) rows. The number of columns will be at least the number of functions but may be more as some functions may output more than one column.

See Also

heuristics and heuristics to wrap heuristics to be applied.

rowIndexes to get apply to output row indexes for the pair.

correctGreater to get the correct output based on the criterion column. (CorrectGreater should be used with heuristics while probGreater should be used with heuristicsProb.)

Examples

```
## Fit two models to data.
data <- cbind(y=c(30,20,10,5), x1=c(1,1,0,0), x2=c(1,1,0,1))
ttb <- ttbModel(data, 1, c(2:ncol(data)))
lreg <- logRegModel(data, 1, c(2:ncol(data)))

## Generate predictions for all row pairs for these two models:
rowPairApply(data, heuristics(ttb, lreg))
## Returns a matrix of 2 columns, named ttbModel and regModel, and 6 rows.
## The original data had 4 rows, meaning there are 4*3/2 = 6 row pairs.

## To see which row pair is which row, use rowIndexes:
rowPairApply(data, rowIndexes(), heuristics(ttb, lreg))
## Returns a matrix with columns Row1, Row2, ttbModel, logRegModel.
## (RowIndexes returns *two* columns.)

## To see whether the first row was actually greater than the second in the
## row pair, use correctGreater and give it the criterion column index, in
## this case 1.
rowPairApply(data, heuristics(lreg, ttb), correctGreater(1))
## Returns a matrix with columns logRegModel, ttbModel,
## CorrectGreater.  Values are -1, 0, or 1.

## To do the same analysis for the *probabilty* that the first row is
## greater. use heuristicsProb and probGreater.  Warning: Not all heuristica
## models have implemented the prob greater function.
rowPairApply(data, heuristicsProb(lreg, ttb), probGreater(1))
## Returns a matrix with columns logRegModel, ttbModel, ProbGreater.
## Values range from 0.0 to 1.0.
```

`rowPairApplyList` *Apply list of functions to all row pairs.*

Description

Apply a list of functions like heuristic predictions to all row pairs in a matrix or data.frame. This does not accept arbitrary functions– they must be functions designed to be run by rowPairApply.

Usage

```
rowPairApplyList(
  test_data,
  function_creator_list,
  also_reverse_row_pairs = FALSE
)
```

Arguments

`test_data`	The data to apply the functions to as a matrix or data.frame. Heuristics must have already been fitted to trying data and must include the same criterion_col and cols_to_fit.
`function_creator_list`	List of the functions that generate the functions to apply, such as heuristics(ttb) and correctGreater(col).
`also_reverse_row_pairs`	Optional parameter. When it has its default value of FALSE, it will apply every function only once to any given row pair, e.g. myFunction(row1, row2). When it is true, it will also apply the function to every reverse row pair, e.g. myFunction(row2, row1).

Value

A matrix of outputs from the functions. The number of rows is based on the number of row pairs in test_data. If the input has N rows, the output will have N x (N-1) rows. The number of columns will be at least the number of functions but may be more as some functions may output more than one column.

See Also

`rowPairApply` for no need to use a list. Examples and details are there.

Examples

```
# This function is called like
# rowPairApplyList(data, list(heuristics(ttb, reg)))
# instead of
# rowPairApply(data, heuristics(ttb, reg))
# See rowPairApply for details.
```

singleCueModel *Single Cue Model*

Description

Create a single cue model by specifying columns and a dataset. It sorts cues in order of cueValidity and uses the cue with the highest cueValidity. If the cue does not discriminate it guesses randomly. If several cues have the highest validity, then on each prediction it randomly selects which one to use (so it might not give the same answer every time).

Usage

```
singleCueModel(
  train_data,
  criterion_col,
  cols_to_fit,
  reverse_cues = TRUE,
  fit_name = "singleCueModel"
)
```

Arguments

train_data	Training/fitting data as a matrix or data.frame.
criterion_col	The index of the column in train_data that has the criterion.
cols_to_fit	A vector of column indices in train_data, used to fit the criterion.
reverse_cues	Optional parameter to reverse cues as needed. By default, the model will reverse the cue values for cues with cue validity < 0.5, so a cue with validity 0 becomes a cue with validity 1. Set this to FALSE if you do not want that, i.e. the cue stays validity 0.
fit_name	Optional The name other functions can use to label output. It defaults to the class name.

Details

This single cue model follows the definition used in this reference: Hogarth, R. & Karelaia, N. (2007). Heuristic and Linear Models of Judgment: Matching Rules and Environments. Psychological Review. 114(3), pp.733-758. Note that other researchers have sometimes used other measures than cue validity to select the single cue to be used.

Value

An object of class singleCueModel, which can be passed to a variety of functions to make predictions, e.g. predictPair and percentCorrectList.

See Also

predictPairProb for prediction.

Examples

```
##Fit column (5,4) to column (1,0), having validity 1.0, and column (0,1),
## validity 0.
train_matrix <- cbind(y=c(5,4), x1=c(1,0), x2=c(0,1))
singlecue <- singleCueModel(train_matrix, 1, c(2,3))
predictPair(oneRow(train_matrix, 1), oneRow(train_matrix, 2), singlecue)
```

`statsFromConfusionMatrix` *Accuracy, sensitivity, specificity, and precision of 2x2 confusion matrix.*

Description

In heuristica, "positive" means the row1 > row2. Other heuristica create confusion matrices with the expected layout, but below is documentation of that layout. A package like 'caret' offers a more general-purpose confusion matrix.

Usage

```
statsFromConfusionMatrix(confusion_matrix)
```

Arguments

`confusion_matrix` A 2x2 confusion matrix.

Details

This assumes the input matrix is 2x2 and will STOP if not. It also assumes negatives are left and higher, and predictions are the rows, that is: true negative [-1,-1] false negative [-1,1] false negative [1, -1] true positive [1, 1]

The outputs are defined as: accuracy = (true positive + true negative) / all sensitivity = true positive rate = true positive / all positive (sensitivity is also called recall) specificity = true negative rate = true negative / all negative precision = positive predictive value = true positive

Value

A list with accuracy, sensitivity, specificity, and precision

`ttbGreedyModel` *Greedy Take The Best*

Description

A variant of the Take The Best heuristic with a different cue order, namely using conditional cue validity, where the validity of a cue is judged only on row pairs not already decided by prior cues. Specifically, it uses the cue ranks returned by `conditionalCueValidityComplete`.

Usage

```
ttbGreedyModel(
  train_data,
  criterion_col,
  cols_to_fit,
  fit_name = "ttbGreedyModel"
)
```

Arguments

`train_data`	Training/fitting data as a matrix or data.frame.
`criterion_col`	The index of the column in train_data that has the criterion.
`cols_to_fit`	A vector of column indices in train_data, used to fit the criterion.
`fit_name`	Optional The name other functions can use to label output. It defaults to the class name. It is useful to change this to a unique name if you are making multiple fits, e.g. "ttb1", "ttb2", "ttbNoReverse."

Value

An object of `class` ttbGreedyModel, which can be passed in to `predictPair`.

References

Martignon, L., & Hoffrage, U. (2002). Fast, frugal, and fit: Simple heuristics for paired comparisons. Theory and Decision, 52: 29-71.

See Also

`conditionalCueValidityComplete` for the metric used to sort cues.

`ttbModel` for the original version of Take The Best.

`predictPair` for predicting whether row1 is greater.

`predictPairProb` for predicting the probability row1 is greater.

Examples

```
## A data set where Take the Best and Greedy Take the Best disagree.
matrix <- cbind(y=c(3:1), x1=c(1,0,0), x2=c(1,0,1))
ttb <- ttbModel(matrix, 1, c(2,3))
ttb$cue_validities
# Returns
#  x1  x2
# 1.0 0.5
ttbG <- ttbGreedyModel(matrix, 1, c(2:3))
ttbG$cue_validities
# Returns
#  x1  x2
#   1   1
# because after using x1, only decisions between row 2 and 3 are left,
# and x2 gets 100% right  on those (after reversal).  However, these
# cue_validities depend on using x1, first, so cue_rank is key.
ttbG$cue_ranks
# Returns
#  x1  x2
#   1   2

# Now see how this affects predictions on row 2 vs. 3.
# Take the best guesses (output 0).
predictPair(oneRow(matrix, 2), oneRow(matrix, 3), ttb)
# Greedy Take The Best selects row 2 (output 1).
predictPair(oneRow(matrix, 2), oneRow(matrix, 3), ttbG)
```

`ttbModel` *Take The Best*

Description

An implementation of the Take The Best heuristic. It sorts cues in order of `cueValidity`, making a decision based on the first cue that discriminates (has differing values on the two objects).

Usage

```
ttbModel(
  train_data,
  criterion_col,
  cols_to_fit,
  reverse_cues = TRUE,
  fit_name = "ttbModel"
)
```

Arguments

train_data	Training/fitting data as a matrix or data.frame.
criterion_col	The index of the column in train_data that has the criterion.
cols_to_fit	A vector of column indices in train_data, used to fit the criterion.
reverse_cues	Optional parameter to reverse cues as needed. By default, the model will reverse the cue values for cues with cue validity < 0.5, so a cue with validity 0 becomes a cue with validity 1. Set this to FALSE if you do not want that, i.e. the cue stays validity 0.
fit_name	Optional The name other functions can use to label output. It defaults to the class name. It is useful to change this to a unique name if you are making multiple fits, e.g. "ttb1", "ttb2", "ttbNoReverse."

Details

Cues that are tied in validity are sorted once at fitting time, and that order is used consistently for all predictions with that model. But re- fitting may lead to a different cue order. (An alternative would be to randomly re-order on every prediction.)

Value

An object of class ttbModel, which can be passed to a variety of functions to make predictions, e.g. predictPair and percentCorrectList.

References

Gigerenzer, G. & Goldstein, D. G. (1996). "Reasoning the fast and frugal way: Models of bounded rationality". Psychological Review, 103, 650-669.

Wikipedia's entry on https://en.wikipedia.org/wiki/Take-the-best_heuristic.

See Also

cueValidity for the metric used to sort cues.

predictPair for predicting whether row1 is greater.

predictPairProb for predicting the probability row1 is greater.

percentCorrectList for the accuracy of predicting all row pairs in a matrix or data.frame.

Examples

```
# Fit column 1 (y) to columns 2 and 3 (x1 and x2) of train_matrix.
train_matrix <- cbind(y=c(5,4), x1=c(1,0), x2=c(0,0))
ttb <- ttbModel(train_matrix, 1, c(2,3))
# Have ttb predict whether row 1 or 2 has a greater value for y.  The
# output is 1, meaning it predicts row1 is bigger.
predictPair(oneRow(train_matrix, 1), oneRow(train_matrix, 2), ttb)

# Now ask it the reverse-- predict whther row 2 or row 1 is greater.  The
# output is -1, meaning it still predicts row1 is bigger.  (It is a
```

```
# symmetric heuristic.)
predictPair(oneRow(train_matrix, 2), oneRow(train_matrix, 1), ttb)

# But this test data results in an incorrect prediction-- that row1 has a
# smaller criterion than row2-- because x1 has a reversed direction.
test_matrix <- cbind(y=c(5,4), x1=c(0,1), x2=c(0,0))
predictPair(oneRow(test_matrix, 1), oneRow(test_matrix, 2), ttb)
```

`unitWeightModel` *Unit-weight linear model*

Description

Unit-weight linear model inspired by Robyn Dawes. Unit Weight Model assigns unit (+1 or -1) weights based on `cueValidity`.

- A cue validity > 0.5 results in a weight of +1.
- A cue validity < 0.5 results in a weight of -1.

This version differs from others in that it uses a weight of 0 if cue validity is 0.5 (rather than randomly assigning +1 or -1) to give faster convergence of average accuracy.

Usage

```
unitWeightModel(
  train_data,
  criterion_col,
  cols_to_fit,
  reverse_cues = TRUE,
  fit_name = "unitWeightModel"
)
```

Arguments

`train_data`	Training/fitting data as a matrix or data.frame.
`criterion_col`	The index of the column in train_data that has the criterion.
`cols_to_fit`	A vector of column indices in train_data, used to fit the criterion.
`reverse_cues`	Optional parameter to reverse cues as needed.
`fit_name`	Optional The name other functions can use to label output. It defaults to the class name.

Value

An object of `class` unitWeightModel. This is a list containing at least the following components:

- "cue_validities": A list of cue validities for the cues in order of cols_to_fit.
- "linear_coef": A list of linear model coefficients (-1 or +1) for the cues in order of cols_to_fit. (It can only return -1's if reverse_cues=TRUE.)

References

Wikipedia's entry on https://en.wikipedia.org/wiki/Unit-weighted_regression.

See Also

cueValidity for the metric used to to determine cue direction.

predictPair for predicting whether row1 is greater.

predictPairProb for predicting the probability row1 is greater.

---

validityWeightModel *Validity Weight Model, a linear model weighted by cue validities*

Description

Validity Weight Model is a linear model with weights calculated by cueValidity.

Usage

```
validityWeightModel(
  train_data,
  criterion_col,
  cols_to_fit,
  reverse_cues = TRUE,
  fit_name = "validityWeightModel"
)
```

Arguments

train_data	Training/fitting data as a matrix or data.frame.
criterion_col	The index of the column in train_data that has the criterion.
cols_to_fit	A vector of column indices in train_data, used to fit the criterion.
reverse_cues	Optional parameter to reverse cues as needed. By default, the model will reverse the cue values for cues with cue validity < 0.5, so a cue with validity 0 becomes a cue with validity 1. Set this to FALSE if you do not want that, i.e. the cue stays validity 0.
fit_name	Optional The name other functions can use to label output. It defaults to the class name.

Value

An object of class validityWeightModel. This is a list containing at least the following components:

- "cue_validities": A list of cue validities for the cues in order of cols_to_fit.
- "linear_coef": Same as cue validities for this model.

See Also

`cueValidity` for the metric used to to determine cue direction.

`predictPair` for predicting whether row1 is greater.

`predictPairProb` for predicting the probability row1 is greater.

Index