Package: CPTtools (via r-universe)

June 11, 2024

Title Tools for Creating Conditional Probability Tables

Version 0.8-3

Date 2023/07/18

Author Russell Almond

Maintainer Russell Almond <ralmond@fsu.edu>

Description Provides support parameterized tables for Bayesian networks, particularly the IRT-like DiBello tables. Also, provides some tools for visualing the networks.

License Artistic-2.0

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

URL http://pluto.coe.fsu.edu/RNetica

Depends methods, R (>= 3.5.0)

Imports grDevices, graphics, stats, utils, einsum, tidyr, dplyr

Suggests ggplot2, lattice, knitr, rmarkdown, tidyverse, googlesheets4, cowplot, vdiffr, withr, testthat (>= 3.0.0)

VignetteBuilder knitr

Support c('Bill & Melinda Gates Foundation grant ``Games as Learning/Assessment: Stealth Assessment'' (no. 0PP1035331, Val Shute, PI)', 'National Science Foundation grant ``DIP: Game-based Assessment and Support of STEM-related Competencies'' (no. 1628937, Val Shute, PI)', 'National Science Foundation grant ``Mathematical Learning via Architectural Design and Modeling Using E-Rebuild.'' (no. 1720533, Fengfeng Ke, PI)', 'Institute of Educational Statistics Grant: ``Exploring adaptive cognitive and affective learning support for next-generation STEM learning games.'' (no. R305A170376-20, Val Shute and Russell Almond, PIs)')

Config/testthat/edition 3

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

RemoteUrl https://github.com/ralmond/CPTtools

RemoteRef HEAD

RemoteSha daf1d2ab92741e96c773c81ac5c63583a7f2ef31

Contents

`CPTtools-package` *Tools for Creating Conditional Probability Tables*

Description

Provides support parameterized tables for Bayesian networks, particularly the IRT-like DiBello tables. Also, provides some tools for visualing the networks.

Details

The DESCRIPTION file: This package was not yet installed at build time.

CPTtools is a collection of various bits of R code useful for processing Bayes net output. Some were designed to work with ETS's proprietary StatShop code, and some with RNetica. The code collected in this package is all free from explicit dependencies on the specific Bayes net package and will hopefully be useful with other systems as well.

The majority of the code are related to building conditional probability tables (CPTs) for Bayesian networks. The package has two output representations for a CPT. The first is a `data.frame` object where the first several columns are factor variables corresponding the the parent variables, and the remaining columns are numeric variables corresponding to the state of the child variables. The rows represent possible configurations of the parent variables. An example is shown below.

```
       S1       S2       Full    Partial       None
1    High     High 0.81940043 0.15821522 0.02238436
2  Medium     High 0.46696668 0.46696668 0.06606664
3     Low     High 0.14468106 0.74930671 0.10601223
4    High   Medium 0.76603829 0.14791170 0.08605000
5  Medium   Medium 0.38733177 0.38733177 0.22533647
6     Low   Medium 0.10879020 0.56342707 0.32778273
7    High      Low 0.65574465 0.12661548 0.21763987
8  Medium      Low 0.26889642 0.26889642 0.46220715
9     Low      Low 0.06630741 0.34340770 0.59028489
10   High LowerYet 0.39095414 0.07548799 0.53355787
11 Medium LowerYet 0.11027649 0.11027649 0.77944702
12    Low LowerYet 0.02337270 0.12104775 0.85557955
```

The second representation is a table (`matrix`) with just the numeric part. Two approaches to building these tables from parameters are described below. The more flexible discrete partial credit model is used for the basis of the parameterized networks in the `Peanut` package.

In addition to the code for building partial credit networks, this package contains some code for building Bayesian network structures from (inverse) correlation matrixes, and graphical displays for Bayes net output. The latter includes some diagnostic plots and additional diagnostic tests.

Discrete Partial Credit Framework

The original parameterization for creating conditional probability tables based on Almond et al (2001) proved to be insufficiently flexible. Almond (2015) describes a newer parameterization based on three steps:

1. Translate the parent variables onto a numeric effective theta scale (`effectiveThetas`).
2. Combine the parent effective thetas into a single effective theta using a combination rule (`Compensatory`, `OffsetConjunctive`).
3. Convert the effective theta for each row of the table into conditional probabilities using a link function (`gradedResponse`, `partialCredit`, `normalLink`).

The `partialCredit` link function is particularly flexible as it allows different parameterizations and different combination rules for each state of the child variable. This functionality is best captured by the two high level functions:

`calcDPCTable` Creates the probability table for the discrete partial credit model given the parameters.

`mapDPC` Finds an MAP estimate for the parameters given an observed table of counts.

This parameterization serves as basis for the model used in the `Peanut` package.

Other parametric CPT models

The first two steps of the discrete partial credit framework outlined above are due to a suggestion by Lou DiBello (Almond et al, 2001). This lead to an older framework, in which the link function was hard coded into the conditional probability table formation. The models were called DiBello-*XX*, where *XX* is the name of the link function. Almond et al. (2015) describes several additional examples.

`calcDDTable` Calculates DiBello-Dirichlet model probability and parameter tables.

`calcDNTable` Creates the probability table for DiBello-Normal distribution. This is equivalent to using the `normalLink` in the DPC framework. This also uses a link scale parameter.

`calcDSTable` Creates the probability table for DiBello-Samejima distribution. This is equivalent to using the `gradedResponse` in the DPC framework.

`calcDSllike` Calculates the log-likelihood for data from a DiBello-Samejima (Normal) distribution.

Diez (1993) and Srinivas (1993) describe an older parametric framework for Bayes nets based on the noisy-or or noisy-max function. These are also available.

`calcNoisyAndTable` Calculate the conditional probability table for a Noisy-And or Noisy-Min distribution.

`calcNoisyOrTable` Calculate the conditional probability table for a Noisy-Or distribution.

Building Bayes nets from (inverse) correlation matrixes

Almond (2010) noted that in many cases the best information about the relationship among variables came from a procedure that produces a correlation matrix (e.g., a factor analysis). Applying a trick from Whittaker (1990), connecting pairs of nodes corresponding to nonzero entries in an inverse correlation matrix produces an undirected graphical model. Ordering in the nodes in a perfect ordering allows the undirected model to be converted into a directed model (Bayesian network). The conditional probability tables can then be created through a series of regressions.

The following functions implement this protocol:

`structMatrix` Finds graphical structure from a covariance matrix.

`mcSearch` Orders variables using Maximum Cardinality search.

`buildParentList` Builds a list of parents of nodes in a graph.

`buildRegressions` Creates a series of regressions from a covariance matrix.

`buildRegressionTables` Builds conditional probability tables from regressions.

Other model construction tools

These functions are a grab bag of lower level utilities useful for building CPTs:

`areaProbs` Translates between normal and categorical probabilities.

`numericPart` Splits a mixed data frame into a numeric matrix and a factor part..

`dataTable` Constructs a table of counts from a setof discrete observations..

`eThetaFrame` Constructs a data frame showing the effective thetas for each parent combination..

`effectiveThetas` Assigns effective theta levels for categorical variable.

`getTableStates` Gets meta data about a conditional probability table..

`rescaleTable` Rescales the numeric part of the table.

`scaleMatrix` Scales a matrix to have a unit diagonal.

`scaleTable` Scales a table according to the Sum and Scale column.

Bayes net output displays and tests

Almond et al. (2009) suggested using hanging barplots for displaying Bayes net output and gives several examples. The function `stackedBars` produces the simple version of this plot and the function `compareBars` compares two distributions (e.g., prior and posterior). The function `buildFactorTab` is useful for building the data and the function `colorspread` is useful for building color gradients.

Madigan, Mosurski and Almond (1997) describe a graphical weight of evidence balance sheet (see also Almond et al, 2015, Chapter 7; Almond et al, 2013). The function `woeHist` calculates the weights of evidence for a series of observations and the function `woeBal` produces a graphical display.

Sinharay and Almond (2006) propose a graphical fit test for conditional probability tables (see also, Almond et al, 2015, Chapter 10). The function `OCP` implements this test, and the function `betaci` creates the beta credibility intervals around which the function is built.

The key to Bayesian network models are the assumptions of conditional independence which underlie the model. The function `localDepTest` tests these assumptions based on observed (or imputed) data tables.

The function `mutualInformation` calculates the mutual information of a two-way table, a measure of the strength of association. This is similar to the measure used in many Bayes net packages (e.g., `MutualInfo`).

Data sets

Two data sets are provided with this package:

`ACED` Data from ACED field trial (Shute, Hansen, and Almond, 2008). This example is based on a field trial of a Bayesian network based Assessment for Learning system, and contains both item-level response and high-level network summaries. A complete description of the Bayes net can be found at `http://ecd.ralmond.net/ecdwiki/ACED/ACED`.

`MathGrades` Grades on 5 mathematics tests from Mardia, Kent and Bibby (from Whittaker, 1990).

Index

Complete index of all functions.

Index: This package was not yet installed at build time.

Acknowledgements

We are grateful to support from the following projects for supporting the work in the development and maintenance of this package.

- Bill \& Melinda Gates Foundation grant "Games as Learning/Assessment: Stealth Assessment" (\#0PP1035331, Val Shute, PI)
- National Science Foundation grant "DIP: Game-based Assessment and Support of STEM-related Competencies" (\#1628937, Val Shute, PI).
- National Scient Foundation grant "Mathematical Learning via Architectual Design and Modeling Using E-Rebuild." (\#1720533, Fengfeng Ke, PI).

Author(s)

Russell Almond

Maintainer: Russell Almond <ralmond@fsu.edu>

References

Almond, R.G. (2015). An IRT-based Parameterization for Conditional Probability Tables. Paper submitted to the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence conference.

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer.

Almond, R. G. (2010). 'I can name that Bayesian network in two matrixes.' *International Journal of Approximate Reasoning.* **51**, 167-178.

Almond, R. G., Shute, V. J., Underwood, J. S., and Zapata-Rivera, J.-D (2009). Bayesian Networks: A Teacher's View. *International Journal of Approximate Reasoning.* **50**, 450-460.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

Diez, F. J. (1993) Parameter adjustment in Bayes networks. The generalized noisy OR-gate. In Heckerman and Mamdani (eds) *Uncertainty in Artificial Intelligence 93.* Morgan Kaufmann. 99–105.

Muraki, E. (1992). A Generalized Partial Credit Model: Application of an EM Algorithm. *Applied Psychological Measurement*, **16**, 159-176. DOI: 10.1177/014662169201600206

Samejima, F. (1969) Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph No. 17*, **34**, (No. 4, Part 2).

Shute, V. J., Hansen, E. G., & Almond, R. G. (2008). You can't fatten a hog by weighing it—Or can you? Evaluating an assessment for learning system called ACED. *International Journal of Artificial Intelligence and Education*, **18**(4), 289-316.

Sinharay, S. and Almond, R.G. (2006). Assessing Fit of Cognitively Diagnostic Models: A case study. *Educational and Psychological Measurement*. **67**(2), 239–257.

Srinivas, S. (1993) A generalization of the Noisy-Or model, the generalized noisy OR-gate. In Heckerman and Mamdani (eds) *Uncertainty in Artificial Intelligence 93.* Morgan Kaufmann. 208–215.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics*. Wiley.

Madigan, D., Mosurski, K. and Almond, R. (1997) Graphical explanation in belief networks. *Journal of Computational Graphics and Statistics*, **6**, 160-181.

Almond, R. G., Kim, Y. J., Shute, V. J. and Ventura, M. (2013). Debugging the Evidence Chain. In Almond, R. G. and Mengshoel, O. (Eds.) *Proceedings of the 2013 UAI Application Workshops: Big Data meet Complex Models and Models for Spatial, Temporal and Network Data (UAI2013AW)*, 1-10. http://ceur-ws.org/Vol-1024/paper-01.pdf

See Also

RNetica ~~ Peanut ~~

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

## New Discrete Partial Credit framework:

## Complex model, different rules for different levels
cptPC2 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                         list(full=log(1),partial=log(c(S1=1,S2=.75))),
                         betas=list(full=c(0,999),partial=1.0),
                         rule=list("OffsetDisjunctive","Compensatory"))

## Graded Response using the older DiBello-Samejima framework.
cptGraded <- calcDSTable(list(S1=skill1l),gradeL, 0.0, 0.0, dinc=c(.3,.4,.3))

## Building a Bayes net from a correlation matrix.
data(MathGrades)
pl <- buildParentList(structMatrix(MathGrades$var),"Algebra")
rt <- buildRegressions(MathGrades$var,MathGrades$means,pl)
tabs <- buildRegressionTables(rt, MathGrades$pvecs, MathGrades$means,
                              sqrt(diag(MathGrades$var)))

## Stacked Barplots:
margins.prior <- data.frame (
 Trouble=c(Novice=.19,Semester1=.24,Semester2=.28,Semseter3=.20,Semester4=.09),
 NDK=c(Novice=.01,Semester1=.09,Semester2=.35,Semseter3=.41,Semester4=.14),
 Model=c(Novice=.19,Semester1=.28,Semester2=.31,Semseter3=.18,Semester4=.04)
)

margins.post <- data.frame(
 Trouble=c(Novice=.03,Semester1=.15,Semester2=.39,Semseter3=.32,Semester4=.11),
 NDK=c(Novice=.00,Semester1=.03,Semester2=.28,Semseter3=.52,Semester4=.17),
 Model=c(Novice=.10,Semester1=.25,Semester2=.37,Semseter3=.23,Semester4=.05))

stackedBars(margins.post,3,
            main="Marginal Distributions for NetPASS skills",
            sub="Baseline at 3rd Semester level.",
            cex.names=.75, col=hsv(223/360,.2,0.10*(5:1)+.5))

compareBars(margins.prior,margins.post,3,c("Prior","Post"),
            main="Margins before/after Medium Trouble Shooting Task",
            sub="Observables:  cfgCor=Medium, logCor=High, logEff=Medium",
            legend.loc = "topright",
            cex.names=.75, col1=hsv(h=.1,s=.2*1:5-.1,alpha=1),
            col2=hsv(h=.6,s=.2*1:5-.1,alpha=1))

## Weight of evidence balance sheets
sampleSequence <- read.csv(system.file("testFiles","SampleStudent.csv",
```

```
                                        package="CPTtools"),
                              header=TRUE,row.names=1)

woeBal(sampleSequence[,c("H","M","L")],c("H"),c("M","L"),lcex=1.25)


### Observable Characteristic Plot
pi <- c("+"=.15,"-"=.85)
nnn <- c("(0,0,0)"=20,"(0,0,1)"=10,
         "(0,1,0)"=10,"(0,1,0)"=5,
         "(1,0,0)"=10,"(1,0,1)"=10,
         "(1,1,1)"=10,"(1,1,1)"=25)
xx1 <- c("(0,0,0)"=2,"(0,0,1)"=5,
       "(0,1,0)"=1,"(0,1,1)"=3,
       "(1,0,0)"=0,"(1,0,1)"=2,
       "(1,1,0)"=5,"(1,1,1)"=24)
grouplabs <- c(rep("-",3),"+")
grouplabs1 <- rep(grouplabs,each=2)
OCP2 (xx1,nnn,grouplabs1,pi,c("-","+"),ylim=c(0,1), reflty=c(2,4),
      setlabs=c("Low Skill3","High Skill3"),setat=-.8,
      main="Data for which Skill 3 is relevant")
```

`ACED.scores` *Data from ACED field trial*

Description

ACED (Adaptive Content with Evidence-Based Diagnosis; Shute, Hansen and Almond, 2008) is a Bayes net based assessment system which featured: (a) adaptive item selection and (b) extended feedback for incorrect items. This data contains both item level and pretest/posttest data from a field trial of the ACED system.

Usage

```
data("ACED")
```

Format

ACED contains 3 primary `data.frame` objects and some supplementary data.

All of the data tables have two variables which can serve as keys. `SubjID` and `AltID`. Either can be used as a primary key in joins. Note that the first two digits of the `AltID` gives the session (i.e., class) that the student was in. Note also that students in the control group have only pretest and posttest data; hence they do not appear in `ACED.items`, `ACED.scores` or `ACED.splitHalves`.

`ACED.scores` is data frame with 230 observations on 74 variables. These are mostly high-level scores from the Bayesian network.

`Cond_code` a factor giving the experimental condition for this student, the levels are "adaptive_acc", "adaptive_full", "linear_full", and "control". Note that there are no control students in this data set.

`Sequencing` a factor describing whether the sequence of items was `Linear` or `Adaptive`

`Feedback` a factor describing whether the feedback for incorrect items was `Extended` or `AccuracyOnly`

`All_Items` a numeric vector giving the number of items in ACED

`Correct` a numeric vector giving the number of items the student got correct

`Incorr` a numeric vector giving the number of items the student got incorrect

`Remain` a numeric vector giving the number of items not reached or skipped

`ElapTime` a difftime vector giving the total time spent on ACED (in seconds) The next group of columns give "scores" for each of the nodes in the Bayesian network. Each node has four scores, and the columns are names p*nodeScoreType* where *node* is replaced by one of the short codes in `ACED.allSkills`.

p***node***`H` a numeric vector giving the probability *node* is in the high state

p***node***`M` a numeric vector giving the probability *node* is in the medium state

p***node***`L` a numeric vector giving the probability *node* is in the low state

`EAP`***node*** the expected a posteriori value of *node* assuming an equal interval scale, where `L=0`, `M=1` and `H=2`

`MAP`***node*** a factor vector giving maximum a posteriori value of *node*, i.e., `which.max(p`*node*`H, p`*node*`M, p`*node*`L)`.

`ACED.skillNames` a list with two components, `long` and `short` giving the long (spelled out in CamelCase) and short names for the skills is a character vector giving the abbreviations used for the node/skill/attributes names.

`ACED.items` is data frame with 230 observations on 73 variables. These are mostly item-level scores from the field trial. The first two columns are `SubjID` and `AltID`. The remaining columns correspond to ACED internal tasks, and are coded 1 for correct, 0 for incorrect, and `NA` for not reached.

`ACED.taskNames` is essentially the row names from `ACED.items`. The naming scheme for the tasks reflect the skills measured by the task. The response variable names all start with `t` (for task) followed by the name of one or more skills tapped by the task (if there is more than one, then the first one is "primary".) This is followed by a numeric code, 1, 2 or 3, giving the difficulty (easy, medium or hard) and a letter (a, b or c) used to indicate alternate tasks following the same task model.

`ACED.prePost` is data frame with 290 observations on 32 variables giving the results of the pretest and posttest.

`SubjID` ID assigned by study team, "S" followed by 3 digits. Primary key.

`AltID` ID assigned by the ACED software vendor. Pattern is "sXX-YY", where *XX* is the session and *YY* is a student with the session.

`Classroom` A factor correpsonding to the student's class.

`Gender` A factor giving the student's gender (I'm not sure if this is self-report or administrative records.)

Race A factor giving the student's self-reported race. The codebook is lost.

Level_Code a factor variable describing the academic track of the student with levels Honors, Academic, Regular, Part 1, Part 2 and ELL. The codes Part 1 and Part 2 refer to special education students in Part 1 (mainstream classroom) or Part 2 (sequestered).

pre_scaled scale score (after equating) on pretest

post_scaled scale score (after equating) on posttest

gain_scaled post_scaled - pre_scaled

Form_Order a factor variables describing whether (AB) Form A was the pretest and Form B was the posttest or (BA) vise versa.

PreACorr number of correct items on Form A for students who took Form A as a pretest

PostBCorr number of correct items on Form B for students who took Form B as a posttest

PreBCorr number of correct items on Form B for students who took Form B as a pretest

PostACorr number of correct items on Form A for students who took Form A as a posttest

PreScore a numeric vector with either the non-missing value from PreACorr and PreBCorr

PostScore a numeric vector with either the non-missing value from PostACorr and PostBCorr

Gain PostScore - PreScore

preacorr_adj PreACorr adjusted to put forms A and B on the same scale

postbcorr_adj PostBCorr adjusted to put forms A and B on the same scale

prebcorr_adj PreBCorr adjusted to put forms A and B on the same scale

postacorr_adj PostACorr adjusted to put forms A and B on the same scale

Zpreacorr_adj standardized version of preacorr_adj

Zpostbcorr_adj standardized version of postbcorr_adj

Zprebcorr_adj standardized version of prebcorr_adj

Zpostacorr_adj standardized version of postacorr_adj

scale_prea score on Form A for students who took Form A as a pretest scaled to range 0-100

scale_preb score on Form B for students who took Form B as a pretest scaled to range 0-100

pre_scaled scale score on pretest (whichever form)

scale_posta score on Form A for students who took Form A as a posttest scaled to range 0-100

scale_postb score on Form B for students who took Form B as a posttest scaled to range 0-100

Flagged a logical variable (codebook lost)

Cond a factor describing the experimental condition with levels Adaptive/Accuracy, Adaptive/Extended, Linear/Extended and Control. Note that controls are included in this data set.

Sequencing a factor describing whether the sequence of items was Linear or Adaptive

Feedback a factor describing whether the feedback for incorrect items was Extended or AccuracyOnly

ACED.Qmatrix is a logical matrix whose rows correspond to skills (long names) and whose columns correspond to tasks which is true if the skill is required for solving the task (according to the expert).

ACED.QEM is a reduced Q-matrix containing the 15 *evidence models* (unique rows in the Q-matrix). The entries are character values with "0" indicating skill not needed, "+" indicating skill is

needed and "++" indicating skill is primary. The Tasks column lists the tasks corresponding to this evidence model (1, 2 and 3 again represent difficulty level, and the letters indicating variants). The Anchor column is used to identify subscales for scale identification.

ACED.splithalves is a list of two datasets labeled "A" and "B". Both have the same structure as ACED.scores (less the datas giving study condition). These were created by splitting the 62 items into 2 subforms with 31 items each. For the most part, each item was paired with an variant which differed only by the last letter. The scores are the results of Bayes net scoring with half of the items.

ACED.pretest and ACED.posttest are raw data from the external pretest and posttest given before and after the study. Each is a list with four components:

Araw Unscored responses for students who took that form as pre(post)test. The first row is the key.

Ascored The scored responses for the Araw students; correct is 1, incorrect is 0.

Braw Unscored responses for students who took that form as pre(post)test. The first row is the key.

Bscored The scored responses for the Araw students; correct is 1, incorrect is 0.

Because of the counterbalancing each student should appear in either Form A or Form B in the pretest and in the other group in the posttest. Note that the A and B forms here have no relationship with the A and B forms in ACED.splithalves.

Details

ACED is a Bayesian network based Assessment for Learning learning system, thus it served as both a assessment and a tutoring system. It had two novel features which could be turned on and off, elaborated feedback (turned off, it provided accuracy only feedback) and adaptive sequencing of items (turned off, it scheduled items in a fixed linear sequence).

It was originally built to cover all algebraic sequences (arithmetic, geometric and other recursive), but only the branch of the system using geometric sequences was tested. Shute, Hansen and Almond (2008) describe the field trial. Students from a local middle school (who studied arithmetic, but not geometric sequences as part of their algebra curriculum) were recruited for the study. The students were randomized into one of four groups:

Adaptive/Accuracy Adaptive sequencing was used, but students only received correct/incorrect feedback.

Adaptive/Extended Adaptive sequencing was used, but students received extended feedback for incorrect items.

Linear/Extended The fixed linear sequencing was used, but students received extended feedback for incorrect items.

Control The students did independent study and did not use ACED.

Because students in the control group were not exposed to the ACED task, neither the Bayes net level scores nor the item level scores are available for those groups, and those students are excluded from ACED.scores and ACED.items. The students are in the same order in all of the data sets, with the 60 control students tacked onto the end of the ACED.prePost data set.

All of the students (including the control students) were given a 25-item pretest and a 25-item posttest with items similar to the ones used in ACED. The design was counterbalanced, with half of the students receiving Form A as the pretest and Form B as the posttest and the other half the other

way around, to allow the two forms to be equated using the pretest data. The details are buried in `ACED.prePost`.

Note that some irregularities were observed with the English Language Learner (`ACED.prePost$Level_code=="ELL"`) students. Their teachers were allowed to translated words for the students, but in many cases actually wound up giving instruction as part of the translation.

Source

Shute, V. J., Hansen, E. G., & Almond, R. G. (2008). You can't fatten a hog by weighing it—Or can you? Evaluating an assessment for learning system called ACED. *International Journal of Artificial Intelligence and Education*, **18**(4), 289-316.

Thanks to Val Shute for permission to use the data.

ACED development and data collection was sponsored by National Science Foundation Grant No. 0313202.

References

A more detailed description, including a Q-matrix can be found at the ECD Wiki: http://ecd.ralmond.net/ecdwiki/ACED/ACED.

Examples

```
data(ACED)
```

`areaProbs` *Translates between normal and categorical probabilities*

Description

Maps between a continuous (normal) variable and a discrete variable by establishing a set of bins to maintain a particular probability vector. The `pvecToCutpoints` function returns the cut points separating the bins, the `pvecToMidpoints` returns a central point from each bin, and the `areaProbs` calculates the fraction of a normal curve falling into a particular bin.

Usage

```
pvecToCutpoints(pvec, mean = 0, std = 1)
pvecToMidpoints(pvec, mean = 0, std = 1)
areaProbs(pvec, condmean, condstd, mean = 0, std = 1)
```

Arguments

`pvec`	A vector of marginal probabilities for the categories of the discrete variable. Elements should be ordered from smallest to largest.
`mean`	The mean of the continuous variable.
`std`	The standard deviation of the continuous variable.
`condmean`	The conditional mean of the continuous variable.
`condstd`	The conditional standard deviation of the continuous variable.

Details

Let S be a discrete variable whose states s_k are given by `names(pvec)[k]` and for which the marginal probability $Pr(S = s_k) = p_k$ is given by `pvec[k]`. Let Y be a continuous normal variable with mean `mean` and standard deviation `std`. These function map between S and Y.

The function `pvecToCutpoints` produces a series of cutpoints, c_k, such that setting s_k to S when $c_k \leq Y \leq c_{k+1}$ produces the marginal probability specified by `pvec`. Note that c_1 is always `-Inf` and c_{K+1} is always `Inf` (where K is `length(pvec)`).

The function `pvecToMidpoints` produces the midpoints (with respect to the normal density) of the intervals defined by `pvecToCutpoints`. In particular, if $Pr(S \geq s_k) = P_k$, then the values returned are `qnorm`$(P_k + p_k/2)$.

The function `areaProbs` inverts these calculations. If `condmean` is $E[Y|x]$ and `condstd` is $\sqrt{var(Y|x)}$, then this function calculates $Pr(S|x)$ by calculating the area under the normal curve.

Value

For `pvecToCutpoints`, a vector of length one greater than `pvec` giving the endpoints of the bins. Note that the first and last values are always infinite.

For `pvecToCutpoints`, a vector of length the same length as `pvec` giving the midpoint of the bins.

For `areaProbs` a vector of probabilities of the same length as `pvec`.

Warning

Variables are given from *lowest* to *highest* state, for example 'Low', 'Medium', 'High'. StatShop expects variables in the opposite order.

Note

The function `effectiveThetas` does something similar, but assumes all probability values are equally weighted.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G. 'I Can Name that Bayesian Network in Two Matrixes.' *International Journal of Approximate Reasoning*, **51**, 167–178.

See Also

`effectiveThetas`

Examples

```
probs <- c(Low=.05,Med=.9,High=.05)
cuts <- pvecToCutpoints(probs)
mids <- pvecToMidpoints(probs)


areaProbs(probs,1,.5)
```

---

barchart.CPF *This function produces a plot of a conditional probability frame.*

Description

This is an extension of barchart.array for displaying conditional probability tables. In particular, it will display the output of calcDPCFrame.

Usage

```
## S3 method for class 'CPF'
barchart(x, data = NULL, ..., baseCol = "firebrick",
             auto.key=TRUE, par.settings)
```

Arguments

x	A conditional probaiblity frame (see as.CPF.
data	Ignore this value, used for compatability with barchart.
...	Other arguments passed on to barchart, in particular, other lattice graphics arguments.
baseCol	This should be a specification of a color. The color is designed as a gradient starting at the base color and getting progressively lighter. If its value is NULL, the colors are left at the default (pastel palette) and the value of par.settings is passed through unchanged.
auto.key	This is the auto.key parameter from barchart, it is overridden to have a default of true.
par.settings	This is the par.settings parameter from barchart. If baseCol is not null, then a value for superpose.polygon is added to set the bar colors.

Details

The function barchart.array and the function as.CPA to convert the conditional probability frame to an array do 90 percent of the work.

A few minor touches:

- The function takes special care of one row [un]conditional probability frames.

- The function overrides the default colors using colorspread to produce varying intensities of the same color.
- The function adds the dimension names, so that the labels indicate which variables they belong to.
- The function sets the default value of auto.key to TRUE so that a legend for the colors is produced.

Note that the color change is brought about internally by modifying par.settings. To suppress this behavior, set baseCol to null, and the user value for par.settings will be passed through unchanged.

Value

An object of class lattice that when printed will produce the graph.

Author(s)

Russell Almond

See Also

as.CPA, colorspread, barchart.array, calcDPCFrame

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

cpfTheta <- calcDPCFrame(list(),skill1l,numeric(),0,rule="Compensatory",
                         link="normalLink",linkScale=.5)

barchart.CPF(cpfTheta)

cptComp <- calcDPCFrame(list(S2=skill2l,S1=skill1l),correctL,
                        lnAlphas=log(c(1.2,.8)), betas=0,
                        rule="Compensatory")
barchart.CPF(cptComp,layout=c(3,1))

cptPC1 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                        lnAlphas=log(1),
                        betas=list(full=c(S1=0,S2=999),partial=c(S2=999,S2=0)),
                        rule="OffsetDisjunctive")
barchart.CPF(cptPC1,baseCol="slateblue")
```

betaci *Credibility intervals for a proportion based on beta distribution*

Description

This generates upper and lower bounds for a highest posterior density credibility interval for a beta distribution by looking at appropriate quantiles of the beta distribution. This is designed to work with sums of classification probabilities.

Usage

```
betaci(sumData, totals = NULL, limits = c(lower = 0.025, upper = 0.975),
       a = 0.5, b = 0.5)
```

Arguments

sumData	A vector or matrix of counts or sums proportions. Note these do not need to be integers, sums of classification probabilities work here.
totals	Total number of individuals as reference for sumData. If missing or NULL then the value use is rowSums(data).
limits	The upper and lower credibility limits.
a	Value for the shape1 parameter of the beta prior.
b	Value for the shape2 parameter of the beta prior.

Details

This function computes the upper and lower bounds of a credibility interval for a beta distribution based on sumData successes out of totals trials. Note that as a beta distribution is used for the basic calculations, neither sumData nor totals need be integers.

To avoid problems with zero cells (or cells with values equal to totals), a small prior is added to the beta calculations. By default a Jeffrey's prior $(.5, .5)$ is added to the data. Thus the final returned value is:

$$\texttt{qbeta}(prob, sumData + a, totals - sumData + b)$$

where prob varies over the values in limits. Note that a and b can be scalars or an array conformable with totals.

If totals is not supplied, and sumData is a vector, then the sum is used as the total. If the totals is not supplied and sumData is a matrix, then the row sums are used. For higher dimensional arrays, it is necessary to explicitly specify a total.

Value

A list of the same length as limits with the same names. Each component is a quantile of the posterior distribution which has the same shape as sumData.

Note that limits is not limited to length 2, although this is the most obvious application.

Note

A previous verison used `colSums(data)` rather than `rowSums()`. Using the row sums makes more sense as this corresponds to the CPF format.

Author(s)

Russell Almond

See Also

See `OCP` for an application.

Examples

```
x <- matrix(c(7,4,2,31),2,2)
## Use row sums as totals
betaci(x)

## fixed totals
x <- c(7,2,31)
nn <- c(30,15,35)
betaci(x,nn,c(Q1=.25,Q2=.5,Q3=.75))

## Prior varies according to cell.
pi0 <- c(.2,.2,.8)
betaci(x,nn,a=pi0,b=1-pi0)
```

---

buildFactorTab *Builds probability tables from Scored Bayes net output.*

Description

Looks for margin statistics in scored Bayes net output, and puts them into tables with rows representing variables and columns representing variable states.

The `marginTab` function does this for a single individual. The `buildMarginTab` uses the grand mean across all individuals and the `buildFactorTab` breaks down groups according to a factor variable. The function `build2FactorTab` builds a three-way table.

Usage

```
buildFactorTab(data, fact, stateNames, skillName, reverse = TRUE,
               stem="P", sep=".")
build2FactorTab(data, fact1, fact2, stateNames, skillName,
                reverse = TRUE, stem="P",sep=".")
buildMarginTab(data, stateNames, skillNames, reverse = TRUE,
               stem="P",sep=".")
marginTab(datarow, stateNames, skillNames, reverse = TRUE,
          stem="P",sep=".")
```

Arguments

data	A data sets of StatShop statistics for many individuals.
datarow	One row of such a data set.
fact	A factor variable according to which to split the data. Length should be the same as the length of data.
fact1	A factor variable according to which to split the data.
fact2	A factor variable according to which to split the data.
stateNames	Names of the variable states.
skillName, skillNames	Name(s) of the proficiency variable(s) to be used.
reverse	Reverse the order of the states for display (i.e., convert from StatShop order of highest first to more natural order of lowest first.
stem	A character string giving a prefix used to indicate variable names.
sep	A character string giving a separator used to separate prefix from variable names.

Details

This looks for columns marked "<stem><sep><skillName>" in the data frame, and builds them into a matrix. It is assumed that all variables have the same number of states and that they are in the same order, and the order is the same as given in stateNames.

The functions buildFactorTab and build2FactorTab really expect their skillNames argument to be a single variable name. However, they should work with multiple variables if suitable values are chosen for the state names.

Value

For marginTab a matrix with columns corresponding to skillNames and rows corresponding to stateNames giving the probabilities for a single individual.

For buildMarginTab a matrix with columns corresponding to skillNames and rows corresponding to stateNames giving the average probabilities for the entire data set.

For buildFactorTab a matrix with columns corresponding to the unique values of fact and rows corresponding to stateNames entries give the average probabilities across the groups.

For build2FactorTab a 3 dimensional array with the first dimension corresponding to the unique values of fact1, the second dimension corresponding to the unique values of fact2 and the last dimension corresponding to stateNames entries give the average probabilities across the groups.

Author(s)

Russell Almond

See Also

stackedBars,compareBars

Examples

```
data(ACED)

marginTab(ACED.scores[1,], c("H","M","L"), ACED.skillNames$short, reverse = TRUE,
          stem="P",sep=".")

buildMarginTab(ACED.scores, c("H","M","L"), ACED.skillNames$short[1:4],
               reverse = TRUE,
               stem="P",sep=".")

buildFactorTab(ACED.scores, ACED.scores$Cond_code, c("H","M","L"), "sgp",
               reverse = TRUE,
               stem="P", sep=".")

build2FactorTab(ACED.scores, ACED.scores$Sequencing, ACED.scores$Feedback,
                c("H","M","L"), "sgp",
                reverse = TRUE, stem="P",sep=".")
```

`buildParentList` *Builds a list of parents of nodes in a graph*

Description

Takes an incidence matrix describing a graph, and an order of the nodes, and builds a list of parents for each node. If the `ord` argument is not supplied, the ordering is constructed through maximum cardinality search (see `mcSearch`).

Usage

```
buildParentList(sm, start = colnames(sm)[1], ord = mcSearch(sm, start))
```

Arguments

`sm`	A logical matrix whose rows and columns correspond to nodes (variables) and a true value indicates an edge between the variables.
`start`	The name of the first element.
`ord`	A vector of size equal to the number of columns of `sm` giving the ordering of nodes.

Details

The `sm` argument should be an incidence matrix for a graph, with row and column names set to the names of the nodes/variables.

A node i is a *parent* of node j if

- aThey are neighbors, that is `sm[i,j] == TRUE`.

- bThe node i appears before node j in `ord`, that is `ord[i]` < `ord[j]`.

The argument `start` is used only if `ord` is not supplied, in which case it is passed to `mcSearch`.

Value

A list with as many elements as there are columns in `sm`, and whose elements appear in the order specified by `ord`. Each element of that list is a character vector giving the names of the parents.

Author(s)

Russell Almond

References

Almond, R. G. (2010). 'I can name that Bayesian network in two matrixes.' *International Journal of Approximate Reasoning*. **51**, 167-178.

See Also

`mcSearch`, `structMatrix`

Examples

```
data(MathGrades)
MG.struct <- structMatrix(MathGrades$var)

parents <- buildParentList(MG.struct)   # Arbitrary start
parentsa <- buildParentList(MG.struct, "Algebra") # Put algebra first.
```

---

buildRegressions *Creates a series of regressions from a covariance matrix*

Description

This function takes a covariance matrix and list of variables and their parents and returns a collection of regression model parameters for each variable regressed on its parents. This is a compact representation of a normal graphical model.

Usage

```
buildRegressions(Sigma, means = 0,
                 parents = buildParentList(structMatrix(Sigma)))
```

Arguments

Sigma	A covariance matrix among a collection of continuous variables.
means	The means of those variables
parents	A named list of length equal to ncol(Sigma) whose elements should correspond to the rows/columns of Sigma. Each element should be a character vector giving the names of the parents of the given node.

Details

This function performs one regression for each element of the parents list. The name of the dependent variable for each regression is given by names(parents) and the independent variables is given by the values of parents. (The function buildParentList() builds a suitable list of elements.)

If means is not supplied, then the variables are assumed to be centered, otherwise the given vector is used as the means.

Value

A list of length equal to parents whose elements are also a list having the following structure

b	A vector of slopes for the regression with names equal to the names of the parent variables. Note that if there are no parents, this will be numeric(0).
a	The intercept from the regression.
std	The residual standard deviation from the regression.

Author(s)

Russell Almond

References

Almond, R. G. (2010). 'I can name that Bayesian network in two matrixes.' *International Journal of Approximate Reasoning.* **51**, 167-178.

See Also

buildParentList, buildRegressionTables

Examples

```
data(MathGrades)
pl <- buildParentList(structMatrix(MathGrades$var),"Algebra")
rt <- buildRegressions(MathGrades$var,MathGrades$mean,pl)
```

buildRegressionTables *Builds conditional probability tables from regressions*

Description

Takes a description of a normal graphical model as a series of regressions and a description of the corresponding discrete variables and builds a series of conditional probability tables for the corresponding Bayesian network.

Usage

```
buildRegressionTables(regs, pvecs, mean = 0, std = 1)
```

Arguments

regs	A list with names corresponding to the variables in the model giving a series of regression coefficients (see Details).
pvecs	A list with names corresponding to the variables in the model giving a series of probability vectors in order from highest state to lowest state (see Details).
mean	A vector of means for the continuous variables.
std	A vector of standard deviations for the continuous variables.

Details

The regs argument should be a list whose names are the names of the variables. Each element should have the following fields:

b A vector of slopes for the regression with names equal to the names of the parent variables. Note that if there are no parents, this should be numeric(0).

a The intercept from the regression.

std The residual standard deviation from the regression.

The function buildRegressions() creates an appropriate list.

The pvecs should be a list whose names are the names of the variables. Each element should be a named vector of probabilities in order from the *Highest* to the *Lowest* state, e.g. c(High=.2,Med=.5,Low=.3).

The values mean and std should either be scalars or vectors of the same length as the number of elements in regs and pvecs. If vectors, they should have names corresponding to the variable names. Note that the order of the elements does not need to be the same in all four arguments, but that the names of all four arguments need to be identical (unless mean or std are given as scalars, in which case they will be appropriately replicated.)

Value

A list of conditional probability tables whose names correspond to regs. Each conditional probability table is expressed as a data frame whose columns correspond to either parent variables or states of the child variable and whose rows correspond to configurations of the parent variable.

Warning

Variables are given from *highest* to *lowest* state, for example 'High', 'Medium', 'Low'. This is the order expected by StatShop. Note that `pvecToCutpoints` expects probability vectors in the opposite order.

Author(s)

Russell Almond

References

Almond, R. G. (2010). 'I can name that Bayesian network in two matrixes.' *International Journal of Approximate Reasoning*. **51**, 167-178.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics*. Wiley.

See Also

`buildRegressions`

Examples

```
data(MathGrades)
pl <- buildParentList(structMatrix(MathGrades$var),"Algebra")
rt <- buildRegressions(MathGrades$var,MathGrades$means,pl)
tabs <- buildRegressionTables(rt, MathGrades$pvecs, MathGrades$means,
                              sqrt(diag(MathGrades$var)))
```

---

calcDDTable *Calculates DiBello–Dirichlet model probability and parameter tables*

Description

The DiBello–Dirichlet model creates a hyper-Dirichlet prior distribution by interpolating between an `masterProfile` and a `noviceProfile`. This function builds the hyper-Dirichlet parameter table, or with normalization, the conditional probability table for this distribution type.

Usage

```
calcDDTable(skillLevels, obsLevels, skillWeights, masterProfile,
            noviceProfile = 0.5, rule = "Compensatory")
calcDDFrame(skillLevels, obsLevels, skillWeights, masterProfile,
            noviceProfile = 0.5, rule = "Compensatory")
```

Arguments

skillLevels	A list of character vectors giving names of levels for each of the condition variables.
obsLevels	A character vector giving names of levels for the output variables from highest to lowest. As a special case, can also be a vector of integers.
skillWeights	A numeric vector of the same length as skillLevels giving the weight to be applied to each skill.
masterProfile	The Dirichlet prior for "experts" (see Details). Its length should match obsLevels.
noviceProfile	The Dirichlet prior for "novices" (see Details). Its length should match obsLevels or as a special case a scalar quantity gives a uniform prior. Default is uniform prior with weight 1/2.
rule	Function for computing effective theta (see Details).

Details

Assume for the moment that there are two possible skill profiles: "expert" and "novice". This model presumes a conditional probability table for the outcome given skill profile with two rows each of which is an independent categorical distribution. The natural conjugate prior is an independent Dirichlet distribution for each row. The parameters for this distribution are given in the masterProfile and noviceProfile arguments.

If there is more than one parent variable or if the parent variable has more than one state, the situation becomes muddier. The "expert" state is obviously the one with all the variables at the highest levels and the "novice" is the one with all variables at the lowest levels. If we can assign an integer between 0 and 1 to each of the intermediate states, then we can interpolate between them to produce Dirichlet priors for each row.

This distribution type uses the DiBello effective theta technique to come up with the interpolation. Each parent variable state is assigned a 'theta' value using the effectiveThetas function to assign a numeric value to each one. These are then combined using the function rule in the rule argument. The resulting theta values are then scaled to a range of 0–1. The prior for that row is a weighted combination of the masterProfile and noviceProfile.

The combination of the individual effective theta values into a joint value for effective theta is done by the function reference by rule. This should be a function of three arguments: theta — the vector of effective theta values for each parent, alphas — the vector of discrimination parameters, and beta — a scalar value giving the difficulty. The initial distribution supplies three functions appropriate for use with calcDSTable: Compensatory, Conjunctive, and Disjunctive. Note that the beta argument is effectively ignored because of the later scaling of the output.

Normally obslevel should be a character vector giving state names. However, in the special case of state names which are integer values, R will "helpfully" convert these to legal variable names by prepending a letter. This causes other functions which rely on the names() of the result being the state names to break. As a special case, if the value of obsLevel is of type numeric, then calcDSFrame() will make sure that the correct values are preserved.

Value

For calcDDTable, a matrix whose rows correspond configurations of the parent variable states (skillLevels) and whose columns correspond to obsLevels. Each row of the table is the parameters of a Dirichlet distribution, so the whole matrix is the parameters for a hyper-Dirichlet

distribution. The order of the parent rows is the same as is produced by applying `expand.grid` to `skillLevels`.

For `calcDDFrame` a data frame with additional columns corresponding to the entries in `skillLevels` giving the parent value for each row.

Note

Unlike `calcDSTable`, there is not a corresponding DiBello-Dirichlet distribution support in StatShop. This function is used to model the parameters of an unconstrained hyper-Dirichlet distribution.

This was originally designed for use in Situational Judgment Tests where experts might not agree on the "key".

Note: Zeros in the `masterProfile` indicate responses that a master would **never** make. They will result in zero probability of mastery for any response which yields that outcome.

References

Almond, R.G. and Roberts, R. (Draft) Bayesian Scoring for Situational Judgment Tests. Unpublished white paper.

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

See Also

`effectiveThetas`,`Compensatory`, `calcDNTable`, `calcDSTable`, `expand.grid`

Examples

```
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Low")
option5L <- c("A","B","C","D","E")

## Expert responses
eProfile <- c(A=7,B=15,C=3,D=0,E=0)

paramT <- calcDDTable(list(S1=skill1l,S2=skill2l), option5L,
                      c(S1=2,S2=1), masterProfile=eProfile+0.5)

paramF <- calcDDFrame(list(S1=skill1l,S2=skill2l), option5L,
                      c(S1=2,S2=1), masterProfile=5*eProfile+0.5,
                      noviceProfile=2)
```

`calcDNTable` *Creates the probability table for DiBello–Normal distribution*

Description

The `calcDNTable` function takes a description of input and output variables for a Bayesian network distribution and a collection of IRT-like parameter (discrimination, difficulty) and calculates a conditional probability table using the DiBello–Normal distribution (see Details). The `calcDNFrame` function returns the value as a data frame with labels for the parent states.

Usage

```
calcDNTable(skillLevels, obsLevels, lnAlphas, beta, std, rule = "Compensatory")
calcDNFrame(skillLevels, obsLevels, lnAlphas, beta, std, rule = "Compensatory")
```

Arguments

`skillLevels`	A list of character vectors giving names of levels for each of the condition variables.
`obsLevels`	A character vector giving names of levels for the output variables from highest to lowest. Can also be a vector of integers (see Details).
`lnAlphas`	A vector of log slope parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`beta`	A vector of difficulty (-intercept) parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`std`	The log of the residual standard deviation (see Details).
`rule`	Function for computing effective theta (see Details).

Details

The DiBello–Normal distribution (Almond et al, 2015) is a variant of the DiBello–Samejima distribution (Almond et al, 2001) for creating conditional probability tables for Bayesian networks which uses a regression-like (probit) link function in place of Samejima's graded response link function. The basic procedure unfolds in three steps.

1. Each level of each input variable is assigned an "effective theta" value — a normal value to be used in calculations.
2. For each possible skill profile (combination of states of the parent variables) the effective thetas are combined using a combination function. This produces an "effective theta" for that skill profile.
3. Taking the effective theta value as the mean, the probability that the examinee will fall into each category.

The parent (conditioning) variables are described by the skillLevels argument which should provide for each parent variable in order the names of the states ranked from highest to lowest value. These are calculated through the function effectiveThetas which gives equally spaced points on the probability curve. Note that for the DiBello-Normal distribution, Step 1 and Step 3 are inverses of each other (except for rounding error).

The combination of the individual effective theta values into a joint value for effective theta is done by the function reference by rule. This should be a function of three arguments: theta — the vector of effective theta values for each parent, alphas — the vector of discrimination parameters, and beta — a scalar value giving the difficulty. The initial distribution supplies five functions appropriate for use with calcDSTable: Compensatory, Conjunctive, and Disjunctive, OffsetConjunctive, and OffsetDisjunctive. The last two have a slightly different parameterization: alpha is assumed to be a scalar and betas parameter is vector valued. Note that the discrimination and difficulty parameters are built into the structure function and not the probit curve.

The effective theta values are converted to probabilities by assuming that the categories for the consequence variable (obsLevels) are generated by taking equal probability divisions of a standard normal random variable. However, a person with a given pattern of condition variables is drawn from a population with mean at effective theta and standard deviation of exp(std). The returned numbers are the probabilities of being in each category.

Normally obslevel should be a character vector giving state names. However, in the special case of state names which are integer values, R will "helpfully" convert these to legal variable names by prepending a letter. This causes other functions which rely on the names() of the result being the state names to break. As a special case, if the value of obsLevel is of type numeric, then calcDNFrame() will make sure that the correct values are preserved.

Value

For calcDNTable, a matrix whose rows correspond configurations of the parent variable states (skillLevels) and whose columns correspond to obsLevels. Each row of the table is a probability distribution, so the whole matrix is a conditional probability table. The order of the parent rows is the same as is produced by applying expand.grid to skillLevels.

For calcDNFrame a data frame with additional columns corresponding to the entries in skillLevels giving the parent value for each row.

Note

This distribution class was developed primarily for modeling relationships among proficiency variables. For models for observables, see calcDSTable.

This function has largely been superceeded by calls to calcDPCTable with normalLink as the link function.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

See Also

effectiveThetas,Compensatory, OffsetConjunctive,eThetaFrame, calcDSTable, calcDNllike, calcDPCTable, expand.grid

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
skill3l <- c("Advanced","Proficient","Basic","Developing")

cptSkill3 <- calcDNTable(list(S1=skill1l,S2=skill2l),skill3l,
                         log(c(S1=1,S2=.75)),1.0,log(0.5),
                         rule="Compensatory")

cpfSkill3 <- calcDNFrame(list(S1=skill1l,S2=skill2l),skill3l,
                         log(c(S1=1,S2=.75)),1.0,log(0.5),
                         rule="Compensatory")
```

---

calcDPCTable *Creates the probability table for the discrete partial credit model*

Description

The calcDPCTable function takes a description of input and output variables for a Bayesian network distribution and a collection of IRT-like parameter (discrimination, difficulty) and calculates a conditional probability table using the discrete partial credit distribution (see Details). The calcDPCFrame function returns the value as a data frame with labels for the parent states.

Usage

```
calcDPCTable(skillLevels, obsLevels, lnAlphas, betas,
             rules = "Compensatory", link="partialCredit",
             linkScale=NULL, Q=TRUE,
             tvals=lapply(skillLevels,
               function (sl) effectiveThetas(length(sl))))

calcDPCFrame(skillLevels, obsLevels, lnAlphas, betas,
             rules = "Compensatory", link="partialCredit",
             linkScale=NULL, Q=TRUE,
             tvals=lapply(skillLevels,
               function (sl) effectiveThetas(length(sl))))
```

Arguments

skillLevels	A list of character vectors giving names of levels for each of the condition variables.
obsLevels	A character vector giving names of levels for the output variables from highest to lowest. As a special case, can also be a vector of integers.
lnAlphas	A list of vectors of log slope parameters. Its length should be 1 or length(obsLevels)-1. The required length of the individual component vectors depends on the choice of rule (and is usually either 1 or the length of skillLevels).
betas	A list of vectors of difficulty (-intercept) parameters. Its length should be 1 or length(obsLevels)-1. The required length of the individual component vectors depends on the choice of rule (and is usually either 1 or the length of skillLevels).
rules	A list of functions for computing effective theta (see Details). Its length should be length(obsLevels)-1 or 1 (implying that the same rule is applied for every gap.)
link	The function that converts a table of effective thetas to probabilities
linkScale	An optional scale parameter for the link function. This is only used with certain choices of link function.
Q	This should be a Q matrix indicating which parent variables are relevant for which state transitions. It should be a number of states minus one by number of parents logical matrix. As a special case, if all variable are used for all levels, then it can be a scalar value.
tvals	A list of the same length as skillLevels. Each element should be a numeric vector values on the theta (logistic) scale corresponding to the levels for that parent variable. The default spaces them equally according to the normal distribution (see effectiveThetas).

Details

The discrete graded response model is a generalization of the DiBello–Samejima mechanism for creating conditional probability tables for Bayesian network models using IRT-like parameters (calcDSTable). The basic procedure unfolds in three steps.

1. Each level of each input variable is assigned an "effective theta" value — a normal value to be used in calculations.
2. For each possible skill profile (combination of states of the parent variables) the effective thetas are combined using a one of the rule functions. This produces an "effective theta" for that skill profile.
3. The effective theta table is input into the link function to produce a probability distribution over the states of the outcome variables.

The parent (conditioning) variables are described by the skillLevels argument which should provide for each parent variable in order the names of the states ranked from highest to lowest value. The default implementation uses the function effectiveThetas to calculate equally spaced points on the normal curve. This can be overridden by supplying a tvals argument. This should be a list

of the same length as skillLevels with each element having the same length as the corresponding element of skillLevels.

The tvals (either default or user supplied) are used to create a table of rows with values $\theta_1, \ldots, \theta_K$, corresponding to all possible combinations of the parent variables (using expand.grid).

Let X be the child variable of the distribution, and assume that it can take on M possible states labeled x_1 through x_M in increasing order. (Note: that calcDPCTable assumes variable states are ordered the other direction: from highest to lowest.) For each state but the lowest state (the last one in the input order) defines a combination rule $Z_m(\theta_1, \ldots, \theta_K; alphas, betas)$. Applying these functions to the rows of the table produces a table of effective thetas for each configuration of the parent variables and each child state except for the lowest. (The metaphor is this theta represents the "ability level" required to reach that output state.)

Note that the $Z_m(\cdot)$s do not need to have the same parameters or even the same functional form. The argument rules should contain a list of the names of the combination functions, the first one corresponding to $Z_M(\cdot)$, and so forth in descending order. As a special case, if rules has only one element, than it is used for all of the transitions. Similarly, the lnAlphas and betas should also be lists of the parameters of the combination functions corresponding to the transitions between the levels. The betas[[m]] represent difficulties (negative intercepts) and the exp(lnAlphas[[m]]) represent slopes for the transition to level m (following the highest to lowest order). Again if these lists have length one, the same value is used for all transitions.

The length of the elements of lnAlphas and betas is determined by the specific choice of combination function. The functions Compensatory, Conjunctive, and Disjunctive all assume that there will be one lnAlpha for each parent variable, but a single beta. The functions OffsetConjunctive, and OffsetDisjunctive both assume that there will be one beta for each parent variable, but a single lnAlpha.

The code link function is then applied to the table of effective theta values to produce a conditional probability distribution. Two link functions are currently supported: partialCredit is based on the generalized partial credit model (Muraki, 1992), gradedResponse is a modified version of the graded response model (Samejima, 1969). (The modification corrects for problems when the curves cross.) A third planned link function is based on a normal error model, this will require the extra linkScale parameter.

The Q matrix is used in situations where some of the parent variables are not relevant for one or more parent transitions. If parent k is relevant for the transition between state m+1 and m (remember that states are coded from highest to lowest) then Q[m,k] should be TRUE. In particular, eTheta[,Q[m,]] is passed to the combination rule, not all of theta. If there are false entries in Q the corresponding sets of alphas and betas need to have the correct length. Generally speaking, Q matrixes with FALSE entries are not appropriate with the gradedResponse link. As a special case if Q=TRUE, then all parent variables are used for all state transitions.

Normally obslevel should be a character vector giving state names. However, in the special case of state names which are integer values, R will "helpfully" convert these to legal variable names by prepending a letter. This causes other functions which rely on the names() of the result being the state names to break. As a special case, if the value of obsLevel is of type numeric, then calcDSFrame() will make sure that the correct values are preserved.

Value

For calcDPCTable, a matrix whose rows correspond configurations of the parent variable states (skillLevels) and whose columns correspond to obsLevels. Each row of the table is a probability

distribution, so the whole matrix is a conditional probability table. The order of the parent rows is the same as is produced by applying `expand.grid` to `skillLevels`.

For `calcDPCFrame` a data frame with additional columns corresponding to the entries in `skillLevels` giving the parent value for each row.

Note

The framework set up by this function is completely expandable. The `link` and the elements of `rules` can be any value that is suitable for the first argument of `do.call`.

Elements of `rules` are called with the expression `do.call(rules[[kk]],list(thetas,exp(lnAlphas[[kk]]),betas[[k` where `thetas` is the matrix of effective theta values produced in the first step of the algorithm, and the return function should be a vector of effective thetas, one for each row of `thetas`.

The `link` function is called with the expression `do.call(link,list(et,linkScale,obsLevels))` where `et` is the matrix of effective thetas produced in the second step. It should return a conditional probability table with the same number of rows and one more column than `et`. All of the rows should sum to 1.0.

Author(s)

Russell Almond

References

Almond, R.G. (2015). An IRT-based Parameterization for Conditional Probability Tables. Paper submitted to the 2015 Bayesian Application Workshop at the Uncertainty in Artificial Intelligence conference.

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Muraki, E. (1992). A Generalized Partial Credit Model: Application of an EM Algorithm. *Applied Psychological Measurement*, **16**, 159-176. DOI: 10.1177/014662169201600206

Samejima, F. (1969) Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph No. 17*, **34**, (No. 4, Part 2).

See Also

`effectiveThetas`,`Compensatory`, `OffsetConjunctive`,`eThetaFrame`, `calcDNTable`, `calcDSTable`, `expand.grid`, `gradedResponse`, `partialCredit`

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

## Simple binary model, these three should be the same.
```

```
cptCorrect <- calcDPCTable(list(S1=skill1l,S2=skill2l),correctL,
                           log(c(S1=1,S2=.75)),1.0,rule="Compensatory",
                           link="partialCredit")
cptCorrect2 <- calcDPCTable(list(S1=skill1l,S2=skill2l),correctL,
                           log(c(S1=1,S2=.75)),1.0,rule="Compensatory",
                           link="gradedResponse")
cptCorrect1 <- calcDSTable(list(S1=skill1l,S2=skill2l),correctL,
                           log(c(S1=1,S2=.75)),1.0,rule="Compensatory")
stopifnot (all (abs(cptCorrect2-cptCorrect1) <.001))
stopifnot (all (abs(cptCorrect-cptCorrect1) <.001))

## Conjunctive uses multiple betas, not multiple alphas.
cptConj <- calcDPCTable(list(S1=skill1l,S2=skill2l),correctL,
                        log(1),c(S1=0.5,S2=.7),rule="OffsetConjunctive")

## Test for no parent case
cptTheta <- calcDPCTable(list(),skill1l,numeric(),0,rule="Compensatory",
                         link="normalLink",linkScale=.5)
cpfTheta <- calcDPCFrame(list(),skill1l,numeric(),0,rule="Compensatory",
                         link="normalLink",linkScale=.5)


## Simple model, Skill 1 needed for step 1, Skill 2 for Step 2.
cptPC1 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                       lnAlphas=log(1),
                       betas=list(full=c(S1=0,S2=999),partial=c(S2=999,S2=0)),
                       rule="OffsetDisjunctive")
##Variant using Q-matrix
cptPC1a <- calcDPCTable(list(S1=skill1l,S2=skill2l),pcreditL,
                       lnAlphas=log(1),
                       betas=list(full=c(S1=0),partial=c(S2=0)),
                       Q=matrix(c(TRUE,FALSE,FALSE,TRUE),2,2),
                       rule="OffsetDisjunctive")
stopifnot(all(abs(as.vector(numericPart(cptPC1))-as.vector(cptPC1a))<.0001))


## Complex model, different rules for different levels
cptPC2 <- calcDPCTable(list(S1=skill1l,S2=skill2l),pcreditL,
                         list(full=log(1),partial=log(c(S1=1,S2=.75))),
                         betas=list(full=c(0,999),partial=1.0),
                         rule=list("OffsetDisjunctive","Compensatory"))

## Graded Response Model, typically uses different difficulties
cptGraded <- calcDPCTable(list(S1=skill1l),gradeL,
                         log(1),betas=list(A=2,B=1,C=0,D=-1),
                         rule="Compensatory",link="gradedResponse")

## Partial credit link is somewhat different
cptPC5 <- calcDPCTable(list(S1=skill1l),gradeL,
                         log(1),betas=list(A=2,B=1,C=0,D=-1),
                         rule="Compensatory",link="partialCredit")
cptPC5a <- calcDPCTable(list(S1=skill1l),gradeL,
                         log(1),betas=1,
```

```
                          rule="Compensatory",link="partialCredit")

## Need to be careful when using different slopes (or non-increasing
## difficulties) with graded response link as curves may cross.

cptCross <- calcDPCTable(list(S1=skill1l),pcreditL,
                          log(1),betas=list(full=-1,partial=1),
                          rule="Compensatory",link="gradedResponse")
stopifnot (all(abs(cptCross[,"Partial"])<.001))
```

`calcDSllike` *Calculates the log-likelihood for data from a DiBello–Samejima (Normal) distribution*

Description

These functions take `data` which represent draws from a categorical data with the given DiBello–Samejima distribution and returns the log-likelihood of the given data.

Usage

```
calcDSllike(data, parents, skillLevels, child, obsLevels,
            lnAlphas, beta, dinc = 0, rule = "Compensatory")
calcDNllike(data, parents, skillLevels, child, obsLevels,
            lnAlphas, beta, std, rule = "Compensatory")
```

Arguments

`data`	A data frame whose columns contain variables corresponding to `parent` and `child`.
`parents`	A vector of names for the columns in `data` corresponding to the parent variables.
`child`	The name of the child variable, should refer to a column in `data`.
`skillLevels`	A list of character vectors giving names of levels for each of the condition variables.
`obsLevels`	A character vector giving names of levels for the output variables from highest to lowest.
`lnAlphas`	A vector of log slope parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`beta`	A vector of difficulty (-intercept) parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`dinc`	Vector of difficulty increment parameters (see `calcDSTable`).
`rule`	Function for computing effective theta (see `calcDSTable`).
`std`	The log of the residual standard deviation (see Details).

Details

This function assumes that the observed data are independent draws from a Bayesian network. This function calculates the log-likelihood of a single conditional probability table. First, it calculates a table of counts corresponding states of the parent and child variables using the function dataTable. Next it calculates the conditional probability for each cell using the function calcDSTable or calcDNTable.

It then calculates the log-likelihood as the sum of $count(cell) * log(Pr(cell))$ where this value is set to zero if $count(cell)$ is zero (this allows cells with zero probability as long as the count is also zero).

Value

A real giving the log-likelihood of the observed data.

Note

This function is primarily about testing the log likelihood calculations used internally in StatShop.

This function is largely superceeded by the likelihood calculation internal to mapDPC. In particular, if probs is the result of the call to calcDPCTable, and postTable is the expected contingency table (e.g., the output of expTable). Then the log likelihood is `-2*sum(as.vector(postTable)*as.vector(log(probs)))`.

Author(s)

Russell Almond

References

http://comet.research.ets.org/~ralmond/StatShop

See Also

dataTable, calcDSTable, Compensatory,OffsetConjunctive, eThetaFrame, calcDNTable

Examples

```
skill1l <- c("High","Medium","Low")
skill3l <- c("High","Better","Medium","Worse","Low")
correctL <- c("Correct","Incorrect")

x <- read.csv(system.file("testFiles", "randomPinned100.csv",
              package="CPTtools"),
            header=TRUE, as.is=TRUE)
x[,"Skill1"] <- ordered(x[,"Skill1"],skill1l)
x[,"Skill3"] <- ordered(x[,"Skill3"],skill3l)
x[,"Comp.Correct"] <- ordered(x[,"Comp.Correct"],correctL)


like <- calcDSllike(x,c("Skill1","Skill3"),
                    list(Skill1=skill1l, Skill3=skill3l),
                    "Comp.Correct", correctL,
```

```
log(c(0.45,-0.4)),-1.9,rule="Compensatory")
```

`calcDSTable` *Creates the probability table for DiBello–Samejima distribution*

Description

The `calcDSTable` function takes a description of input and output variables for a Bayesian network distribution and a collection of IRT-like parameter (discrimination, difficulty) and calculates a conditional probability table using the DiBello-Samejima distribution (see Details). The `calcDSFrame` function returns the value as a data frame with labels for the parent states.

Usage

```
calcDSTable(skillLevels, obsLevels, lnAlphas, beta, dinc = 0,
            rule = "Compensatory")
calcDSFrame(skillLevels, obsLevels, lnAlphas, beta, dinc = 0,
            rule = "Compensatory")
```

Arguments

`skillLevels`	A list of character vectors giving names of levels for each of the condition variables.
`obsLevels`	A character vector giving names of levels for the output variables from highest to lowest. As a special case, can also be a vector of integers.
`lnAlphas`	A vector of log slope parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`beta`	A vector of difficulty (-intercept) parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`dinc`	Vector of difficulty increment parameters (see Details).
`rule`	Function for computing effective theta (see Details).

Details

The DiBello–Samejima model is a mechanism for creating conditional probability tables for Bayesian network models using IRT-like parameters. The basic procedure unfolds in three steps.

1. Each level of each input variable is assigned an "effective theta" value — a normal value to be used in calculations.
2. For each possible skill profile (combination of states of the parent variables) the effective thetas are combined using a combination function. This produces an "effective theta" for that skill profile.
3. The effective theta is input into Samejima's graded-response model to produce a probability distribution over the states of the outcome variables.

The parent (conditioning) variables are described by the skillLevels argument which should provide for each parent variable in order the names of the states ranked from highest to lowest value. The original method (Almond et al., 2001) used equally spaced points on the interval $[-1, 1]$ for the effective thetas of the parent variables. The current implementation uses the function effectiveThetas to calculate equally spaced points on the normal curve.

The combination of the individual effective theta values into a joint value for effective theta is done by the function reference by rule. This should be a function of three arguments: theta — the vector of effective theta values for each parent, alphas — the vector of discrimination parameters, and beta — a scalar value giving the difficulty. The initial distribution supplies five functions appropriate for use with calcDSTable: Compensatory, Conjunctive, and Disjunctive, OffsetConjunctive, and OffsetDisjunctive. The last two have a slightly different parameterization: alpha is assumed to be a scalar and betas parameter is vector valued. Note that the discrimination and difficulty parameters are built into the structure function and not the IRT curve.

The Samejima graded response link function describes a series of curves:

$$P_m(\theta) = Pr(X >= x - m|\theta) = logit^{-1}(\theta - d_m)$$

for $m > 1$, where $D = 1.7$ (a scale factor to make the logistic curve match more closely with the probit curve). The probability for any given category is then the difference between two adjacent logistic curves. Note that because a difficulty parameter was included in the structure function, we have the further constraint that $\sum d_m = 0$.

To remove the parameter restriction we work with the difference between the parameters: $d_m - d_{m-1}$. The value of d_2 is set at -sum(dinc)/2 to center the d values. Thus the dinc parameter (which is required only if length(obsLevels)>2) should be of length length(obsLevels)-2. The first value is the difference between the d values for the two highest states, and so forth.

Normally obslevel should be a character vector giving state names. However, in the special case of state names which are integer values, R will "helpfully" convert these to legal variable names by prepending a letter. This causes other functions which rely on the names() of the result being the state names to break. As a special case, if the value of obsLevel is of type numeric, then calcDSFrame() will make sure that the correct values are preserved.

Value

For calcDSTable, a matrix whose rows correspond configurations of the parent variable states (skillLevels) and whose columns correspond to obsLevels. Each row of the table is a probability distribution, so the whole matrix is a conditional probability table. The order of the parent rows is the same as is produced by applying expand.grid to skillLevels.

For calcDSFrame a data frame with additional columns corresponding to the entries in skillLevels giving the parent value for each row.

Note

This distribution class is not suitable for modeling relationship among proficiency variable, primarily because the normal mapping used in the effective theta calculation and the Samejima graded response models are not inverses. For those model, the function calcDNTable, which uses a probit link function, is recommended instead.

This function has largely been superceeded by calls to calcDPCTable with gradedResponse as the link function.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

Samejima, F. (1969) Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph No. 17*, **34**, (No. 4, Part 2).

See Also

effectiveThetas,Compensatory, OffsetConjunctive,eThetaFrame, calcDNTable, calcDSllike, calcDPCTable, expand.grid

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
correctL <- c("Correct","Incorrect")
gradeL <- c("A","B","C","D","E")

cptCorrect <- calcDSTable(list(S1=skill1l,S2=skill2l),correctL,
                          log(c(S1=1,S2=.75)),1.0,rule="Conjunctive")

cpfCorrect <- calcDSFrame(list(S1=skill1l,S2=skill2l),correctL,
                          log(c(S1=1,S2=.75)),1.0,rule="Conjunctive")


cptGraded <- calcDSTable(list(S1=skill1l),gradeL, 0.0, 0.0, dinc=c(.3,.4,.3))
```

`calcNoisyAndTable` *Calculate the conditional probability table for a Noisy-And or Noisy-Min distribution*

Description

Calculates the conditional probability table for a noisy-and distribution. This follows a logical model where all inputs must be true for the output to be true; however, some "noise" is allowed that produces random deviations from the pure logic. The noisy-min is a generalization in which all variables are ordered, and the weakest of the parent variables drives the conditional probabilities of the child variable.

Usage

```
calcNoisyAndTable(skillLevels, obsLevels = c("True", "False"),
                  bypass = rep(0, length(skillLevels)), noSlip=1,
                  thresholds = sapply(skillLevels, function(states) states[1]))
calcNoisyAndFrame(skillLevels, obsLevels = c("True", "False"),
                  bypass = rep(0, length(skillLevels)), noSlip=1,
                  thresholds = sapply(skillLevels, function(states) states[1]))
```

Arguments

skillLevels	A list of character vectors giving names of levels for each of the condition variables.
obsLevels	A character vector giving names of levels for the output variables from highest to lowest. As a special case, can also be a vector of integers. Its length should be 2, and the first value is considered to be logically equivalent to "true".
noSlip	A scalar value between 0 and 1. This represents the probability that the output will be true when all of the inputs are true (e.g., 1 - the probability that an examinee will make a careless error).
bypass	A vector of the same length as skillLevels. For each parent variable, this represents the probability that the process will act as if that input condition is met, even if it is not met.
thresholds	If the input variables have more than two states, values that are equal to or higher than this threshold are considered true. It is assumed that the states of the variables are ordered from highest to lowest.

Details

The noisy-and distribution assumes that both the input and output variables are binary. Basically, the output should be true if all of the inputs are true. Let $S_k = 1$ if the kth input is true, and let r_k be the bypass parameter corresponding to the kth input variable. (If the S_k's represent a skill, then r_k represents the probability that an examinee who lacks that skill will bypass the need for that skill in solving the problem.) Then the probability of the true state for the output variable will be:

$$\Pr(X = True|\mathbf{S}) = r_0 \prod_k r_k^{1-S_k},$$

where r_0 (the noSlip parameter) is the probability that the output will be true even when all of the inputs are true.

It is assumed that all variables are ordered from highest to lowest state, so that the first state corresponds to "true" the others to false. If the input variable has more than two states, then it can be reduced to a binary variable by using the threshold argument. Any values which are equal to or higher than the threshold for that variable are assumed to be true. (In this case, higher means closer to the the beginning of the list of possible values.)

The noisy-min is a generalization

Value

For `calcNoisyAndTable`, a matrix whose rows correspond configurations of the parent variable states (`skillLevels`) and whose columns correspond to `obsLevels`. Each row of the table is a probability distribution, so the whole matrix is a conditional probability table. The order of the parent rows is the same as is produced by applying `expand.grid` to `skillLevels`.

For `calcNoisyAndFrame` a data frame with additional columns corresponding to the entries in `skillLevels` giving the parent value for each row.

Note

This is related to the DINA and NIDA models, but uses a slightly different parameterization. In particular, if the `noSlip` parameter is omitted, it is a noisy input deterministic and-gate (NIDA), and if the `bypass` parameters are omitted, it is similar to a deterministic input noisy and-gate (DINA), except is lacks a guessing parameter.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann.

Diez, F. J. (1993) Parameter adjustment in Bayes networks. The generalized noisy OR-gate. In Heckerman and Mamdani (eds) *Uncertainty in Artificial Intelligence 93.* Morgan Kaufmann. 99–105.

Srinivas, S. (1993) A generalization of the Noisy-Or model, the generalized noisy OR-gate. In Heckerman and Mamdani (eds) *Uncertainty in Artificial Intelligence 93.* Morgan Kaufmann. 208–215.

See Also

`calcDSTable`, `calcDNTable`, `calcDPCTable`, `expand.grid`, `calcNoisyOrTable`

Examples

```
## Logical and table
and <- calcNoisyAndFrame(list(c("True","False"),c("True","False")),
                                 c("Right","Wrong"))

## DINA, logical-and except that is allows for a small chance of slipping.
dina <- calcNoisyAndFrame(list(c("True","False"),c("True","False")),
                          noSlip=.9)

##NIDA, logical-and except that inputs can randomly be bypassed
nida <- calcNoisyAndFrame(list(c("True","False"),c("True","False")),
                          bypass=c(.3,.4))
```

```
##Full Noisy And distribution
noisyAnd <- calcNoisyAndFrame(list(c("True","False"),c("True","False")),
                          noSlip=.9,bypass=c(.3,.4))

thresh <- calcNoisyAndFrame(list(c("H","M","L"),c("H","M","L")),
                                  c("Right","Wrong"),
                                  threshold=c("M","H"))
```

`calcNoisyOrTable` *Calculate the conditional probability table for a Noisy-Or distribution*

Description

Calculates the conditional probability table for a noisy-and distribution. This follows a logical model where at least one inputs must be true for the output to be true; however, some "noise" is allowed that produces random deviations from the pure logic.

Usage

```
calcNoisyOrTable(skillLevels, obsLevels = c("True", "False"),
                 suppression = rep(0, length(skillLevels)), noGuess = 1,
                 thresholds = sapply(skillLevels, function(states) states[1]))
calcNoisyOrFrame(skillLevels, obsLevels = c("True", "False"),
                 suppression = rep(0, length(skillLevels)), noGuess = 1,
                 thresholds = sapply(skillLevels, function(states) states[1]))
```

Arguments

`skillLevels`	A list of character vectors giving names of levels for each of the condition variables.
`obsLevels`	A character vector giving names of levels for the output variables from highest to lowest. As a special case, can also be a vector of integers. Its length should be 2, and the first value is considered to be logically equivalent to "true".
`suppression`	A vector of the same length as `skillLevels`. For each parent variable, this represents the probability that the process will act as if that input condition is not met, even if it is met.
`noGuess`	A scalar value between 0 and 1. This represents the probability that the the output will be false even when all of the inputs are false (e.g., 1-guessing probability).
`thresholds`	If the input variables have more than two states, values that are equal to or higher than this threshold are considered true. It is assumed that the states of the variables are ordered from highest to lowest.

Details

The noisy-or distribution assumes that both the input and output variables are binary. Basically, the output should be true if any of the inputs are true. Let $S_k = 1$ if the kth input is true, and let q_k be the `suppression` parameter corresponding to the kth input variable. (If the S_k's represent a skill, then q_k represents the probability that an examinee who has that skill will fail to correctly apply it.) Then the probability of the true state for the output variable will be:

$$\Pr(X = True|\mathbf{S}) = 1 - q_0 \prod_k q_k^{1-S_k},$$

where q_0 (the `noGuess` parameter) is the probability that the output will be false even when all of the inputs are false.

It is assumed that all variables are ordered from highest to lowest state, so that the first state corresponds to "true" the others to false. If the input variable has more than two states, then it can be reduced to a binary variable by using the `threshold` argument. Any values which are equal to or higher than the `threshold` for that variable are assumed to be true. (In this case, higher means closer to the the beginning of the list of possible values.)

Value

For `calcNoisyOrTable`, a matrix whose rows correspond configurations of the parent variable states (`skillLevels`) and whose columns correspond to `obsLevels`. Each row of the table is a probability distribution, so the whole matrix is a conditional probability table. The order of the parent rows is the same as is produced by applying `expand.grid` to `skillLevels`.

For `calcNoisyOrFrame` a data frame with additional columns corresponding to the entries in `skillLevels` giving the parent value for each row.

Note

This is related to the DINO and NIDO models, but uses a slightly different parameterization. In particular, if the `noSlip` parameter is omitted, it is a noisy input deterministic and-gate (NIDO), and if the `bypass` parameters are omitted, it is similar to a deterministic input noisy and-gate (DINO), except is lacks a slip parameter.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann.

Diez, F. J. (1993) Parameter adjustment in Bayes networks. The generalized noisy OR-gate. In Heckerman and Mamdani (eds) *Uncertainty in Artificial Intelligence 93.* Morgan Kaufmann. 99–105.

Srinivas, S. (1993) A generalization of the Noisy-Or model, the generalized noisy OR-gate. In Heckerman and Mamdani (eds) *Uncertainty in Artificial Intelligence 93.* Morgan Kaufmann. 208–215.

See Also

calcDSTable, calcDNTable, calcDPCTable, expand.grid, calcNoisyOrTable

Examples

```
## Logical or table
or <- calcNoisyOrFrame(list(c("True","False"),c("True","False")),
                                  c("Right","Wrong"))


## DINO, logical-or except that is allows for a small chance of slipping.
dino <- calcNoisyOrFrame(list(c("True","False"),c("True","False")),
                          noGuess=.9)


##NIDO, logical-or except that inputs can randomly be bypassed
nido <- calcNoisyOrFrame(list(c("True","False"),c("True","False")),
                          suppression=c(.3,.4))

##Full Noisy Or distribution
noisyOr <- calcNoisyOrFrame(list(c("True","False"),c("True","False")),
                          noGuess=.9,suppression=c(.3,.4))


thresh <- calcNoisyOrFrame(list(c("H","M","L"),c("H","M","L")),
                                  c("Right","Wrong"),
                                  threshold=c("M","H"))
```

---

colorspread *Produces an ordered palate of colours with the same hue.*

Description

This takes a colour specification, and produces an ordered series of colours by manipulating the saturate (and possibly value) of the color, leaving the hue constant. This produces a colour palate suitable for plotting ordered factors, which looks good on a colour display, but also reproduces well on a grayscale printer (or for persons with limited colour perception).

Usage

```
colorspread(col, steps, maxsat = FALSE, rampval = FALSE)
```

Arguments

col	A color in any format suitable as input to col2rgb.
steps	A integer describing the number of colors to create.
maxsat	A logical value. If true, the final color in the series will have saturation 1, instead of whatever is appropriate for the input.
rampval	A logical value. If true, the value as well as the saturation of the color is ramped.

Details

The colour is converted to a RGB value using col2rgb and then to an HSV value using rgb2hsv. The saturation is then scaled into steps equal intervals. If requested, the value is scaled as well.

Value

A character vectors of length steps giving the colour palate from lowest to highest intensity. This is suitable to passing to the col argument of most graphics functions.

Note

Many of the built-in colours come with 4 intensity variants are meant to work well together. In some cases an expression like paste("firebrick",1:4,sep="") may work better than colorspread. To see the built-in colours, use the colors function.

Author(s)

Russell Almond

See Also

compareBars, link{stackedBars}

Examples

```
barplot(rep(1,4),col=colorspread("slategray",4))
barplot(rep(1,4),col=colorspread("slategray",4,maxsat=TRUE))
barplot(rep(1,4),col=colorspread("violetred",4))
barplot(rep(1,4),col=colorspread("violetred",4,rampval=TRUE))
```

---

compareBars *Produces comparison stacked bar charts for two sets of groups*

Description

This produces set of stacked bar charts grouped for comparison between two groups. For example, if suppose that there is a set of probabilities over a collection of proficiency variables measures both before and after obtaining a certain piece of evidence. The compareBars function would produce stacked bar charts which compare the prior and posterior probabilities for each variable.

Usage

```
compareBars(data1, data2, profindex,
            groupNames = c(deparse(data1), deparse(data2)), ...,
            ylim = c(min(offsets) - 0.25, max(1 + offsets)),
            cex.names = par("cex.axis"), digits = 2, legend.loc = c(0,1),
            legend.cex = par("cex"), col = par("col"), col1 = NULL,
            col2 = NULL, main = NULL, sub = NULL, xlab = NULL,
            ylab = NULL, rotlab = FALSE)

compareBars2(data1, data2, profindex,
             groupNames=c("Prior","Post"), error.bars=2, scale=100,
             err.col="gray20", ..., ylim = NULL)
```

Arguments

data1	Data set with first (prior) values
data2	Data set with second (posterior) values
profindex	Index of one of the proficiency levels which will be used as the baseline for the stacked bar charts.
groupNames	Names of the groups represented by data1 and data2 respectively.
...	Other arguments to barplot.
ylim	Default limits for Y axis.
cex.names	Character magnification for names.
digits	Number of digits for overlaid numeric variables.
legend.loc	Location for legend, see legend.
legend.cex	Character magnification for legend.
col	The normal graphics col parameter (see par, passed through to other graphics operators using ... Is also the default for col1 and col2 if those values are not supplied.
col1	Color scale for the first data set. This should be a vector of colors equal to the number of groups.
col2	Color scale for the second data set. This should be a vector of colors equal to the number of groups.
main	Character scalar giving main title (see title).
sub	Character scalar giving sub title (see title).
xlab	Character scalar giving x-axis label (see title).
ylab	Character scalar giving x-axis label (see title).
rotlab	If TRUE labels are rotated 90 degrees.
error.bars	The number of standard errors for error bars.
err.col	The color for error bars.
scale	Scales data as probabilities (scale=1) or percentages (scale=100).

Value

Invisibly returns the x-co-ordinates of the bars.

Note

The function `compareBars2` is a somewhat experimental extension to `compareBars` which adds error bars to the posterior. The result is not entirely satisfactory, and this function may change with future releases.

Author(s)

Russell Almond

References

Almond, R. G., Shute, V. J., Underwood, J. S., and Zapata-Rivera, J.-D (2009). Bayesian Networks: A Teacher's View. *International Journal of Approximate Reasoning.* **50**, 450-460.

See Also

`stackedBars`, `colorspread`, `buildFactorTab`, `barplot`

Examples

```
margins.prior <- data.frame (
 Trouble=c(Novice=.19,Semester1=.24,Semester2=.28,Semseter3=.20,Semester4=.09),
 NDK=c(Novice=.01,Semester1=.09,Semester2=.35,Semseter3=.41,Semester4=.14),
 Model=c(Novice=.19,Semester1=.28,Semester2=.31,Semseter3=.18,Semester4=.04)
)

margins.post <- data.frame(
 Trouble=c(Novice=.03,Semester1=.15,Semester2=.39,Semseter3=.32,Semester4=.11),
 NDK=c(Novice=.00,Semester1=.03,Semester2=.28,Semseter3=.52,Semester4=.17),
 Model=c(Novice=.10,Semester1=.25,Semester2=.37,Semseter3=.23,Semester4=.05))

foo <-
compareBars(margins.prior,margins.post,3,c("Prior","Post"),
            main="Margins before/after Medium Trouble Shooting Task",
            sub="Observables:  cfgCor=Medium, logCor=High, logEff=Medium",
            legend.loc = "topright",
            cex.names=.75, col1=hsv(h=.1,s=.2*1:5-.1,alpha=1),
            col2=hsv(h=.6,s=.2*1:5-.1,alpha=1))

compareBars2(margins.prior,25*margins.post,3,c("Prior","Post"),
            main="Margins before/after Medium Trouble Shooting Task",
            sub="Observables:  cfgCor=Medium, logCor=High, logEff=Medium",
            legend.loc = "topright",
            cex.names=.75, col1=hsv(h=.1,s=.2*1:5-.1,alpha=1),
            col2=hsv(h=.6,s=.2*1:5-.1,alpha=1))
```

`Compensatory` *DiBello–Samejima combination function*

Description

These functions take a vector of "effective theta" values for a collection of parent variables and calculates the effective theta value for the child variable according to the named rule. Used in calculating DiBello–Samejima and DiBello–Normal probability tables. These all have one slope parameter (alpha) per parent variable.

Usage

```
Compensatory(theta, alphas, beta)
Conjunctive(theta, alphas, beta)
Disjunctive(theta, alphas, beta)
```

Arguments

`theta`	A matrix of effective theta values whose columns correspond to parent variables and whose rows correspond to possible skill profiles.
`alphas`	A vector of discrimination parameters in the same order as the columns of `theta`. (Note these function expect discrimination parameters and not log discrimination parameters as used in `calcDSTable`.)
`beta`	A difficulty (-intercept) parameter.

Details

For `Compensatory`, the combination function for each row is:

$$(alphas[1] * theta[1] + ... + alphas[K] * theta[K])/sqrt(K) - beta$$

where K is the number of parents. (The $\sqrt{K}$ is a variance stabilization parameter.)

For `Conjunctive`, the combination function for each row is:

$$min(alphas[1] * theta[1], ..., alphas[K] * theta[K]) - beta$$

For `Disjunctive`, the combination function for each row is:

$$max(alphas[1] * theta[1], ..., alphas[K] * theta[K]) - beta$$

Value

A vector of normal deviates corresponding to the effective theta value. Length is the number of rows of `thetas`.

Note

These functions expect the unlogged discrimination parameters, while `calcDSTable` expect the log of the discrimination parameters. The rationale is that log discrimination is bound away from zero, and hence a more natural space for MCMC algorithms. However, it is poor programming design, as it is liable to catch the unwary.

These functions are meant to be used as structure functions in the DiBello–Samejima and DiBello–Normal models. Other structure functions are possible and can be excepted by those functions as long as they have the same signature as these functions.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

See Also

`effectiveThetas`,`calcDSTable`, `calcDNTable`, `calcDPCTable`, `OffsetConjunctive`, `eThetaFrame`

Examples

```
thetas <- expand.grid(list(S1=seq(1,-1), S2 = seq(1, -1)))
Compensatory(thetas, c(S1=1.25,S2=.75), 0.33)
Conjunctive(thetas, c(S1=1.25,S2=.75), 0.33)
Disjunctive(thetas, c(S1=1.25,S2=.75), 0.33)

skill <- c("High","Medium","Low")
eThetaFrame(list(S1=skill,S2=skill), c(S1=1.25,S2=.75), 0.33, "Compensatory")
eThetaFrame(list(S1=skill,S2=skill), c(S1=1.25,S2=.75), 0.33, "Conjunctive")
eThetaFrame(list(S1=skill,S2=skill), c(S1=1.25,S2=.75), 0.33, "Disjunctive")
```

`CPA` *Representation of a conditional probability table as an array.*

Description

A conditional probability table for a node can be represented as a array with the first p dimensions representing the parent variables and the last dimension representing the states of the node. Given a set of values for the parent variables, the values in the last dimension contain the conditional probabilities corresponding conditional probabilities. A `CPA` is a special `array` object which represents a conditional probability table.

Usage

```
is.CPA(x)
as.CPA(x)
```

Arguments

x	Object to be tested or coerced into a CPA.

Details

One way to store a conditional probability table is as an array in which the first p dimensions represent the parent variables, and the $p + 1$ dimension represents the child variable. Here is an example with two parents variables, A and B, and a single child variable, C:

```
, , C=c1
```

	b1	b2	b3
a1	0.07	0.23	0.30
a2	0.12	0.25	0.31
a3	0.17	0.27	0.32
a4	0.20	0.29	0.33

```
, , C=c2
```

	b1	b2	b3
a1	0.93	0.77	0.70
a2	0.88	0.75	0.69
a3	0.83	0.73	0.68
a4	0.80	0.71	0.67

[Because R stores (and prints) arrays in column-major order, the last value (in this case tables) is the one that sums to 1.]

The CPA class is a subclass of the array class (formally, it is class c("CPA","array")). The CPA class interprets the dimnames of the array in terms of the conditional probability table. The first p values of names(dimnames(x)) are the input names of the edges (see NodeInputNames() or the variable names (or the parent variable, see NodeParents(), if the input names were not specified), and the last value is the name of the child variable. Each of the elements of dimnames(x) should give the state names (see NodeStates()) for the respective value. In particular, the conversion function as.CPF() relies on the existence of this meta-data, and as.CPA() will raise a warning if an array without the appropriate dimnames is supplied.

Although the intended interpretation is that of a conditional probability table, the normalization constraint is not enforced. Thus a CPA object could be used to store likelihoods, probability potentials, contingency table counts, or other similarly shaped objects. The function normalize scales the values of a CPA so that the normalization constraint is enforced.

The method NodeProbs() returns a CPA object. The function as.CPA() is designed to convert between CPFs (that is, conditional probability tables stored as data frames) and CPAs. It assumes

that the factors variables in the data frame represent the parent variables, and the numeric values represent the states of the child variable. It also assumes that the names of the numeric columns are of the form *varname*.*state*, and attempts to derive variable and state names from that.

If the argument to `as.CPA(x)` is an array, then it assumes that the `dimnames(x)` and `names(dimnames(x))` are set to the states of the variables and the names of the variables respectively. A warning is issued if the names are missing.

Value

The function `is.CPA()` returns a logical value indicating whether or not the `is(x,"CPA")` is true.

The function `as.CPA` returns an object of class `c("CPA","array")`, which is essentially an array with the dimnames set to reflect the variable names and states.

Note

The obvious way to print a `CPA` would be to always show the child variable as the rows in the individual tables, with the parents corresponding to rows and tables. R, however, internally stores arrays in column-major order, and hence the rows in the printed tables always correspond to the second dimension. A new print method for `CPA` would be nice.

This is an S3 object, as it just an array with a special interpretation.

Author(s)

Russell Almond

See Also

`NodeProbs()`, `Extract.NeticaNode`, `CPF`, `normalize()`

Examples

```
# Note:  in R 4.0, the factor() call is required.
arf <- data.frame(A=factor(rep(c("a1","a2"),each=3)),
                  B=factor(rep(c("b1","b2","b3"),2)),
                  C.c1=1:6, C.c2=7:12, C.c3=13:18, C.c4=19:24)
arfa <- as.CPA(arf)

arr1 <- array(1:24,c(4,3,2),
            dimnames=list(A=c("a1","a2","a3","a4"),B=c("b1","b2","b3"),
                          C=c("c1","c2")))
arr1a <- as.CPA(arr1)


## Not run:
  ## Requires RNetica
  as.CPA(node[])

## End(Not run)
```

CPF *Representation of a conditional probability table as a data frame.*

Description

A conditional probability table for a node can be represented as a data frame with a number of factor variables representing the parent variables and the remaining numeric values representing the conditional probabilities of the states of the nodes given the parent configuration. Each row represents one configuration and the corresponding conditional probabilities. A CPF is a special data.frame object which represents a conditional probability table.

Usage

```
is.CPF(x)
as.CPF(x)
```

Arguments

x	Object to be tested or coerced into a CPF.

Details

One way to store a conditional probability table is a table in which the first several columns indicate the states of the parent variables, and the last several columns indicate probabilities for several child variables. Consider the following example:

	A	B	C.c1	C.c2	C.c3	C.c4
[1,]	a1	b1	0.03	0.17	0.33	0.47
[2,]	a2	b1	0.05	0.18	0.32	0.45
[3,]	a1	b2	0.06	0.19	0.31	0.44
[4,]	a2	b2	0.08	0.19	0.31	0.42
[5,]	a1	b3	0.09	0.20	0.30	0.41
[6,]	a2	b3	0.10	0.20	0.30	0.40

In this case the first two columns correspond to parent variables A and B. The variable A has two possible states and the variable B has three. The child variable is C and it has for possible states. The numbers in each row give the conditional probabilities for those states give the state of the child variables.

The class CPF is a subclass of data.frame (formally, it is class c("CPF","data.frame")). Although the intended interpretation is that of a conditional probability table, the normalization constraint is not enforced. Thus a CPF object could be used to store likelihoods, probability potentials, contingency table counts, or other similarly shaped objects. The function normalize scales the numeric values of CPF so that each row is normalized.

The [method for a NeticaNode returns a CPF (if the node is not deterministic).

The function as.CPF() is designed to convert between CPAs (that is, conditional probability tables stored as arrays) and CPFs. In particular, as.CPF is designed to work with the output of NodeProbs() or a similarly formatted array. It assumes that names(dimnames(x)) are the names of the variables, and dimnames(x) is a list of character vectors giving the names of the states of the variables. (See CPA for details.) This general method should work with any numeric array for which both dimnames(x) and names(dimnames(x)) are specified.

The argument x of as.CPF() could also be a data frame, in which case it is permuted so that the factor variable are first and the class tag "CDF" is added to its class.

Value

The function is.CPF() returns a logical value indicating whether or not the is(x,"CDF") is true.

The function as.CPF returns an object of class c("CPF","data.frame"), which is essentially a data frame with the first couple of columns representing the parent variables, and the remaining columns representing the states of the child variable.

Note

The parent variable list is created with a call expand.grid(dimnames(x)[1:(p-1)]). This produces conditional probability tables where the first parent variable varies fastest. The Netica GUI displays tables in which the last parent variable varies fastest.

Note, this is an S3 class, as it is basically a data.frame with special structure.

Change in R 4.0. Note that under R 4.0, character vectors are no longer automaticall coerced into factors when added to data frames. This is probably a good thing, as the code can assume that a column that is a factor is an index for a variable, and one that is a character is a comment or some other data.

Author(s)

Russell Almond

See Also

NodeProbs(), Extract.NeticaNode, CPA, normalize()

Examples

```
# Note:  in R 4.0, the factor() call is required.
arf <- data.frame(A=factor(rep(c("a1","a2"),each=3)),
                  B=factor(rep(c("b1","b2","b3"),2)),
                  C.c1=1:6, C.c2=7:12, C.c3=13:18, C.c4=19:24)
arf <- as.CPF(arf)


arr <- array(1:24,c(2,3,4),
             dimnames=list(A=c("a1","a2"),B=c("b1","b2","b3"),
                           C=c("c1","c2","c3","c4")))
arrf <- as.CPF(arr)
stopifnot(
```

```
  is.CPF(arrf),
  all(levels(arrf$A)==c("a1","a2")),
  all(levels(arrf$B)==c("b1","b2","b3")),
  nrow(arrf)==6, ncol(arrf)==6
)

##Warning, this is not the same as arf, rows are permuted.
as.CPF(as.CPA(arf))

## Not run:
  ## Requires RNetica
  as.CPF(NodeProbs(node))

## End(Not run)
```

---

cptChi2 *Compare and observed to an expected conditional probability table.*

Description

Each row of a conditional probability frame (CPF) is a probability distribution. Observed data comes in the form of a contingency table (or expected contingency table) where the rows represent configurations of the parent variables and the columns represent the state of the child variable. The cptChi2 calculates the chi-squared data fit statistic for each row of the table, and sums them together. This provides a test of the plausibility that the data came from the proposed model.

Usage

```
cptChi2(obs, exp)
```

Arguments

obs	A CPF which contains the observed data.
exp	A CPF which contains the candidate distribution.

Details

The cannonical use for this plot is to test the conditional probability model used in particular Bayesian network. The exp argument is the proposed table. If the parents are fully observed, then the obs argument would be a contingency table with the rows representing parent configurations and the columns the data. If the parents are not fully observed, then obs can be replaced by an expected contingincy table (see Almond, et al, 2015).

To avoid potential problems with zero cells, the data table is modified by adding the prior. That is the observed value is taken as obs+exp. This should not have a big effect if the sample size is large.

Value

The sum of the chi-squres for all of the tables with the degrees of freedom. The degrees of freedom is given as an attribute df.

Note

This is called a chi-square statistic, and the percentage points of the chi-square distribution can be used for reference. However, it is unclear whether the statistic is actually distributed according to the chi-square distribution under the hypothesis that the data come from the distribution given in exp. In particular, if an expected table is used instead of an actual table, the distribution is likely to not be exactly chi-squared.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 10.

Sinharay, S. and Almond, R.G. (2006). Assessing Fit of Cognitively Diagnostic Models: A case study. *Educational and Psychological Measurement*. **67**(2), 239–257.

See Also

betaci, OCP, barchart.CPF, OCP.CPF

Examples

```
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

cpfTheta <- calcDPCFrame(list(),skill1l,numeric(),0,rule="Compensatory",
                         link="normalLink",linkScale=.5)

## Compatable data
datTheta <- cpfTheta ## Copy to get the shape
datTheta[1,] <- rmultinom(1,25,cpfTheta[1,])

cptChi2(datTheta,cpfTheta)

## Incompatable data
datThetaX <- cpfTheta ## Copy to get the shape
datThetaX[1,] <- rmultinom(1,100,c(.05,.45,.5))

cptChi2(datThetaX,cpfTheta)

cptComp <- calcDPCFrame(list(S2=skill2l,S1=skill1l),correctL,
                        lnAlphas=log(c(1.2,.8)), betas=0,
                        rule="Compensatory")

cptConj <- calcDPCFrame(list(S2=skill2l,S1=skill1l),correctL,
                        lnAlphas=log(c(1.2,.8)), betas=0,
```

```
                        rule="Conjunctive")

datComp <- cptComp
for (i in 1:nrow(datComp))
    ## Each row has a random sample size.
    datComp[i,3:4] <- rmultinom(1,rnbinom(1,mu=15,size=1),cptComp[i,3:4])

datConj <- cptConj
for (i in 1:nrow(datConj))
    ## Each row has a random sample size.
    datConj[i,3:4] <- rmultinom(1,rnbinom(1,mu=15,size=1),cptConj[i,3:4])

## Compatable
cptChi2(datConj,cptConj)
## Incompatable
cptChi2(datComp,cptConj)


cptPC1 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                       lnAlphas=log(1),
                       betas=list(full=c(S1=0,S2=999),partial=c(S2=999,S2=0)),
                       rule="OffsetDisjunctive")
cptPC2 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                       lnAlphas=log(1),
                       betas=list(full=c(S1=0,S2=999),partial=c(S2=1,S2=0)),
                       rule="OffsetDisjunctive")


datPC1 <- cptPC1
for (i in 1:nrow(datPC1))
    ## Each row has a random sample size.
    datPC1[i,3:5] <- rmultinom(1,rnbinom(1,mu=25,size=1),cptPC1[i,3:5])

datPC2 <- cptPC2
for (i in 1:nrow(datPC2))
    ## Each row has a random sample size.
    datPC2[i,3:5] <- rmultinom(1,rnbinom(1,mu=25,size=1),cptPC2[i,3:5])

## Compatable
cptChi2(datPC1,cptPC1)
## Inompatable
cptChi2(datPC2,cptPC1)
```

`dataTable` *Constructs a table of counts from a set of discrete observations.*

Description

This constructs a table of counts in a special format useful for conditional probability tables. The rows correspond to configurations of the parent variables and the columns correspond to possible

states of the child variables.

Usage

```
dataTable(data, parents, child, childStates)
```

Arguments

data	A data frame whose columns contain variables corresponding to parent and child.
parents	A vector of names for the columns in data corresponding to the parent variables.
child	The name of the child variable, should refer to a column in data.
childStates	A character vector giving names of levels for the output variables from highest to lowest.

Details

Apply the function table to generate a table of counts for the indicated variables (subsetting the table if necessary). Then reformats this into a matrix whose columns correspond to the child variable.

Value

A matrix whose columns correspond to childStates and whose rows correspond to the possible combinations of parents.

Author(s)

Russell Almond

See Also

table, calcDSllike

Examples

```
skill1l <- c("High","Medium","Low")
skill3l <- c("High","Better","Medium","Worse","Low")
correctL <- c("Correct","Incorrect")

x <- read.csv(system.file("testFiles", "randomPinned100.csv",
                          package="CPTtools"),
              header=TRUE, as.is=TRUE)
x[,"Skill1"] <- ordered(x[,"Skill1"],skill1l)
x[,"Skill3"] <- ordered(x[,"Skill3"],skill3l)
x[,"Comp.Correct"] <- ordered(x[,"Comp.Correct"],correctL)


tab <- dataTable(x, c("Skill1","Skill3"),"Comp.Correct",correctL)
data.frame(expand.grid(list(Skill1=skill1l,Skill3=skill3l)),tab)
```

`defaultAlphas` *Generates default values for Alpha and Beta parameters*

Description

The expected shape of the Alpha and Beta parameters depends on the type of rule. If `isOffsetRule(rule)` is true, then the betas should be a vector of the same length as `pnames` and alphas should be a scalar. If it is false, then alphas is the vector and betas is the scalar.

Usage

```
defaultAlphas(rule, pnames, states=c("Yes","No"),
              link="partialCredit")
defaultBetas(rule, pnames, states=c("Yes","No"),
             link="partialCredit")
```

Arguments

`rule`	A rule (e.g., `Compensatory`, `OffsetConjunctive` or the name of a rule.
`pnames`	A character vector giving the names of the parent variables.
`states`	A character vector giving the names of the states of the child variable.
`link`	A link function (e.g., `partialCredit`) or the name of a link function.

Details

Rules come in two styles:

Linear rules, e.g., `Compensatory` These take multiple slopes (alphas) and a single intercept (beta).

Offset rules, e.g., `OffsetConjunctive` These take multiple intercepts (betas) and a single slope (alpha).

The value of `getOffsetRules()` determines which rules use which style. These functions are useful for getting the shape right when the parameters have not yet been established.

There may or may not be different values for alpha or beta for each state transition depending on the link function. If multiples are allowed then a list of alphas or betas is returned, corresponding to the _transitions_ between the states (so the length of the list is one less than the length of `states`).

The shape of the result is as follows:

partialCredit With this link function both alphas and betas will be lists if the number of states is greater than 2.

gradedResponse With this link function, the betas are a list and the alphas are a single vector. Note that the betas should be in increasing order, but this is not enforced.

normalLink Both alphas and betas are vectors.

Value

Either a named vector (the names are pnames) of the default value (1 for alphas, 0 for betas) or scalar value, or a named list of such. If a list, then the names follow the names of the states with the last one omitted.

Note

As more link functions are added, this needs to be expanded.

Author(s)

Russell Almond

See Also

calcDPCTable, getOffsetRules, partialCredit, gradedResponse, normalLink

Examples

```
defaultAlphas("Compensatory",c("S1","S2"),c("Yes","No"))
defaultBetas("Compensatory",c("S1","S2"),c("Yes","No"))

defaultAlphas("Compensatory",c("S1","S2"),c("Yes","Maybe","No"))
defaultBetas("Compensatory",c("S1","S2"),c("Yes","Maybe","No"))

defaultAlphas("Compensatory",c("S1","S2"),c("Yes","Maybe","No"),
              "gradedResponse")
defaultBetas("Compensatory",c("S1","S2"),c("Yes","Maybe","No"),
             "gradedResponse")

defaultAlphas("Compensatory",c("S1","S2"),c("Yes","Maybe","No"),
              "normalLink")
defaultBetas("Compensatory",c("S1","S2"),c("Yes","Maybe","No"),
             "normalLink")

defaultAlphas("OffsetConjunctive",c("S1","S2"),c("Yes","No"))
defaultBetas("OffsetConjunctive",c("S1","S2"),c("Yes","No"))

defaultAlphas("OffsetConjunctive",c("S1","S2"),
              c("Yes","Maybe","No"),"gradedResponse")
defaultBetas("OffsetConjunctive",c("S1","S2"),
               c("Yes","Maybe","No"),"gradedResponse")
```

EAPBal *Produces a graphical balance sheet for EAP or other univarate statistics.*

Description

This provides a graph of the history of any given measure of performance. The input should be a list of values `hist` and the list of the `contexts` or game levels associated with them.

Usage

```
EAPBal(hist, contexts = names(hist), obs = NULL,
varname = deparse(substitute(hist)),
elim = c(-1, 1), etic = 4, title = paste("EAP Balance Sheet:", varname),
col = "slategray", posCol = "cyan", negCol = "red",
stripCol = c("white", "lightgray"), lcex = 0.65)
```

Arguments

`hist`	A vector of doubles giving the value of the statistic at each time point.
`contexts`	A list of names for the events associated with the values. Defaults to the names of the `hist` argument.
`obs`	Observable values associated with each level. If supplied it should be a vector as long as `hist`.
`varname`	The name of the variable which is being monitored.
`elim`	A vector of length two giving the minimum and maximum EAP value.
`etic`	How many tic marks to use on the EAP axis.
`title`	The main title for the overall plot.
`col`	Color for the EAP bars.
`posCol`	Color for positive movements.
`negCol`	Color for negative movements.
`stripCol`	The colors to be used for the time step labels. Setting this to a vector of two colors creates alternate color stripes. Set this to "white" to disable that effect.
`lcex`	Character expansion size for labels.

Details

Madigan, Mosurski and Almond (1997) described a graphical weight of evidence balance sheet. The key to this display is following a particular node (or more properly hypothesis involving a node) across time as evidence is entered into the system. There are two columns, one showing the probability distribution, and one showing the weight of evidence—the change in probabilities.

Often, the nodes in an ordered categorical value are assigned numeric values and an *expected a posteriori* or EAP value is calculated instead. The EAP balance sheet is similar to the woe balance sheet, except that it now uses a single numeric value.

The input is a sequence of EAP value, labeled by the event which causes the change. The output is a graphical display in three columns.

Value

The midpoints of the bars (see `barplot`) are returned invisibly.

Author(s)

Russell Almond

References

Madigan, D., Mosurski, K. and Almond, R. (1997) Graphical explanation in belief networks. *Journal of Computational Graphics and Statistics*, **6**, 160-181.

See Also

`woeBal`, `barplot`, `colors`

Examples

```
sampleSequence <- read.csv(system.file("testFiles","SampleStudent.csv",
                                       package="CPTtools"),
                           header=TRUE,row.names=1)
SolveGeometricProblems <- sampleSequence$H-sampleSequence$L
names(SolveGeometricProblems) <- rownames(sampleSequence)
EAPBal(SolveGeometricProblems,  lcex=1.25)
EAPBal(SolveGeometricProblems,rownames(sampleSequence),
      varname="Solve Geometric Problems",
      obs=sampleSequence[,"Acc"],lcex=1.25)
```

---

effectiveThetas *Assigns effective theta levels for categorical variable*

Description

Calculates a vector of normal quantiles corresponding to effective theta levels for a categorical variable for use in a DiBello-Samejima distribution.

Usage

```
effectiveThetas(nlevels)
```

Arguments

`nlevels`	Integer giving the number of levels of the categorical variable.

Details

The DiBello–Samejima models map levels of categorical values into effective "theta" values, or corresponding continuous values. These can then be input into IRT equations to calculate cell probabilities.

The default algorithm works by assuming that the categories are created by cutting the normal distribution into equal probability intervals. The value returned for each interval is the midpoint (wrt the normal measure) of that interval.

Value

A vector of normal quantiles of length `nlevels`.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015). *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Examples

```
effectiveThetas(5)
```

`eThetaFrame` *Constructs a data frame showing the effective thetas for each parent combination.*

Description

This evaluates the combination function but not the link function of of an effective theta distribution. It produces a table of effective thetas one for each configuration of the parent values according to the combination function given in the `model` argument.

Usage

```
eThetaFrame(skillLevels, lnAlphas, beta, rule = "Compensatory")
```

Arguments

`skillLevels`	A list of character vectors giving names of levels for each of the condition variables.
`lnAlphas`	A vector of log slope parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`beta`	A vector of difficulty (-intercept) parameters. Its length should be either 1 or the length of `skillLevels`, depending on the choice of `rule`.
`rule`	Function for computing effective theta (see Details).

Details

The DiBello framework for creating for creating conditional probability tables for Bayesian network models using IRT-like parameters unfolds in three steps.

1. Each level of each input variable is assigned an "effective theta" value — a normal value to be used in calculations.
2. For each possible skill profile (combination of states of the parent variables) the effective thetas are combined using a combination function. This produces an "effective theta" for that skill profile. The function `rule` determines the rule for combination.
3. The effective theta is input into a link function (e.g., Samejima's graded-response function) to produce a probability distribution over the states of the outcome variables.

This function applies the first two of those steps and returns a data frame with the original skill levels and the effective thetas.

The parent (conditioning) variables are described by the `skillLevels` argument which should provide for each parent variable in order the names of the states ranked from highest to lowest value. The original method (Almond et al., 2001) used equally spaced points on the interval $[-1, 1]$ for the effective thetas of the parent variables. The current implementation uses the function `effectiveThetas` to calculate equally spaced points on the normal curve.

The combination of the individual effective theta values into a joint value for effective theta is done by the function reference by `rule`. This should be a function of three arguments: `theta` — the vector of effective theta values for each parent, `alphas` — the vector of discrimination parameters, and `beta` — a scalar value giving the difficulty. The initial distribution supplies five functions appropriate for use with `calcDSTable`: `Compensatory`, `Conjunctive`, and `Disjunctive`, `OffsetConjunctive`, and `OffsetDisjunctive`. The last two have a slightly different parameterization: `alpha` is assumed to be a scalar and `betas` parameter is vector valued. Note that the discrimination and difficulty parameters are built into the structure function and not the IRT curve.

Value

For a data frame with one column for each parent variable and an additional column for the effective theta values. The number of rows is the product of the number of states in each of the components of the `skillLevels` argument.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015). *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

See Also

effectiveThetas,Compensatory, OffsetConjunctive,calcDNTable, calcDSTable, calcDPCTable, expand.grid

Examples

```
skill <- c("High","Medium","Low")
eThetaFrame(list(S1=skill,S2=skill), log(c(S1=1.25,S2=.75)), 0.33,
           "Compensatory")
eThetaFrame(list(S1=skill,S2=skill), log(c(S1=1.25,S2=.75)), 0.33,
            "Conjunctive")
eThetaFrame(list(S1=skill,S2=skill), log(c(S1=1.25,S2=.75)), 0.33,
            "Disjunctive")
eThetaFrame(list(S1=skill,S2=skill), log(1.0), c(S1=0.25,S2=-0.25),
            "OffsetConjunctive")
eThetaFrame(list(S1=skill,S2=skill), log(1.0), c(S1=0.25,S2=-0.25),
            "OffsetDisjunctive")
```

---

ewoe.CPF — *Calculates the expected weight of evidence from a conditional probability frame.*

Description

The expected weight of evidence (EWOE) is a measure of how much information about a hypothesis can be learned from a potential observation. The hypothesis corresponds to a grouping of the rows of the CPF (and the negation of the hypothesis to the remaining rows).

Usage

```
ewoe.CPF(cpf, pos = 1L, neg = NULL)
```

Arguments

cpf	A conditional probability frame (CPF), data frame, or matrix whose columns represent states of one variable, and rows represent configurations of the parent variable(s).
pos	An expression for selecting rows of the cpf which corresponds to the hypothesis.
neg	An expression for selecting the rows corresponding to the complement of the hypothesis. (The default value is -pos if pos is numeric; !pos if pos is logical, and setdiff(rownames(cpf),pos if pos is a character vector.

Details

Good (1985) defines the weight of evidence E for a hypothesis H as

$$W(H:E) = log\frac{P(E|H)}{P(E|\not H)} = log\frac{P(H|E)}{P(\not H|E)} - log\frac{P(H)}{P(\not H)}.$$

The expected weight of evidence (Good and Card, 1971) looks at potential future observations to find which might have the highest weight of evidence. The expected weight of evidence is

$$EWOE(H:E) = \sum_{ein E} W(H:e)P(e|H).$$

In this calculation, the potential evidence corresponds to the columns of the (numericPart) of cpf. The hypothesis is found by splitting the rows. The pos and neg arguments can be any way of specifying a set of rows.

This is similar to the mutualInformation, only EWOE works for a binary hypothesis, while mutualInformation works for any number of states.

Value

A numeric value giving the weight of evidence in _centibans_ (where the logs are taken base 10 and the result is multiplied by 100).

Author(s)

Russell Almond

References

Good, I.J. (1985). Weight of Evidence: A brief survey. In Bernardo, J., DeGroot, M., Lindley, D. and Smith, A. (eds). *Bayesian Statistics 2*. North Holland. 249–269.

Good, I. J. and Card, W. (1971). The Diagnostic Process with Special Reference to Errors. *Methods of Information in Medicine*, 10, 176–188.

See Also

CPF, mutualInformation, expTable

Examples

```
ACED <- dplyr::inner_join(ACED.scores,ACED.items,by="SubjID")
expcr <- expTable(ACED,"cr","tCommonRatio1a")
ewoe.CPF(expcr,"H")
ewoe.CPF(expcr,c("H","M"))
```

`expTable` *Builds expected contingency tables.*

Description

The `table` function in base-R builds contingency tables from cross-classifying factors. If the factor is not known, but instead estimated using a Bayesian classifier, then instead of a single column with the factor value, there will be several columns with giving the probability that the individual is in each state. The table built is then an expected contingency table.

Usage

```
expTable(data, pvecVars, facVars,
         pvecregex = "P\\.<var>\\.\\.<state>")
pvecTable(data, pvarName,
         regx = sub("<var>",pvarName,"P\\.<var>\\.\\.<state>"))
catTable(data, fvarName,
         cc = contrasts(as.factor(dplyr::pull(data, fvarName)),
              contrasts = FALSE))
```

Arguments

`data`	This is a data frame where columns corresponding to probability vectors have a regular naming pattern.
`pvecVars`, `pvarName`	The names (or name for `pvecName`) of the estimated variable(s).
`facVars`, `fvarName`	The names (or name for `catTable`) of the factor variable(s).
`pvecregex`, `regx`	A regular expression template for finding the column names. The string "<var>" is replaced with the variable name (`pvecName`) and the string "<state>" is replaced with a regular expression that extract the state name.
`cc`	This is a contrasts matrix (see `contrasts` which shows how to convert the factor variable to columns.

Details

For an individual, i, let Y_i be a fully observed variable which takes on states ${ y_1, ..., y_m}$. Let S_i be a latent with states ${s_1, ..., s_k}$, and let $P(S_i)$ is an estimate of S_i, so it is a vector over the states.

The expected matrix is formed as follows:

1. Initialize a k by m (rows correspond to states of S and columns to states of Y) matrix with 0.
2. For each individual, add $P(S_i)$ to the column corresponding to Y_i

The result is the expected value of the contingency table. The general case can be handled with an Einstein sum (see einsum), with a rule "za,zb,zc,... -> abc...".

The assumption is that the estimates of the latent variable are saved in columns with a regular naming convention. For example, the ACED data set uses the names P.cr..H, P.cr..M and P.cr..L are the high, medium and low probabilities for the common ratio variable. The regular expression which will capture the names of the states is "P\.cr\.\.(\w+)", where "\w+" is one or more word constituent characters. The parentheses around the last part are used to extract the state names.

Internally, the function substitutes the value of pvecName for "<var>", and the "(\w+)" is substituted for "<state>". If this regular expression doesn't work for grabbing the state names, a better expression can be substituted, but it should be the first sub-expression marked with parentheses. Note also that the period has a special meaning in regular expressions so it needs to be quoted. Note also, that the backslash needs to be quoted in R strings.

Value

The functions pvecTable and catTable return a matrix with rows corresponding to the original data, and columns to the states of the variable.

The function expTable produces an array whose dimensions correspond to the states of probability and factor variables.

Author(s)

Russell Almond

References

Observable Analysis paper (in preparation).

See Also

mutualInformation, gkGamma, table, contrasts, einsum

Examples

```
data(ACED)
ACED.joined <- dplyr::inner_join(ACED.scores,ACED.items,by="SubjID")
head(pvecTable(ACED.joined,"cr"))
head(catTable(ACED.joined,"tCommonRatio1a"))
expTable(ACED.joined,"cr","tCommonRatio1a")
```

`fcKappa` *Functions for measuring rater agreement.*

Description

The functions take a "confusion matrix", a square matrix were the rows and columns represent classifications by two different raters, and compute measures of rater agreement. Cohen's kappa (`fcKappa`) is corrected for random labeling of ratings. Goodman and Kruskall's lambda (`gkLambda`) is corrected for labeling every subject at the modal category.

Usage

```
fcKappa(tab, weights = c("None", "Linear", "Quadratic"), W=diag(nrow(tab)))
gkLambda(tab, weights = c("None", "Linear", "Quadratic"), W=diag(nrow(tab)))
accuracy(tab, weights = c("None", "Linear", "Quadratic"), W=diag(nrow(tab)))
```

Arguments

`tab`	A square matrix whose rows and columns represent rating categories from two raters or classifiers and whose cells represent observed (or expected) counts. If one classifier is regarded as "truth" it should be represented by columns.
`weights`	A character scalar which should be one of "None", "Linear", or `Quadratic` which gives the weighting to be used if `W` is not supplied directly (see details).
`W`	A square matrix of the same size as tab giving the weights (see details). If missing and `weights` are supplied one of the standard weights are used.

Details

Lets say the goal is to classify a number of subjects into K categories, and that two raters: Rater 1, and Rater 2 do the classification. (These could be human raters or machine classification algorithms. In particular a Bayes net modal predication is a classier.) Let p_{ij} be the probability that Rater 1 places a subject into Category i and Rater 2 places the same subject into Cateogry j. The $K \times K$ matrix, `tab`, is the *confusion matrix.*

Note that `tab` could be a matrix of probabilities or a matrix of counts, which can be easily turned into a matrix of probabilities by dividing by the total. In the case of a Bayes net, expected counts could be used instead. For example, if Rater 1 was a Bayes net and the predicted probabilities for the three categories was $(.5, .3, .2)$, and and Rater 2 was the true category which for this subject was 1, then that subject would contribute .5 to $p_{1,1}$, .3 to $p_{2,1}$, and .2 to $p_{3,1}$.

In both cases, $\sum p_{kk}$, is a measure of agreement between the two raters. If scaled as probabilities, the highest possible agreement is $+1$ and the lowest 0. This is the `accuracy` function.

However, raw agreement has a problem as a measure of the quality of a rating, it depends on the distributions of the categories in the population of interest. In particular, if a majority of the subject are of one category, then it is very easy to match just by labeling everything as the most frequent category.

The most well-known correct is the Fliess-Cohen kappa (`fcKappa`). This adjusts the agreement rate for the probability that the raters will agree by chance. Let p_{i+} be the row sums, and Let p_{+j}

be the column sums. The probability of a chance agreement, is then $\sum p_{k+}p_{+k}$. So the adjusted agreement is:

$$\kappa = \frac{\sum p_{kk} - \sum p_{k+}p_{+k}}{1 - \sum p_{k+}p_{+k}}.$$

So kappa answers the question how much better do the raters do than chance agreement.

Goodman and Kruskal (1952) offered another way of normalizing. In this case, let Rater 1 be the classificaiton method and Rater 2 be the truth. Now look at a classifier which always classifies somebody in Category k; that classifier will be right with probability p_{+k}. The best such classifer will be $\max p_{+k}$. So the adjusted agreement becomes:

$$\lambda = \frac{\sum p_{kk} - \max p_{+k}}{1 - \max p_{+k}}.$$

Goodman and Kruskal's lambda (`gkLambda`) is appropriate when there is a different treatment associated with each category. In this case, lambda describes how much better one could do than treating every subject as if they were in the modal category.

Weights are used if the misclassification costs are not equal in all cases. If the misclassification cost is c_{ij}, then the weight is defined as $w_{ij} = 1 - c_{ij}/\max c_{ij}$. Weighted agreeement is defined as $\sum\sum w_{ij}p_{ij}$.

If the categories are ordered, there are three fairly standard weighting schemes (especially for kappa).

None $w_{ij} = 1$ if $i = j$, 0 otherwise. (Diagonal matrix.)

Linear $w_{ij} = 1 - |i - j|/(K - 1)$. Penalty increases with number of categories of difference.

Quadratic $w_{ij} = 1 - (i - j)^2/(K - 1)^2$. Penalty increases with square of number of categories of difference.

Indeed, quadratic weighted kappa is something of a standard in comparing two human raters or a single machine classification algorithm to a human rater.

The argument `weights` can be used to select one of these three weighting schemes. Alternatively, the weight matrix `W` can be specified directly.

Value

A real number between -1 and 1; with higher numbers indicating more agreement.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J. Steinberg, L.S., Yan, D. and Willamson, D. M. (2015). *Bayesian Networks in Educational Assessment*. Springer. Chapter 7.

Fleiss, J. L., Levin, B. and Paik, M. C. (2003). *Statistical Methods for Rates and Proportions*. Wiley. Chapter 18.

Goodman, Leo A., Kruskal, William H. (1954). Measures of Association for Cross Classifications. *Journal of the American Statistical Association*. **49** (268), 732–764.

See Also

table

Examples

```
## Example from Almond et al. (2015).
read <- matrix(c(0.207,0.029,0,0.04,0.445,0.025,0,0.025,0.229),3,3,
        dimnames=list(estimated=c("Advanced","Intermediate","Novice"),
                      actual=c("Advanced","Intermediate","Novice")))

stopifnot (abs(fcKappa(read)-.8088) <.001)
stopifnot (abs(gkLambda(read)-.762475) <.001)

fcKappa(read,"Linear")
fcKappa(read,"Quadratic")
gkLambda(read,"Linear")
gkLambda(read,"Quadratic")
```

---

getTableStates *Gets meta data about a conditional probability table.*

Description

Fetches the names of the parent variables, or the names of the states of the child variable from a conditional probability table.

Usage

```
getTableStates(table)
getTableParents(table)
```

Arguments

table	A conditional probability table expressed as a data frame.

Details

These functions assume that table is a conditional probability table (or a set of hyper-Dirichlet parameters) which is shaped like a data frame. Columns in the data frame which are factors are assumed to hold values for the parent (conditioning) variables. Columns in the data frame which are numeric are assumed to correspond to possible states of the child (dependent) variable.

Value

For `getTableParents()`, a character vector giving the names of the parent variables (factor columns).

For `getTableStates()`, a character vector giving the names of the child states (numeric columns).

Note

StatShop usually assumes that the states are ordered from the *highest* to the *lowest* possible values, for example 'High', 'Med', 'Low'.

Author(s)

Russell Almond

See Also

`rescaleTable`, `numericPart`

Examples

```
#conditional table
# Note in R 4.0, parents must be explicitly made into factors
X2.ptf <- data.frame(Theta=factor(c("Expert","Novice")),
                     correct=c(4,2),
                     incorrect=c(2,4))
#Unconditional table
Theta.ptf <- data.frame(Expert=3,Novice=3)

stopifnot(
identical(getTableStates(X2.ptf), c("correct","incorrect")),
identical(getTableStates(Theta.ptf), c("Expert","Novice")),
identical(getTableParents(X2.ptf), "Theta"),
identical(getTableParents(Theta.ptf), character(0))
)
```

---

gkGamma *Calculates the Goodman and Kruskal gamma measure of association.*

Description

Goodman and Kruskal introduce a second coefficient, gamma (`gkGamma`) as a measure of association for two ordered categorical variables which do not necessarily have the same number of states. Thus it is useful as a measure of association between an observed variable and a latent proficiency variable.

Usage

```
gkGamma(tab)
```

Arguments

tab	A matrix whose rows and columns represent rating two different classifications and whose cells represent observed (or expected) counts.

Details

Let A and B be ordinal variables (both oriented in the same direction either high to low or low to high). Then tab is a cross-tabulation of the two variables (e.g., what is produced by the table funciton). In the case that one of the variables is a classification by a Bayesian network, this could also be an expected tabulation. It also could be normalized by dividing by the total to give probabilities.

Let (a, b) and (a', b') be two randomly chosen data points. Classify this pair according to the following rule:

Concordant (same) If both relationships point the same way, that is either both $a > a$‘ and $b > b'$ or both $a < a$‘ and $b < b'$, then the relationship is *concordant*.

Discordant (different) If both relationships point in opposite ways, that is either both $a > a$‘ and $b < b'$ or both $a < a$‘ and $b > b'$, then the relationship is *discordant*.

Discordant (different) If both relationships point in opposite ways, that is either both $a > a$‘ and $b < b'$ or both $a < a$‘ and $b > b'$, then the relationship is *discordant*.

Tied If one or both variables are the same, that is either $a = a$‘ or $b = b'$ or both, then the relationship is *tied*.

Let Π_s be the proportion of pairs which are concordant, Π_d be the proportion of pairs which are discordant, and Π_t be the proportion of pairs which are tied. Then gamma is defined as the proportion of concordant pairs minus the proportion of discordant pairs normalized by the number of non-tied pairs, that is:

$$\gamma = \frac{\Pi_s - \Pi_d}{1 - \Pi_i}$$

Like a correlation coefficient it runs from -1 (perfect negative associations) through 0 (no association) to +1 (perfect association). It is comparable to Kendall's tau (see cor(,method="kendall")).

Value

A numeric scalar. Like a correlation coefficient it runs from -1 (perfect negative associations) through 0 (no association) to +1 (perfect association). It is comparable to Kendall's tau (see cor(,method="kendall")).

Author(s)

Russell Almond

References

Goodman, Leo A., Kruskal, William H. (1954). Measures of Association for Cross Classifications. *Journal of the American Statistical Association*. **49** (268), 732–764.

See Also

Other similar statistics (useful for square tables): fcKappa, gkLambda

Other measures of correlation: cor.

Building contingency tables: table.

Examples

```
## Example from Goodman and Kruskall (1963)
nab <- matrix(c(8,0,0,5,8,4,3,1,14,3,0,4),3,4)
stopifnot(all.equal(.6122,gkGamma(nab),tolerance=.0001))

## Using built-in data set US Judges, rounding to produce ordered
##   categories.

table(round(USJudgeRatings[,1:2]))
gkGamma(table(round(USJudgeRatings[,1:2])))
## Kendall's Tau for comparison
cor(round(USJudgeRatings[,1:2]),method="kendall")
```

---

gradedResponse *A link function based on Samejima's graded response*

Description

This function converts a matrix of effective theta values into a conditional probability table by applying Samejima's graded response model to each row of the table.

Usage

```
gradedResponse(et, linkScale = NULL, obsLevels = NULL)
```

Arguments

et	A matrix of effective theta values. There should be one row in this table for each configuration of the parent variables of the conditional probability table and one column for each state of the child variables except for the last.
linkScale	Unused. For compatibility with other link functions.
obsLevels	A character vector giving the names of the child variable states. If supplied, it should have length ncol(et)+1.

Details

This function takes care of the third step in the algorithm of `calcDPCTable`. Its input is a matrix of effective theta values (comparable to the last column of the output of `eThetaFrame`), one column for each of the child variable states (`obsLevels`) except for the last one. Each row represents a different configuration of the parent variables. The output is the conditional probability table.

Let X be the child variable of the distribution, and assume that it can take on M possible states labeled x_1 through x_M in increasing order. The graded response model defines a set of functions $Z_m(\theta_k)$ for $m = 2, \ldots, M$, where

$$Pr(X >= x_m|\theta_k) = logit^{-1} - D * Z_m(\theta_k)$$

The conditional probabilities for each child state given the effective thetas for the parent variables is then given by

$$Pr(X == x_m|\theta_k)\frac{\sum_{r=1}^{m} Z_r(\theta_k)}{\sum_{r=1}^{M} Z_r(\theta_k)}$$

The $K \times M - 1$ matrix `et` is the values of $Z_m(\theta_k)$. This function then performs the rest of the generalized partial credit model. This is a generalization of Muraki (1992), because the functions $Z_m(\cdot)$ are not restricted to be the same functional form for all m.

If supplied `obsLevels` is used for the column names.

Value

A matrix with one more column than `et` giving the conditional probabilities for each configuration of the parent variables (which correspond to the rows).

Note

The `linkScale` parameter is unused. It is for compatibility with other link function choices.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015). *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Muraki, E. (1992). A Generalized Partial Credit Model: Application of an EM Algorithm. *Applied Psychological Measurement*, **16**, 159-176. DOI: 10.1177/014662169201600206

Samejima, F. (1969) Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph No. 17*, **34**, (No. 4, Part 2).

I also have planned a manuscript that describes these functions in more detail.

See Also

Other Link functions: `gradedResponse`,`normalLink`

Functions which directly use the link function: `eThetaFrame`, `calcDPCTable`, `mapDPC`

Earlier version of the graded response link: `calcDSTable`

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

## Get some effective theta values.
et <- effectiveThetas(3)

gradedResponse(matrix(et,ncol=1),NULL,correctL)

gradedResponse(outer(et,c(Full=1,Partial=-1)),NULL,pcreditL)

gradedResponse(outer(et,c(A=2,B=1,C=0,D=-1)),NULL,gradeL)
```

---

isMonotonic *Tests to see if a sequence is ascending or descending*

Description

These functions take a vector and check to see if it is in increasing or decreasing order. The function `isMonotonic` returns true if the sequence is either increasing (nondecreasing) or decreasing (non-increasing), with the parenthesized values used if `strict` is false. For `isMonotonic` an attribute of the return value tell the direction of the series.

Usage

```
isMonotonic(vec, strict = TRUE)
isIncreasing(vec)
isNondecreasing(vec)
isDecreasing(vec)
isNonincreasing(vec)
```

Arguments

`vec`	A vector of sortable objects (i.e., ones for which the comparison operators are defined. The vector should have length of at least 2.
`strict`	A logical value. If true, then the function will check for strict monotonicity.

Details

These function use the following definitions:

Increasing. `vec[1]` < `vec[2]` < ... < `vec[k]`.

Nondecreasing. `vec[1]` <= `vec[2]` <= ... <= `vec[k]`.

Decreasing. `vec[1] > vec[2] > ... > vec[k]`.

Nonincreasing. `vec[1] >= vec[2] >= ... >= vec[k]`.

A sequence is *monotonic* if it is either nondecreasing or nonincreasing. It is *strictly monotonic* if it is either increasing or decreasing. Note that the function `isMonotonic` by default checks for strict monotonicity; which is usually more useful in the context of Bayesian networks.

It is often of interest to check if the direction is increasing or decreasing. The function `isMonotonic` returns an attribute `"direction"` which is positive for increasing series and negative for decreasing series.

Value

The functions `isIncreasing`, `isNondecreasing`, `isDecreasing`, and `isNonincreasing` a logical according to whether or not the corresponding condition holds for `vec`.

The function `isMonotonic` returns a logical value with an additional attribute `"direction"`. The expression `attr(isMonotonic(vec),"direction"` will be positive if the series is increasing and negative if it is decreasing. If all values are equal, and `strict` is false, the direction will be 0.

If `vec` has length less than 2, then `NA` will be returned.

Note

Because `isMonotonic` returns an extra attribute, `isTRUE(isMonotonic(vec)` will always be false. The expression `isTRUE(c(isMonotonic(vec)))` is likely to be more useful.

Author(s)

Russell Almond

See Also

`sort`, `order`

Examples

```
isIncreasing(1:3)
isNondecreasing(c(1,1,3))
isDecreasing(3:1)
isNonincreasing(c(3,1,1))

isMonotonic(1:3)
isMonotonic(c(1,1,3))  #FALSE
isMonotonic(c(1,1,3),strict=FALSE)
isMonotonic(3:1)
isMonotonic(c(3,3,1),strict=FALSE)
isMonotonic(c(0,0,0)) #FALSE
isMonotonic(c(0,0,0),FALSE) #TRUE
isMonotonic(c(3,0,3)) # FALSE
```

`isOffsetRule` *Distinguishes Offset from ordinary rules.*

Description

An *offset rule* is one where there is one intercept (beta) parameter for each parent and there is a single slope parameters. As opposed to a regression-style rule where there is a different slope (alpha) for each parent and single intercept. This function distinguishes between the twho types of rules.

Usage

```
isOffsetRule(rl)
getOffsetRules()
setOffsetRules(newval)
```

Arguments

`rl`	A character vector of rule names to test to see if these are offset rules.
`newval`	A character vector of rule names to be considered as offset rules.

Details

The `Compensatory` rule acts more or less like a regression, with a slope (or discrimination) parameter for each parent variable, and a single intercept or difficulty parameter. The `Conjunctive` and `Disjunctive` follow the same pattern. In contrast the `OffsetConjunctive` rule, has a different intercept for each parent and a single slope parameter. The `OffsetDisjunctive` rule follows the same pattern.

The `isOffsetRule()` is true if the argument references a function which follows the offset parameterization and false if it follow the regression parameterization. Currently it returns true only for "OffsetConjunctive", and "OffsetDisjunctive", but using this test should continue to work if the number of rules in CPTtools expands.

The expression `getOffsetRules()` returns the list of currently known offset-style rules. The function `setOffsetRules()` allows this list to be manipulated to add a new rule to the list of offset-style rules.

Value

The expression `isOffsetRule(rl)` returns a logical vector of the same length as `rl`, with each element `TRUE` or `FALSE` depending on whether or the corresponding element of `rl` is the name of an offset rule. If `rl` is not a character vector, then the function returns `FALSE`.

The expression `getOffsetRule()` returns the names of the current offset rules.

Note

It makes sense in certain situation to use anonymous function objects as rules, however, it is impossible to test whether or not a function is in the offset rule list. Therefore, it is recommended that only rule names be used with this function.

One consequence of this rule is that when given a function argument, `isOffsetRule` returns `FALSE`.

Author(s)

Russell G. Almond

See Also

`Compensatory`, `OffsetConjunctive`,

Examples

```
stopifnot(
all(isOffsetRule(c("OffsetConjunctive","Conjunctive"))==c(TRUE,FALSE)),
isOffsetRule(OffsetDisjunctive)==TRUE,
all(getOffsetRules() == c("OffsetConjunctive","OffsetDisjunctive"))
)

setOffsetRules(c(getOffsetRules(),"myOffsetRule"))
stopifnot (isOffsetRule("myOffsetRule"))
```

LanguageData *Accuracy and Expected Accuracy Matrixes for Langauge Test.*

Description

This is an example used in Chapter 7 of Almond, et. al. (2015), based on a conceptual assessment described in Mislevy (1995). The assessment has four reporting variables: _Reading_, _Writing_, _Speaking_, and _Listening_, each of which can take on states 'Novice', 'Intermediate', or 'Advanced'. To estimate the reliability 1000 students were simulated from the network (see `NetworkTester` and accuracy, using Maximum A Prior estimates, ('Langauge_modal') and expected accuracy ('Language_exp') estimates were calculated.

Usage

```
data("Language_modal")
data("Language_exp")
```

Format

The format for both 'Language_modal' and 'Language_exp' is: List of 4 $ Reading A 3x3 matrix giving the accuracy of the Reading measure. $ Writing A 3x3 matrix giving the accuracy of the Writing measure. $ Speaking A 3x3 matrix giving the accuracy of the Speaking measure. $ Listening A 3x3 matrix giving the accuracy of the Listening measure.

All cases, the table has been normalized so that all entries sum to 1.

Details

The sample language assessment contains:

A 5 *Reading* tasks which only tap the Reading attribute.

B 3 *Reading/Writing* integrated tasks which require a written response to a textual prompt.

C 3 *Reading/Speaking/Listening* Integrated tasks which require a spoken response to an aural stimulus, with textual instructions.

D 5 *Listening* tasks which have both aural stimulus and instructions.

Because different numbers of tasks (which have different strengths of evidence) are available, different amounts of information are available about each of the four target variables. A number of different measures of reliability can be calculated from the (expected) accuracy matrixes.

Source

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 7.

References

Mislevy, R. J. (1995) Test theory and language learning in assessment. *Language Testing*, *12*, 341–369.

Examples

```
data(Language_modal)
fcKappa(Language_modal$Reading)
gkLambda(Language_modal$Reading)
```

`localDepTest` *Tests for conditional independence between two variables given a third*

Description

The function `ciTest` takes a 3-way contingency table and tests for the conditional independence of the first two variables given the third. The function `localDepTest` is a wrapper which builds the table from factor variables. In psychometrics, when the third variable represents a latent proficiency and the first two item responses, this is sometimes called *local independence*.

Usage

```
localDepTest(obs1, obs2, prof)
ciTest(tab)
```

Arguments

obs1	A factor variable representing the first observable outcome.
obs2	A factor variable representing the second observable outcome.
prof	A factor variable representing the proficiency level, or any variable that is thought to render obs1 and obs2 independent.
tab	A three-way table (see table) where the first two dimensions represent the observable variables and the third the proficiency variable.

Details

Let 1 and 2 represent obs1 and obs2 respectively and let 3 represent prof. In the case of ciTest, 1, 2 and 3 represent the first second and 3rd dimensions of tab. These function then compare the undirected model [13][23] (1 and 2 are conditionally independent given 3) to the unrestricted model [123]. The result is a chi-square statistic comparing the two models, high values of the chi-square statistic indicate a better fit of the unrestricted model compared to the conditional independence model.

Note that the Cochran-Mantel-Haenszel statistic (see mantelhaen.test) is similar, but it assumes that there is no three-way interaction, so it essentially tests [13][23] versus [12][13][23].

Value

A list with three elements:

G2	The chi-square comparison between the two models.
df	The degrees of freedom for the test.
p	The percentage point for G2 in a central chi-square distribution with df degrees of freedom, i.e., the p-value.

Author(s)

Russell Almond

References

Bishop, Feinberg and Holland (1975). *Discrete Multivariate Analysis: Theory and Practice.* MIT Press.

See Also

buildFactorTab, mantelhaen.test, UCBAdmissions

Examples

```
data(UCBAdmissions)
ciTest(UCBAdmissions)

data(ACED)
ACED.items$Correct <- rowSums(ACED.items[,-(1:3)],na.rm=TRUE)
table(ACED.items$tCommonRatio1a,ACED.items$tCommonRatio2a,
      cut(ACED.items$Correct,3))
localDepTest(ACED.items$tCommonRatio1a,ACED.items$tCommonRatio2a,
      cut(ACED.items$Correct,3))
```

---

mapDPC *Finds an MAP estimate for a discrete partial credit CPT*

Description

This finds a set of parameters for a given discrete partial credit model which produces the best fit to the first argument. It is assumed that the first argument is a table produced by adding observed counts to a prior conditional probability table. Thus the result is a maximum a posterior (MAP) estimate of the CPT. The parametric structure of the table is given by the `rules` and `link` parameters.

Usage

```
mapDPC(postTable, skillLevels, obsLevels, lnAlphas, betas,
       rules = "Compensatory", link = "partialCredit",
       linkScale=NULL, Q=TRUE, tvals=lapply(skillLevels,
               function (sl) effectiveThetas(length(sl))),
       ...)
```

Arguments

`postTable`	A table of cell counts which should have the same structure as the output of `calcDPCTable(skillLevels, obsLevels, lnAlphas, betas, rules, link, linkScale)`. As zero counts would cause problems, the prior conditional probability table is normally added to the counts to make the argument a posterior counts.
`skillLevels`	A list of character vectors giving names of levels for each of the condition variables.
`obsLevels`	A character vector giving names of levels for the output variables from highest to lowest. As a special case, can also be a vector of integers.
`lnAlphas`	A list of vectors of initial values for the log slope parameters. Its length should be 1 or `length(obsLevels)-1`. The required length of the individual component vectors depends on the choice of `rule` (and is usually either 1 or the length of `skillLevels`).

betas	A list of vectors of initial values for the difficulty (-intercept) parameters. Its length should be 1 or length(obsLevels)-1. The required length of the individual component vectors depends on the choice of rule (and is usually either 1 or the length of skillLevels).
rules	A list of functions for computing effective theta (see Details). Its length should be length(obsLevels)-1 or 1 (implying that the same rule is applied for every gap.)
link	The function that converts a table of effective thetas to probabilities
linkScale	Initial values for the optional scale parameter for the link function. This is only used with certain choices of link function.
Q	This should be a Q matrix indicating which parent variables are relevant for which state transitions. It should be a number of states minus one by number of parents logical matrix. As a special case, if all variable are used for all levels, then it can be a scalar value.
tvals	A list of the same length as skillLevels. Each element should be a numeric vector values on the theta (logistic) scale corresponding to the levels for that parent variable. The default spaces them equally according to the normal distribution (see effectiveThetas).
...	Additional arguments passed to the optim function.

Details

The purpose of this function is to try to estimate the values of a discrete partial credit model. The structure of the model is given by the rules and link arguments: the form of the table produces is like the output of calcDPCTable(skillLevels, obsLevels, lnAlphas, betas, rules, link, linkScale). It tries to find the values of lnAlphas and betas (and if used linkScale) parameters which are most likely to have generated the data in the postTable argument. The lnAlphas, betas and linkScale arguments provide the initial values for those parameters.

Let $p_{i,j}$ be the value in the ith row and the jth column of the conditional probability table output from calcDPCTable(skillLevels, obsLevels, lnAlphas, betas, rules, link, linkScale), and let $x_{i,j}$ be the corresponding elements of postTable. The mapDPC function uses optim to find the value of the parameters that minimizes the deviance,

$$-2 * \sum_i \sum_j x_{i,j} \log(p_{i,j}).$$

Value

A list with components:

lnAlphas A vector of the same structure as lnAlphas containing the estimated values.

betas A veto of the same structure as betas containing the estimated values.

linkScale If the linkScale was supplied, the estimated value.

convergence An integer code. 0 indicates successful completion, positive values are various error codes (see optim).

value The deviance of the fit DPC model.

The list is the output of the optim) function, which has other components in the output. See the documentation of that function for details.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015). *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Muraki, E. (1992). A Generalized Partial Credit Model: Application of an EM Algorithm. *Applied Psychological Measurement*, **16**, 159-176. DOI: 10.1177/014662169201600206

Samejima, F. (1969) Estimation of latent ability using a response pattern of graded scores. *Psychometrika Monograph No. 17*, **34**, (No. 4, Part 2).

I also have planned a manuscript that describes these functions in more detail.

See Also

optim, calcDPCTable, Compensatory, OffsetConjunctive, gradedResponse, partialCredit

Examples

```
pLevels <- list(Skill1=c("High","Med","Low"))
obsLevels <- c("Full","Partial","None")

trueLnAlphas <- list(log(1),log(.25))
trueBetas <- list(2, -.5)

priorLnAlphas <- list(log(.5),log(.5))
priorBetas <- list(1, -1)

truedist <- calcDPCTable(pLevels,obsLevels,trueLnAlphas,trueBetas,
                         rules="Compensatory",link="partialCredit")
prior <- calcDPCTable(pLevels,obsLevels,priorLnAlphas,priorBetas,
                         rules="Compensatory",link="partialCredit")

post1 <- prior + round(truedist*1000)

map1 <- mapDPC(post1,pLevels,obsLevels,priorLnAlphas,priorBetas,
                         rules="Compensatory",link="partialCredit")

if (map1$convergence != 0) {
  warning("Optimization did not converge:", map1$message)
}

postLnAlphas <- map1$lnAlphas
postBetas <- map1$betas
fitdist <- calcDPCTable(pLevels,obsLevels,map1$lnAlphas,map1$betas,
                         rules="Compensatory",link="partialCredit")
## Tolerance for recovery test.
tol <- .01
maxdistdif <- max(abs(fitdist-truedist))
if (maxdistdif > tol) {
```

```
  stop("Posterior and True CPT differ, maximum difference ",maxdistdif)
}
if (any(abs(unlist(postLnAlphas)-unlist(trueLnAlphas))>tol)) {
  stop("Log(alphas) differ by more than tolerance")
}
if (any(abs(unlist(postBetas)-unlist(trueBetas))>tol)) {
 stop("Betas differ by more than tolerance")
}
```

`MathGrades` *Grades on 5 mathematics tests from Mardia, Kent and Bibby*

Description

Summary statistics for a data set consisting of marks on five mathematics exams originally presented in Mardia, Kent and Bibby (1979). The raw data are not given, but rather the summary statistics reported in Whittaker (1990) are provided.

Usage

```
data(MathGrades)
```

Format

A list consisting of the following components:

varnames Names of the five tests.

means Average score on each test.

var Covariance matrix.

cor Correlation matrix.

icdiag Diagonal of the inverse covariance matrix.

pcor Partial correlation (scaled inverse correlation) matrix.

pvecs A set of marginal distributions for discrete variables corresponding to the five assessments.

Source

Summary statistics reported here are taken from Whittaker (1990).

Original data on 88 students is reported in Mardia, Kent and Bibby (1979).

References

Mardia, K.V. and Kent, J.T. and Bibby, J.M. (1979) *Multivariate Analysis*. Academic Press.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics*. Wiley.

Examples

```
data(MathGrades)

##Note:  Some of these tests may return false due to machine precision
##issues.
round(scaleMatrix(MathGrades$var),2) == MathGrades$cor
round(diag(solve(MathGrades$cor)),2) == MathGrades$icdiag
round(scaleMatrix(solve(MathGrades$cor)),2) == MathGrades$pcor
```

`mcSearch` *Orders variables using Maximum Cardinality search*

Description

Takes a graph described by an incidence matrix, and creates an ordering of the nodes using maximum cardinality search (Tarjan and Yannakakis, 1984). A primary application of this method is to chose an ordering of nodes in an undirected graph to make a corresponding directed graph.

Usage

```
mcSearch(sm, start = colnames(sm)[1])
```

Arguments

`sm`	A logical matrix whose rows and columns correspond to nodes (variables) and a true value indicates an edge between the variables.
`start`	The name of the first element.

Details

The `sm` argument should be an incidence matrix for a graph, with row and column names set to the names of the nodes/variables.

The function returns an ordering of the nodes where each node is chosen so that it has a maximum number of neighbors among those nodes higher in the order.

Ties are broken by chosing the first node in the graph (using the order of the columns) matching the criteria. One special case is the first node which is always an arbitrary choice. The `start` argument can be used to force a particular selection.

Value

A vector of lenght equal to the number of rows whose values correspond to the order of the variable in the final order.

Note

If the graph is triangulated, then the ordering produced by `mcSearch` should be *perfect* — for each node in the order, the set of neighbors of that node which preceed it in the ordering is completely connected. Perfect node orderings are useful in going from undirected to directed graphical representations. If we take an undirected graph and a perfect ordering, a define as the parents of an node all of its neighbors which are previous in the order and define as its children all of the nodes which are later in the order, then when converting the graph back to the undirected form no additional "moralization" edges will be required. Thus, this function can be used to generate orders for `buildParentList`.

Graphical models generally exist only over triangulated graphs. Therefore, and incidence matrix which has been produced through the use of the `structMatrix` function should alway work. Tarjan and Yannakakis (1984) prove that the maximum cardinality search always produces a perfect ordering when the graph is triangulated.

When the graph is not triangulated, the maximum cardinality search algorithm can be used to generate "fill-ins" to triangulate the graph. Lauritzen and Spiegelhalter (1988) note this use. While maximum cardinality search will produce an ordering quickly, the ordering itself has now particular optimality properties as far as the triangulated graph which it creates (Almond, 1995).

Author(s)

Russell Almond

References

Almond, R.G. (1995). *Graphical Belief Modeling*. Chapman and Hall.

Lauritzen, S.L. and D.J. Spiegelhalter (1988). Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion). *Journal of the Royal Statistical Society,Series B*, **50**, 205-247.

Tarjan, R.E. and M. Yannakakis (1984). Simple Linear-Time Algorithms to test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *Siam J. Comput.* **13**, 566-579.

See Also

`structMatrix`, `buildParentList`

Examples

```
data(MathGrades)
MG.struct <- structMatrix(MathGrades$var)

ord <- mcSearch(MG.struct)   # Arbitrary start
orda <- mcSearch(MG.struct, "Algebra") # Put algebra first.

names(sort(orda)) # names of the variables in the chosen order.

# Sort rows and columns of structure matrix by MC order
MG.struct[order(orda), order(orda)]
```

`mutualInformation` *Calculates Mutual Information for a two-way table.*

Description

Calculates the mutual information for a two-way table of observed counts or a joint probability distribution. The mutual information is a measure of association between two random variables.

Usage

```
mutualInformation(table)
```

Arguments

`table`	A two way table or probability distribution. Possibly the output of the `table` command.

Details

This is basically the Kullback-Leibler distance between the joint probability distribution and the probability distribution created by assuming the marginal distributions are independent. This is given in the following formula:

$$I[X;Y] = \sum_x \sum_y \Pr(X = x, Y = y) \log \frac{\Pr(X = x, Y = y)}{\Pr(X = x)\Pr(Y = y)}$$

Note

For square matrixes, the maximum mutual information, which should should correspond to a diagnoal matrix, is `log(d)` where `d` is the number of dimensions.

Author(s)

Russell Almond

References

`https://en.wikipedia.org/wiki/Mutual_information`

Shannon (1948) "A Mathematical Theory of Communication."

See Also

`table`, `CPF`, `ewoe.CPF`, `expTable`

Examples

```
## UCBAdmissions is a three way table, so we need to
## make it a two way table.
mutualInformation(apply(UCBAdmissions,c(1,2),sum))
apply(UCBAdmissions,3,mutualInformation)
apply(UCBAdmissions,2,mutualInformation)
apply(UCBAdmissions,1,mutualInformation)

print(c(mutualInformation(matrix(1,2,2)),0))
print(c(mutualInformation(diag(2)),
        mutualInformation(1-diag(2)), log(2)))
print(c(mutualInformation(diag(3)),log(3)))
```

`normalize` *Normalizes a conditional probability table.*

Description

A conditional probability table (CPT) represents a collection of probability distribution, one for each configuration of the parent variables. This function normalizes the CPT, insuring that the probabilities in each conditional distribution sum to 1.

Usage

```
normalize(cpt)
## S3 method for class 'CPF'
normalize(cpt)
## S3 method for class 'data.frame'
normalize(cpt)
## S3 method for class 'CPA'
normalize(cpt)
## S3 method for class 'array'
normalize(cpt)
## S3 method for class 'matrix'
normalize(cpt)
## S3 method for class 'table'
normalize(cpt)
## Default S3 method:
normalize(cpt)
```

Arguments

`cpt`	A conditional probability table stored in either array (`CPA` format) or data frame (`CPF` format). A general data vector is treated like an unconditional probability vector.

Details

The `normalize` function is a generic function which attempts to normalize a conditional probability distribution.

A conditional probability table in RNetica is represented in one of two ways. In the conditional probability array (`CPA`) the table is represented as a $p+1$ dimensional array. The first p dimensions correspond to configurations of the parent variables and the last dimension the child value. The `normalize.CPA` method adjusts the data value so that the sum across all of the child states is 1. Thus, `apply(result,1:p,sum)` should result in a matrix of 1's. The method `normalize.array` first coerces its argument into a `CPA` and then applies the `normalize.CPA` method.

The second way to represent a conditional probability table in RNetica is to use a data frame (`CPF`). Here the factor variables correspond to a configuration of the parent states, and the numeric columns correspond to states of the child variable. Each row corresponds to a particular configuration of parent variables and the numeric values should sum to one. The `normalize.CPF` function makes sure this constraint holds. The method `normalize.data.frame` first applies `as.CPF()` to make the data frame into a CPF.

The method `normalize.matrix` ensures that the row sums are 1. It does not change the class.

The default method only works for numeric objects. It ensures that the total sum is 1.

`NA`'s are not allowed and will produce a result that is all `NA`s.

Value

An object with similar properties to `cpt`, but adjusted so that probabilities sum to one.

For `normalize.CPA` and `normalize.array` an normalized `CPA` array.

For `normalize.CPF` and `normalize.data.fram` an normalized `CPF` data frame.

For `normalize.matrix` an matrix whose row sums are 1.

For `normalize.default` a numeric vector whose values sum to 1.

Note

May be other functions for CPTs later.

Author(s)

Russell Almond

See Also

`NodeProbs()`

Examples

```
n14 <- normalize(1:4)


normalize(matrix(1:6,2,3))
normalize(array(1:24,c(4,3,2)))
arr <- array(1:24,c(4,3,2),
```

```
              list(a=c("A1","A2","A3","A4"),
                   b=c("B1","B2","B3"),
                   c=c("C1","C2")))
arr <- as.CPA(arr)
normalize(arr)

arf <- as.CPF(arr)
normalize(arf)

df2 <- data.frame(parentval=c("a","b"),
                  prob.true=c(1,1),prob.false=c(1,1))
normalize(df2)


## Admit is the response variable, so force it to be the last dimension.
UCBA1 <- aperm(UCBAdmissions,3:1)
normalize(UCBA1)
```

`normalLink` *Link function using a normal regression.*

Description

This link function assumes that the effective theta for this distribution defines the mean of a normal distribution in a generalized regression model. The link scale parameter describes the residual variance.

Usage

```
normalLink(et, linkScale = NULL, obsLevels = NULL)
```

Arguments

`et`	A matrix of effective theta values. There should be one row in this table for each configuration of the parent variables of the conditional probability table and one column for each state of the child variables except for the last.
`linkScale`	The residual standard deviation parameter. This value must be supplied and should be a positive number; the default value of `NULL` generates an error.
`obsLevels`	An optional character vector giving the names of the child variable states. If supplied, it should have length `ncol(et)+1`.

Details

This function takes care of the third step in the algorithm of `calcDPCTable`. Its input is a matrix of effective theta values (comparable to the last column of the output of `eThetaFrame`), one column for each of the child variable states (`obsLevels`) except for the last one. Each row represents a different configuration of the parent variables. The output is the conditional probability table. The use of this function makes `calcDPCTable` behave like `calcDNTable`.

The idea behind this link function was first proposed in Almond (2010), and it is more completely described in Almond et al. (2015). The motivation comes from assuming that the child variable is created by taking cuts on an underlying continuous variable. The marginal distribution of this variable is a standard normal. The conditional distribution is like a regression prediction with the effective theta from the parent variables (the `et` argument) as the expected value and the `linkScale` parameter as the residual standard deviation.

The calculation works as follows: First cut points are set related to the categories of the child variable. Let `m` be the number of categories (this should be one more than the number of columns of `et`) and the length of `obsLevels` if that is supplied). Then the cut points are set at `cuts <- qnorm(((m - 1):1)/m)`.

Then for each row of the conditional probability table `i`, the probability of being in state `k` is calculated by `pnorm(cuts[k]-et[i, 1])/linkScale) - pnorm(cuts[k-1]-et[i, 1])/linkScale)` with the `pnorm` expression set to 0 or 1 at the endpoints. Note that only the first column of `et` is used in the calculation.

Value

A matrix with one more column than `et` giving the conditional probabilities for each configuration of the parent variables (which correspond to the rows).

Note

The motivation for the normal link function originally came from the observation of odd behavior when variables given a DiBello-Samejima distribution (that is, using the `gradedResponse` link function) were used as parent variables for other variables with a DiBello-Samejima distribution.

One potential reason for the odd behavior was that the graded response link function was not an inverse of the procedure used to assign the `effectiveThetas` to the parent variables. Thus, using a probit link function (`normalLink`) was thought to be better for parent variables than using a logistic link function (`gradedResponse`), at the same time the convention of assigning parent values based on quantiles of the normal distribution started. This made the `normalLink` and `effectiveThetas` approximate inverses (information is still lost through discritization). Note that in the current implementation the scale factor of 1.7 has been added to both the `partialCredit` and `gradedResponse` functions to make the logistic function closer to the normal distribution and a better inverse for the effective theta procedure.

Author(s)

Russell Almond

References

Almond, R. G. (2010). 'I can name that Bayesian network in two matrixes.' *International Journal of Approximate Reasoning.* **51**, 167-178.

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

See Also

Other link functions: partialCredit, gradedResponse.

Functions which directly use the link function: eThetaFrame, calcDPCTable, mapDPC

Earlier version of the graded response link: calcDNTable

Examples

```
skill1l <- c("High","Medium","Low")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

## Get some effective theta values.
et <- effectiveThetas(3)

normalLink(matrix(et,ncol=1),.5,correctL)
normalLink(matrix(et,ncol=1),.3,correctL)
normalLink(matrix(et,nrow=3,ncol=2),.5,pcreditL)
normalLink(matrix(et,nrow=3,ncol=2),.8,pcreditL)

normalLink(matrix(et,nrow=3,ncol=4),.5,gradeL)
normalLink(matrix(et,nrow=3,ncol=4),.25,gradeL)
```

---

numericPart *Splits a mixed data frame into a numeric matrix and a factor part.*

Description

The function numericPart() converts a data.frame to a matrix, by dropping columns which contain non-numeric data. The function factorPart grabs the state information by selecting only columns which are factors.

Usage

```
numericPart(table)
factorPart(table)
```

Arguments

table	A data.frame object.

Details

The primary purpose is to split a conditional probability distribution in data frame format (with a set of factor rows identifying the states of the parent variables) to a table of just the numbers, and a data frame of just the factors so that they can be tackled separately.

Value

A matrix containing just the numeric (or factor) columns of the data frame.

Author(s)

Russell Almond

See Also

data.frame, matrix, data.matrix

Examples

```
name <-c("Shahrazad", "Marguerite")
height <- c(34, 36)
weight <- c(28, 26)
twins <- data.frame(name=as.factor(name),height=height, weight=weight)
numericPart(twins)
factorPart(twins)
```

OCP *Observable Characteristic Plot*

Description

The observable characteristic plot is an analogue of the item characteristic curve for latent class models. Estimates of the class success probability for each latent is plotted. Reference lines are added for various groups which are expected to have roughly the same probability of success.

Usage

```
OCP(x, n, lclabs, pi, pilab = names(pi), lcnames = names(x),
    a = 0.5, b = 0.5, reflty = 1, ..., newplot = TRUE,
    main = NULL, sub = NULL, xlab = "Latent Classes", ylab = "Probability",
    cex = par("cex"), xlim = NULL, ylim = NULL, cex.axis = par("cex.axis"))
OCP2(x, n, lclabs, pi, pilab = names(pi), lcnames = names(x),
     set1 = seq(1, length(x)-1, 2), setlabs = c("Set1", "Set2"),
     setat = -1, a = 0.5, b = 0.5, reflty = 1, ..., newplot = TRUE,
     main = NULL, sub = NULL, xlab = "Latent Classes", ylab = "Probability",
     cex = par("cex"), xlim = NULL, ylim = NULL, cex.axis = par("cex.axis"))
```

Arguments

x	Vector of success counts for each latent class.
n	Vector of of the same size as x of latent class sizes.
lclabs	Character vector of plotting symbols for each latent class
pi	Vector of probability estimates for various levels.

pilab	Character vector of names for each level.
lcnames	Character vector of names for each latent class for axis.
set1	For OCP2, a vector of indexes of the elements of x that form the first set.
setlabs	Character vectors of set labels for OCP2.
setat	Numeric scalar giving the x-coordinate for set labels.
a	The first parameter for beta pseudo-prior, passed to betaci. This should either be a scalar or the same length as x.
b	The second parameter for beta pseudo-prior, passed to betaci. This should either be a scalar or the same length as n.
reflty	Provides the line type (see par(lty)) for reference lines. Should be scalar or same length as pi.
...	Additional graphics parameters passed to plot.window and title.
newplot	Logical. If true, a new plotting window is created. Otherwise, information is added to existing plotting window.
main	Character scalar giving main title (see title).
sub	Character scalar giving sub title (see title).
xlab	Character scalar giving x-axis label (see title).
ylab	Character scalar giving x-axis label (see title).
cex	Scalar giving character expansion for plotting symbols (see par(cex)).
xlim	Plotting limits for x-axis (see plot.window).
ylim	Plotting limits for y-axis (see plot.window).
cex.axis	Scalar giving character expansion for latent class names (axis tick labels; cex values passed to par(axis)).

Details

Most cognitively diagnostic models for assessments assume that students with different patterns of skills will have different patterns of success and failures. The goal of this plot type is to empirically check to see if the various groups conform to their expected probability.

The assumption is that the population is divided into a number of *latent classes* (or groups of latent classes) and that the class label for each member of the class is known. The latent classes are partitioned into *levels*, where each level is assumed to have roughly the same proportion of success. (This is often c("-","+") for negative and positive skill patterns, but there could be multiple levels.)

The key idea of the observable characteristic plot is to compare a credibility interval for the success probability in each latent class, to the modelled success probability given by its level. A beta credibility interval is computed for each latent class using betaci(x,n,a,b), with the expected value set at $(x + a)/(n + a + b)$. The plotting symbols given in lclabs are plotted at the mean (usually, these correspond to the latent class is in), with vertical error bars determined by betaci. Horizontal reference lines are added at the values given by pi. The idea is that if the error bars cross the reference line for the appropriate level, then the latent class fits the model, if not it does not.

The variant OCP2 is the same except that the latent classes are further partitioned into two sets, and the labels for the latent classes for different sets are plotted on different lines. This is particularly useful for testing whether attributes which are not thought to be relevant to a given problem are actually irrelevant.

Value

The output of betaci is returned invisibly.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 10.

Sinharay, S. and Almond, R.G. (2006). Assessing Fit of Cognitively Diagnostic Models: A case study. *Educational and Psychological Measurement*. **67**(2), 239–257.

Sinharay, S., Almond, R. G. and Yan, D. (2004). Assessing fit of models with discrete proficiency variables in educational assessment. ETS Research Report. http://www.ets.org/research/researcher/RR-04-07.html

See Also

betaci

Examples

```
nn <- c(30,15,20,35)
pi <- c("+"=.15,"-"=.85)
grouplabs <- c(rep("-",3),"+")
x <- c("(0,0)"=7,"(0,1)"=4,"(1,0)"=2,"(1,1)"=31)
OCP (x,nn,grouplabs,pi,c("-","+"),ylim=c(0,1), reflty=c(2,4),
     main="Data that fit the model")

x1 <- c("(0,0)"=7,"(0,1)"=4,"(1,0)"=11,"(1,1)"=31)
OCP (x1,nn,grouplabs,pi,c("-","+"),ylim=c(0,1), reflty=c(2,4),
     main="Data that don't fit the model")

nnn <- c("(0,0,0)"=20,"(0,0,1)"=10,
         "(0,1,0)"=10,"(0,1,0)"=5,
         "(1,0,0)"=10,"(1,0,1)"=10,
         "(1,1,1)"=10,"(1,1,1)"=25)
xx <- c("(0,0,0)"=5,"(0,0,1)"=2,
       "(0,1,0)"=2,"(0,1,1)"=2,
       "(1,0,0)"=2,"(1,0,1)"=0,
       "(1,1,0)"=9,"(1,1,1)"=21)
grouplabs1 <- rep(grouplabs,each=2)

OCP2 (xx,nnn,grouplabs1,pi,c("-","+"),ylim=c(0,1), reflty=c(2,4),
     setlabs=c("Low Skill3","High Skill3"),setat=-.8,
     main="Data for which Skill 3 is irrelevant")

xx1 <- c("(0,0,0)"=2,"(0,0,1)"=5,
       "(0,1,0)"=1,"(0,1,1)"=3,
```

```
       "(1,0,0)"=0,"(1,0,1)"=2,
       "(1,1,0)"=5,"(1,1,1)"=24)
OCP2 (xx1,nnn,grouplabs1,pi,c("-","+"),ylim=c(0,1), reflty=c(2,4),
      setlabs=c("Low Skill3","High Skill3"),setat=-.8,
      main="Data for which Skill 3 is relevant")
```

OCP.CPF *Plots a conditional probability distribuiton with data overlayed*

Description

The observable characteristic plot is a plot that compares a conditional probabiltiy distribution with a set of data nominally from that distribution. The exp argument is the conditional distribution in CPF format. This is plotted as a barchart. The obs argument is the data in CPF (rows not normalized). The data are plotted as cumulative probability estimates with error bars overlaid on the barplot. The function OCP2.CPF does the same thing, but contrasts two different data sets.

Usage

```
OCP.CPF(obs, exp, ..., baseCol = "chocolate",
        limits = c(lower = 0.025, upper = 0.975),
        Parents = getTableParents(exp),
        States = getTableStates(exp),
        nstates=length(States),
        key=list(points=FALSE,rectangles=TRUE,text=States),
        par.settings = list(),
        point.pch = paste((nstates-1):0),
        point.col = "black", point.cex = 1.25,
        line.col = "black", line.type=1)
OCP2.CPF(obs1, obs2, exp, ..., baseCol = "chocolate",
        limits = c(lower = 0.025, upper = 0.975),
        Parents = getTableParents(exp),
        States = getTableStates(exp),
        nstates=length(States),
        key=list(points=FALSE,rectangles=TRUE,text=States),
        par.settings = list(),
        point1.pch = paste((nstates-1):0),
        point1.col = "black", point1.cex = 1.25,
        line1.col = "black", line1.type=1,
        point2.pch = paste((nstates-1):0),
        point2.col = "cyan", point2.cex = 1.25,
        line2.col = "cyan", line2.type=2)
```

Arguments

obs, obs1, obs2	A CPF which contains the observed data. The 1 and 2 variants specify the upper and lower plotted data data for OCP2.CPF.
exp	A CPF which contains the candidate distribution.
...	Other arguments passed to barchart.
baseCol	The base color to use for the color gradients in the bars.
limits	A vector of two probabilities representing the quantiles of the beta distribution to be used in error bars.
Parents	A character vector giving the names of the parent variables.
States	A character vector giving the names of the child states.
nstates	An integer giving the nubmer of states.
par.settings	Setting passed to barchart.
key	Settings passed to simpleKey to construct the key.
point.pch, point1.pch, point2.pch	Characters used to plot data points.
point.col, point1.col, point2.col	The colour to be used for the overlay points.
point.cex, point1.cex, point2.cex	The size (cex) to be used for the overlay points.
line.col, line1.col, line2.col	The colour to be used for the overlay lines.
line.type, line1.type, line2.type	The type (lty) for the overlay lines.

Details

The cannonical use for this plot is to test the conditional probability model used in particular Bayesian network. The exp argument is the proposed table. If the parents are fully observed, then the obs argument would be a contingency table with the rows representing parent configurations and the columns the data. If the parents are not fully observed, then obs can be replaced by an expected contingincy table (see Almond, et al, 2015).

The bars are coloured using an intesity scale starting from a base colour. The baseColor argument gives the darkest colour in the scale. The base plot is very similar to barchart.CPF.

Point and interval estimates are made for the cumulative probabilities using a simple Bayesian estimator (this avoids potential issues with 0 cell counts). Basically, the expected values (exp) are added to the data as a prior. The intervals are formued using quantiles of the beta distribution. The limits argument gives the quantiles used.

The point and interval estimaters are plotted over the barchart using a numeric label (going from 0 smallest to $k-1$, were k is the number of states of the child variable) and line running from the lower to upper bound. If the line overlaps the corresponding bar in the barchart, then the data are consistent with the model (at least in that cell).

The parameters point.pch point.col, point.cex, line.col, and line.type control the appearance of the points and lines. These may either be scalars or vectors which match the number of states of the child variable.

The function OCP2.CPF is similar but it meant for situations in which two different data distributions are to be compared. In this one (obs1, point1.*, line1.*) is plotted in the upper half of the bar, and the other (obs2, point2.*, line2.*) in the lower half. Two important applications of this particular variation:

- Two different data sets are related to the presence/absence of an additional parent variable. Thus, this is a test of whether or not the additional parent is relevant.
- Two different data sets represent a focal and reference demographic group. This is a test of whether or not the item behaves similarly for both groups.

Value

Returns an object of class lattice::trellis.object. The default print method will plot this function.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 10.

Sinharay, S. and Almond, R.G. (2006). Assessing Fit of Cognitively Diagnostic Models: A case study. *Educational and Psychological Measurement*. **67**(2), 239–257.

Sinharay, S., Almond, R. G. and Yan, D. (2004). Assessing fit of models with discrete proficiency variables in educational assessment. ETS Research Report. http://www.ets.org/research/researcher/RR-04-07.html

See Also

betaci, OCP, barchart.CPF, cptChi2

Examples

```
skill1l <- c("High","Medium","Low")
skill2l <- c("High","Medium","Low","LowerYet")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

cpfTheta <- calcDPCFrame(list(),skill1l,numeric(),0,rule="Compensatory",
                         link="normalLink",linkScale=.5)

## Compatible data
datTheta <- cpfTheta ## Copy to get the shape
datTheta[1,] <- rmultinom(1,25,cpfTheta[1,])

OCP.CPF(datTheta,cpfTheta)
```

```
## Incompatible data
datThetaX <- cpfTheta ## Copy to get the shape
datThetaX[1,] <- rmultinom(1,100,c(.05,.45,.5))

OCP.CPF(datThetaX,cpfTheta)
OCP2.CPF(datTheta,datThetaX,cpfTheta)

cptComp <- calcDPCFrame(list(S2=skill2l,S1=skill1l),correctL,
                        lnAlphas=log(c(1.2,.8)), betas=0,
                        rule="Compensatory")

cptConj <- calcDPCFrame(list(S2=skill2l,S1=skill1l),correctL,
                        lnAlphas=log(c(1.2,.8)), betas=0,
                        rule="Conjunctive")

datComp <- cptComp
for (i in 1:nrow(datComp))
    ## Each row has a random sample size.
    datComp[i,3:4] <- rmultinom(1,rnbinom(1,mu=15,size=1),cptComp[i,3:4])

datConj <- cptConj
for (i in 1:nrow(datConj))
    ## Each row has a random sample size.
    datConj[i,3:4] <- rmultinom(1,rnbinom(1,mu=15,size=1),cptConj[i,3:4])

## Compatible
OCP.CPF(datConj,cptConj)
## Incompatible
OCP.CPF(datComp,cptConj)
OCP2.CPF(datConj,datComp,cptConj)

cptPC1 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                       lnAlphas=log(1),
                       betas=list(full=c(S1=0,S2=999),partial=c(S2=999,S2=0)),
                       rule="OffsetDisjunctive")
cptPC2 <- calcDPCFrame(list(S1=skill1l,S2=skill2l),pcreditL,
                       lnAlphas=log(1),
                       betas=list(full=c(S1=0,S2=999),partial=c(S2=1,S2=0)),
                       rule="OffsetDisjunctive")


datPC1 <- cptPC1
for (i in 1:nrow(datPC1))
    ## Each row has a random sample size.
    datPC1[i,3:5] <- rmultinom(1,rnbinom(1,mu=25,size=1),cptPC1[i,3:5])

datPC2 <- cptPC2
for (i in 1:nrow(datPC2))
    ## Each row has a random sample size.
    datPC2[i,3:5] <- rmultinom(1,rnbinom(1,mu=25,size=1),cptPC2[i,3:5])

## Compatible
```

```
OCP.CPF(datPC1,cptPC1)
## Incompatible
OCP.CPF(datPC2,cptPC1)
OCP2.CPF(datPC1,datPC2,cptPC1)
```

`OffsetConjunctive` *Conjunctive combination function with one difficulty per parent.*

Description

These functions take a vector of "effective theta" values for a collection of parent variables and calculates the effective theta value for the child variable according to the named rule. Used in calculating DiBello–Samejima and DiBello–Normal probability tables. These versions have a single slope parameter (alpha) and one difficulty parameter per parent variable.

Usage

```
OffsetConjunctive(theta, alpha, betas)
OffsetDisjunctive(theta, alpha, betas)
```

Arguments

`theta`	A matrix of effective theta values whose columns correspond to parent variables and whose rows correspond to possible skill profiles.
`alpha`	A single common discrimination parameter. (Note these function expect discrimination parameters and not log discrimination parameters as used in `calcDSTable`.)
`betas`	A vector of difficulty (-intercept) parameters. Its length should be the same as the number of columns in `theta`.

Details

For `OffsetConjunctive`, the combination function for each row is:

$$alpha * min(theta[1] - betas[1], ..., theta[K] - beta[K])$$

For `OffsetDisjunctive`, the combination function for each row is:

$$alpha * max(theta[1] - betas[1], ..., theta[K] - beta[K])$$

Value

A vector of normal deviates corresponding to the effective theta value. Length is the number of rows of `thetas`.

Note

These functions expect the unlogged discrimination parameters, while `calcDSTable` expect the log of the discrimination parameters. The rationale is that log discrimination is bound away from zero, and hence a more natural space for MCMC algorithms. However, it is poor programming design, as it is liable to catch the unwary.

These functions are meant to be used as structure functions in the DiBello–Samejima and DiBello–Normal models. Other structure functions are possible and can be excepted by those functions as long as they have the same signature as these functions.

Note that the offset conjunctive and disjunctive model don't really make much sense in the no parent case. Use `Compensatory` instead.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015). *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Almond, R.G., DiBello, L., Jenkins, F., Mislevy, R.J., Senturk, D., Steinberg, L.S. and Yan, D. (2001) Models for Conditional Probability Tables in Educational Assessment. *Artificial Intelligence and Statistics 2001* Jaakkola and Richardson (eds)., Morgan Kaufmann, 137–143.

See Also

`effectiveThetas`,`calcDSTable`, `calcDNTable`,`calcDPCTable`, `Compensatory`, `eThetaFrame`

Examples

```
skill <- c("High","Medium","Low")
thetas <- expand.grid(list(S1=seq(1,-1), S2 = seq(1, -1)))
OffsetDisjunctive(thetas, 1.0, c(S1=0.25,S2=-0.25))
OffsetConjunctive(thetas, 1.0, c(S1=0.25,S2=-0.25))
eThetaFrame(list(S1=skill,S2=skill), 1.0, c(S1=0.25,S2=-0.25),
            "OffsetConjunctive")
eThetaFrame(list(S1=skill,S2=skill), 1.0, c(S1=0.25,S2=-0.25),
            "OffsetDisjunctive")
```

`parseProbVec` *Parses Probability Vector Strings*

Description

This takes a bunch of strings of the form `"[High:.3,Med:.5,Low:.2]"` and parses it into a vector `c(High=.3,Med=.5,Low=.2)`.

Usage

```
parseProbVec(pVec)
parseProbVecRow(splitrow)
```

Arguments

pVec	A string of the form "[High:.3,Med:.5,Low:.2]"
splitrow	A collection of strings "High:.3", "Med:.5", "Low:.2".

Details

StatShop outputs marginal distributions in the format [*state0* : *val0* , *state1* : *val1* , . . .]. This function takes a vector of strings containing probability vectors and parses them, returning a matrix of the values, with column names given by the names of the states.

The function parseProbVecRow() is an internal function which parses a single row (after it has been split on the commas).

Value

A matrix containing the values. The rows correspond to the elements of pVec. The columns correspond to the state names.

Author(s)

Russell Almond

References

http://research.ets.org/~ralmond/StatShop/dataFormats.html

See Also

readHistory

Examples

```
parseProbVec(c(Good = "[High:.8,Med:.15,Low:.05]",
               Bad = "[High:.15,Med:.35,Low:.5]",
               Ugly = "[High:.01,Med:.09,Low:.9]"))
```

`partialCredit` *A link function based on the generalized partial credit model*

Description

This function converts a matrix of effective theta values into a conditional probability table by applying the generalized partial credit model to each row of the table.

Usage

```
partialCredit(et, linkScale = NULL, obsLevels = NULL)
```

Arguments

`et`	A matrix of effective theta values. There should be one row in this table for each configuration of the parent variables of the conditional probability table and one column for each state of the child variables except for the last.
`linkScale`	Unused. For compatibility with other link functions.
`obsLevels`	An optional character vector giving the names of the child variable states. If supplied, it should have length `ncol(et)+1`.

Details

This function takes care of the third step in the algorithm of `calcDPCTable`. Its input is a matrix of effective theta values (comparable to the last column of the output of `eThetaFrame`), one column for each of the child variable states (`obsLevels`) except for the last one. Each row represents a different configuration of the parent variables. The output is the conditional probability table.

Let X be the child variable of the distribution, and assume that it can take on M possible states labeled x_1 through x_M in increasing order. The generalized partial credit model defines a set of functions $Z_m(\theta_k)$ for $m = 2, \ldots, M$, where

$$Pr(X >= x_m | X >= x_{m-1}, \theta_k) = logit^{-1} - D * Z_m(\theta_k)$$

The conditional probabilities for each child state is calculated by taking the differences between the curves.

The $K \times M - 1$ matrix `et` is the values of $Z_m(\theta_k)$. This function then performs the rest of the generalized partial credit model. The original Samejima (1969) development assumed that all of the functions $Z_m(\cdot)$ had the same linear form $a(\theta_k - b_m)$, with the b_m strictly increasing (note that in CPTtools, the states are ordered from highest to lowest, so that they should be strictly decreasing). This meant that the curves would never cross. The general notation of `calcDPCTable` does not ensure the curves do not cross, which could result in negative probabilities. This function handles this case by forcing negative probabilities to zero (and adjusting the probabilities for the other state to be properly normalized).

If supplied `obsLevels` is used for the column names.

Value

A matrix with one more column than `et` giving the conditional probabilities for each configuration of the parent variables (which correspond to the rows).

Note

The development here follows Muraki (1992) rather than Samejima (1969).

The `linkScale` parameter is unused. It is for compatibility with other link function choices.

Author(s)

Russell Almond

References

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Yan, D. and Williamson, D.M. (2015). *Bayesian Networks in Educational Assessment.* Springer. Chapter 8.

Muraki, E. (1992). A Generalized Partial Credit Model: Application of an EM Algorithm. *Applied Psychological Measurement*, **16**, 159-176. DOI: 10.1177/014662169201600206

I also have planned a manuscript that describes these functions in more detail.

See Also

Other Link functions: `gradedResponse`,`normalLink`

Functions which directly use the link function: `eThetaFrame`, `calcDPCTable`, `mapDPC`

Examples

```
## Set up variables
skill1l <- c("High","Medium","Low")
correctL <- c("Correct","Incorrect")
pcreditL <- c("Full","Partial","None")
gradeL <- c("A","B","C","D","E")

## Get some effective theta values.
et <- effectiveThetas(3)

partialCredit(matrix(et,ncol=1),NULL,correctL)

partialCredit(outer(et,c(Full=1,Partial=-1)),NULL,pcreditL)

partialCredit(outer(et,c(A=2,B=1,C=0,D=-1)),NULL,gradeL)
```

plotsimplex *Produces a simplex plot of probability vectors.*

Description

A K dimensional probability vector likes on a $(K-1)$-simplex. This simplex can be projected onto 2-dimension, with vertices corresponding to the state of the variable (`simplex_vertex_projection` calculates these co-ordinates). The `simplexplot` function produces the plot.

Usage

```
plotsimplex(data, radius = 1, order = NULL,
  labels_cex = 1, labels = colnames(data), show_labels = TRUE,
  points_col = "#00000044", points_pch = 19, points_cex = 1,
  label_offset = 0.9,
  projection_vertexes = simplex_vertex_projection(ncol(data), radius),
  show_points = TRUE, show_circle = TRUE,
  circle_col = "lightgray", show_edges = TRUE,
  edges_col = "lightgray", ...)
simplex_vertex_projection(nvert, r=1)
```

Arguments

`data`	A data frame or matrix with rows representing individuals, and columns states. The row sums should be one.
`radius`, `r`	The radius of the circle on which to plot.
`order`	Order of the states.
`labels_cex`	Graphical parameter controlling label size.
`labels`	Labels for the vertices.
`show_labels`	Logical, should labels be shown.
`points_col`	Colours for the points.
`points_pch`	Glyph for the points.
`points_cex`	Size for the points.
`label_offset`	Location for the labels.
`projection_vertexes`	A function for computing the vertex locations.
`show_points`	Logical, should the points be plotted.
`show_circle`	Logical, should the reference cicle be plotted.
`circle_col`	Logical, colour for the circle.
`show_edges`	Logical, should the edges be shown.
`edges_col`	The colour for the edges.
`...`	extra arguments are ignored.
`nvert`	The number of vertices.

Details

This function is adapted from the `archetypes::simplexplot` function. As the probability vector is already a simplex, we don't need the archetypes overhead.

Value

The function `simplex_vertex_projection` returns a matrix giving the co-ordinates of the vertices.

The function `simplexplot` is normally called for its side-effects, however, it invisibly returns a data structure containing plot details.

Author(s)

Russell Almond

References

See Section 6 in "Probabilistic Archetypal Analysis" by Seth and Eugster (2014), http://arxiv.org/abs/1312.7604.

This is a simplified version of the code in the archetypes package. https://cran.r-project.org/package=archetypes

See Also

`pvecTable`

Examples

```
data(ACED)
ptab <- pvecTable(ACED.scores,"sgp")
plotsimplex(ptab)
```

---

proflevelci *Produce cumulative sum credibility intervals*

Description

Produces credibility intervals for hanging barplots. Assumes that each column represents a sum of proportions and produces corresponding intervals for the cumulative sums. Values hanging below and above the reference line are treated separately, and returned values below the reference are negative.

Usage

```
proflevelci(data, profindex, limits=list(lower=.025,upper=.975),a=.5, b=.5)
```

Arguments

data	A matrix of data values where each column refers to a bar in barplot. The values should be scaled so that the column sum is the number of individuals in that group.
profindex	The level in the chart which corresponds to the reference (proficiency) line. This should be a positive integer less than nrow(data).
limits	The upper and lower credibility limits.
a	Value for the shape1 parameter of the beta prior.
b	Value for the shape2 parameter of the beta prior.

Details

For a stacked bar plot, the natural comparisons involve not category probabilities but the sum of the category probabilities up through the current bar. For hanging bar plots, this should go in both directions. So for example, if the categories are "Below Basic", "Basic", "Proficient", and "Advanced", and the zero line is to be put between "Basic" and "Proficient", then we need credibility intervals for Pr("Basic" or "Below Basic"), Pr("Basic"), Pr("Proficient"), Pr("Proficient" or "Advanced").

The proflevelci function splits the states up into those above the line and below the line using profindex. It then generates credibility intervals using betaci for the cumulative sums in each group. The primary purpose is to create confidence intervals for stacked bar charts (see compareBars2).

Value

A list of data sets of the same length as the limits argument. Each data set has the same shape as the data argument and represents a quantile of the data associated with the value in limits. With the default limits of lower and upper, the result is a list of two elements

lower	Gives the lower bounds of the confidence interval.
upper	Gives the upper bounds of the confidence interval.

Author(s)

Russell Almond

See Also

betaci, compareBars2

Examples

```
margins <- data.frame (
 Trouble=c(Novice=19,Semester1=24,Semester2=28,Semseter3=20,Semester4=9),
 NDK=c(Novice=1,Semester1=9,Semester2=35,Semseter3=41,Semester4=14),
 Model=c(Novice=19,Semester1=28,Semester2=31,Semseter3=18,Semester4=4)
)

proflevelci(margins,3,limits=c(lower=.025,upper=.975))
```

readHistory *Reads a file of histories of marginal distributions.*

Description

In running a typical Bayes net engine, as each piece of evidence comes in, updated marginal distributions for several variables are output. This function reads a such a log, expressed as a data frame, and creates a data structure suitable for doing weight of evidence analyses. The probabilities are in the pvec (parseProbVec) format.

Usage

```
readHistory(histdat, obscol = "Item", valcol = "Result",
  probcol="Probability")
```

Arguments

histdat	A data frame which has columns corresponding to obscol, valcol and probcol
.	
obscol	Name of the column with the name of the observable at each time point.
valcol	Name of the column with the value of the observable at each time point.
probcol	Name of the column with the probabity vectors.

Details

The assumption is that the histdat data frame contains a history of measurements on a latent variable. The *probcol* column should contain a probability vector of the form: [High:0.527,Medium:0.447,Low:0.025]. This function parses this column (see parseProbVec) and builds a matrix with columns corresponding to the states of the latent variable.

The rows are given names of the form, <obscol>=<valcol>, where <obscol> and <valcol> are the values in the respective columns.

Value

A matrix whose column names are taken from the probability vectors and row names are taken from the obscol and valcol fields.

Note

The previous version (a) directly read the CSV file, and (b) had the names of the columns hard coded. This version will break any code that relied on the old version. However, it is hopefully more generally useful.

Author(s)

Russell Almond

References

http://research.ets.org/~ralmond/StatShop/dataFormats.html

See Also

parseProbVec, woeHist

Examples

```
testFiles <- system.file("testFiles",package="CPTtools")
allcorrect <- readHistory(read.csv(file.path(testFiles,
   "CorrectSequence.csv"),as.is=TRUE),
   probcol="Margin.sequences.")
woeHist(allcorrect,"High",c("Medium","Low"))
allincorrect <- readHistory(read.csv(file.path(testFiles,
  "InCorrectSequence.csv"),as.is=TRUE),
   probcol="Margin.sequences.")
```

---

rescaleTable *Rescales the numeric part of the table*

Description

Takes a table representing a conditional probability distribution or a set of hyper-Dirichlet parameters and rescales the numeric part of the table. The function `rescaleTable()` scales the table by `scaleFactor`, the function `normalizeTable()` scales the function by the sum of the rows, making the result a conditional probability table.

Usage

```
rescaleTable(table, scaleFactor)
normalizeTable(table)
```

Arguments

`table`	A data frame describing a conditional probability table. Assumes that the conditions are expressed as factor variables, and all numeric columns represent states of the child variable.
`scaleFactor`	A scalar or vector of length equal to the number of rows of `table`.

Details

For `rescaleTable()`, every numeric column of `table` is multiplied by `scaleFactor`. This can be used to create a set of hyper-Dirichlet parameters by multiplying a conditional probability table by the effective sample size.

For `normalizeTable()`, the `scaleFactor` is set to be `1/rowSums(table)` (excluding the factor variables) so that the resulting table is a proper conditional probability table.

Value

A data frame of the same shape as `table` with the numeric entries suitably scaled.

Note

The function `scaleTable` does a similar rescaling, only it works with a separate 'Sum' and 'Scale' columns in the table.

Author(s)

Russell Almond

See Also

`getTableStates`, `scaleTable`

Examples

```
#conditional table
X2.ptf <- data.frame(Theta=c("Expert","Novice"),
                     correct=c(4,2),
                     incorrect=c(2,4))

X2.t99 <- rescaleTable(X2.ptf,99/6)  #Reweight to effective samples size of 99
X2.t31 <- rescaleTable(X2.ptf,c(3,1))  #Weight expert prior 3 times more than
                            #novice prior.
X2.dtf <- normalizeTable(X2.ptf)


#Unconditional table
Theta.ptf <- data.frame(Expert=3,Novice=3)
Theta.t100 <- rescaleTable(Theta.ptf,100/6)  #Reweight to effective
                                             #sample size of 100
Theta.dtf <- normalizeTable(Theta.ptf)
```

scaleMatrix *Scales a matrix to have a unit diagonal*

Description

Creates a correlation matrix from a covariance matrix by scaling rows and columns to have a unit diagonal. Also can be used to create a partial correlation matrix from an inverse covariance/correlation matrix.

Usage

```
scaleMatrix(X)
```

Arguments

X	A square, positive definite matrix (covariance matrix).

Details

Divides rows and columns by square root of the diagonal elements.

Value

A matrix of the same size and shape as the original with a unit diagonal.

Author(s)

Russell Almond

Examples

```
data(MathGrades)

## Create a correlation matrix from a covariance matrix.
round(scaleMatrix(MathGrades$var),2) == MathGrades$cor

## Create a partial correlation matrix from a correlation matrix
round(scaleMatrix(solve(MathGrades$cor)),2) == MathGrades$pcor
##Note:  Some of these tests may return false due to machine precision
##issues.
```

scaleTable *Scales a table according to the Sum and Scale column.*

Description

Takes a matrix or vector with a `Sum` and `Scale` column and rescales it by multiplying each remaining element by the value of $Scale/Sum$ for that row.

If the last two rows are not named `Sum` and `Scale` then it simply returns its argument.

Usage

```
scaleTable(table)
```

Arguments

`table`	A matrix or vector in which the last two columns are named "Scale" and "Sum".

Details

The parameters of a Dirichlet distribution can be stored in two ways, one is to have each cell in the table represent a pseudo count. The other was is to have each row represent a probability vector and use an additional pseudo sample size (the `Scale` column). If the probability vector is reported in a some other metric (say as a percentage or as a fraction of some smaller sample) the the `Sum` column is used to store the row sum.

Value

Rescaled table with `Sum` and `Scale` columns removed. This makes some attempt to preserve the type of the `table` argument as a matrix, row vector or numeric object.

Note

Used by the function `compareDS` to compare tables which may be in different formats.

Author(s)

Russell Almond

References

http://research.ets.org/~ralmond/StatShop/dataFormats.html

Examples

```
c1 <- matrix(c(70,20,10,10,20,70),nrow=2,byrow=TRUE,
             dimnames=list(NULL,c("H","M","L")))
s1 <- matrix(c(7,2,1,10,100,1,2,7,10,100),nrow=2,byrow=TRUE,
             dimnames=list(NULL,c("H","M","L","Sum","Scale")))

## 1 row matrixes need special handling (c1[1,] is a vector not a matrix)
c1r1 <- matrix(c1[1,],nrow=1,dimnames=list(NULL,c("H","M","L")))
s1r1 <- matrix(s1[1,],nrow=1,dimnames=list(NULL,c("H","M","L","Sum","Scale")))

stopifnot(
  identical(c1,scaleTable(s1)),
  identical(c1[1,],scaleTable(s1[1,])),
  identical(c1r1,scaleTable(s1r1))
)

# This should have no effect when run on matrixes without the Sum and
# Scale column.
stopifnot(
  identical(c1,scaleTable(c1)),
  identical(c1[1,],scaleTable(c1[1,])),
  identical(c1r1,scaleTable(c1r1))
)
```

---

stackedBarplot *Produces a hanging barplot*

Description

This produces a series of stacked bar plots staggered so that the baseline corresponds to a particular state level. This is primarily designed for producing plots of probability vectors coming out of Bayes net scoring.

Usage

```
stackedBarplot(height, width = 1, space = 0.2, offset = 0, names.arg = NULL,
               legend.text = NULL, horiz = FALSE, density = NULL,
               angle = 45, col = NULL, border = par("fg"),
               main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
               xlim = NULL, ylim = NULL, xpd = TRUE, axis = TRUE,
               axisnames = TRUE, cex.axis = par("cex.axis"),
               cex.names = par("cex.axis"), newplot = TRUE,
               axis.lty = 0, ...)
```

Arguments

height	A matrix giving the heights of the bars. The columns represent bars, and the rows represent groups within each bar.

width	A numeric vector of length equal to the number of columns in height giving the width of the bars. If value is shorter than number of columns in height it is recycled to produce the correct length.
space	A numeric vector of length equal to the number of columns in height giving the space between the bars. If value is shorter than number of columns in height it is recycled to produce the correct length.
offset	A numeric vector of length equal to the number of columns in height giving distance by which the bars should be offset from the zero line on the axis. Setting this to a non-zero value produces a hanging barplot.
names.arg	If not missing, used as axis labels (see axis).
legend.text	If node null, a legend is generated (see legend).
horiz	A logical value. If true, stacked bars are printed horizontally instead of vertically.
density	Density of shading lines (see rect). This should be a scalar or a vector of length equal to the number of rows of height.
angle	Angle of shading lines (see rect). This should be a scalar or a vector of length equal to the number of rows of height.
col	Color used for each bar. This should be a scalar or a vector of length equal to the number of rows of height. If not supplied a grayscale gradient is built.
border	Color for the rectangle borders (see rect). This should be a scalar or a vector of length equal to the number of rows of height.
main	Main title for plot (see title).
sub	Subtitle for plot (see title).
xlab	X-axis label for plot (see title).
ylab	Y-axis label for plot (see title).
xlim	Limits in user co-ordinates for the x-axis. Should be a vector of length 2.
ylim	Limits in user co-ordinates for the y-axis. Should be a vector of length 2.
xpd	A logical value controlling clipping. (see par).
axis	A logical value. If true, a numeric scale is printed on the appropriate axis.
axisnames	A logical value. If true, column names are printed on the appropriate axis.
cex.axis	Character size used for the numeric axis labels (see axis).
cex.names	Character size used for the text (column names) axis labels (see axis).
newplot	A logical value. If true a new graphics region is created. If false, the plot is placed on top of the existing graphics region.
axis.lty	A value passed as the lty argument to axis when plotting the text (column name) axis.
...	Other graphics parameters passed to rect, axis and title.

Details

This is a more detailed version of the stackedBars graph which allows finer control. It is used mainly by compareBars.

There are some differences from stackedBars. First, height can be any value, not just a vector of probability. Second, offset is given as a numeric value in the units of height, rather than as an index into the array of heights. Most of the rest of the arguments merely expose the graphical arguments to the user.

Value

The midpoints of the bars are returned invisibly.

Author(s)

Russell Almond

See Also

compareBars, colorspread, buildMarginTab, marginTab, barplot,stackedBars

Examples

```
margins <- data.frame (
 Trouble=c(Novice=.19,Semester1=.24,Semester2=.28,Semseter3=.20,Semester4=.09),
 NDK=c(Novice=.01,Semester1=.09,Semester2=.35,Semseter3=.41,Semester4=.14),
 Model=c(Novice=.19,Semester1=.28,Semester2=.31,Semseter3=.18,Semester4=.04)
)
margins <- as.matrix(margins)
baseline <- apply(margins[1:2,],2,sum)

stackedBarplot(margins,offset=-baseline,
            main="Marginal Distributions for NetPASS skills",
            sub="Baseline at 2nd Semester level.",
            col=hsv(223/360,.2,0.10*(5:1)+.5))
```

---

stackedBars *Produces a stacked, staggered barplot*

Description

This produces a series of stacked bar plots staggered so that the baseline corresponds to a particular state level. This is primarily designed for producing plots of probability vectors coming out of Bayes net scoring.

Usage

```
stackedBars(data, profindex, ...,
            ylim = c(min(offsets) - 0.25, max(1 + offsets)),
            cex.names = par("cex.axis"), percent=TRUE,
            digits = 2*(1-percent), labrot=FALSE)
```

Arguments

data	A data.frame where each column is a probability vector.
profindex	The index of the proficiency which should be used as a baseline.
...	Graphical arguments passed to barplot.
ylim	Default limits for Y axis.
cex.names	Magnification for names.
percent	Logical value. If true data values are treated as percentages instead of probabilities.
digits	Number of digits for overlaid numeric variables.
labrot	If true, labels are rotated 90 degrees.

Details

This plot type assumes that each column in its first argument is a probability vector. It then produces a stacked bar for each column. The baseline of the bar is offset by the probability for being in the category marked by profindex or below.

The probability values are overlaid on the bars.

Value

Returns the midpoints of the bars invisibly.

Author(s)

Russell Almond

References

This plot type was initially developed in Jody Underwood's Evolve project.

Almond, R. G., Shute, V. J., Underwood, J. S., and Zapata-Rivera, J.-D (2009). Bayesian Networks: A Teacher's View. *International Journal of Approximate Reasoning.* **50**, 450-460.

See Also

compareBars, colorspread, buildMarginTab, marginTab, barplot,stackedBarplot

Examples

```
margins <- data.frame (
 Trouble=c(Novice=.19,Semester1=.24,Semester2=.28,Semseter3=.20,Semester4=.09),
 NDK=c(Novice=.01,Semester1=.09,Semester2=.35,Semseter3=.41,Semester4=.14),
 Model=c(Novice=.19,Semester1=.28,Semester2=.31,Semseter3=.18,Semester4=.04)
)

stackedBars(margins,3,
            main="Marginal Distributions for NetPASS skills",
            sub="Baseline at 3rd Semester level.",
            cex.names=.75, col=hsv(223/360,.2,0.10*(5:1)+.5))

stackedBars(margins,3,
            main="Marginal Distributions for NetPASS skills",
            sub="Baseline at 3rd Semester level.",
            percent=FALSE,digits=2,
            cex.names=.75, col=hsv(223/360,.2,0.10*(5:1)+.5))
```

---

structMatrix *Finds graphical structure from a covariance matrix*

Description

This function finds an undirected graphical representation of a multivariate normal distribution with the given covariance matrix, by associating edges with non-zero entries. Graphical structure is given as an adjacency matrix.

Usage

```
structMatrix(X, threshold = 0.1)
```

Arguments

X	A variance matrix.
threshold	A numeric value giving the threshold for a value to be considered "non-zero".

Details

For a multivariate normal model, zero entries in the inverse covariance matrix correspond to conditional independence statements true in the multivariate normal distribution (Whitaker, 1990; Dempster, 1972). Thus, every non-zero entry in the inverse correlation matrix corresponds to an edge in an undirected graphical model for the structure.

The threshold parameter is used to determine how close to zero a value must be to be considered zero. This allows for both estimation error and numerical precision when inverting the covariance matrix.

Value

An adjacency matrix of the same size and shape as `X`. In this matrix `result[i,j]` is `TRUE` if and only if Node i and Node j are neighbors in the graph.

Note

Models of this kind are known as "Covariance Selection Models" and were first studied by Dempster (1972).

Author(s)

Russell Almond

References

Dempster, A.P. (1972) Covariance Selection. *Biometrics*, **28**, 157–175.

Whittaker, J. (1990). *Graphical Models in Applied Multivariate Statistics*. Wiley.

See Also

`scaleMatrix`, `mcSearch`, `buildParentList`

Examples

```
data(MathGrades)

MG.struct <- structMatrix(MathGrades$var)
```

`woeBal` *Weight of Evidence Balance Sheet*

Description

Creates a weight of evidence balance sheet from a history of marginal distributions.

Usage

```
woeBal(hist, pos, neg, obs=NULL, title = "Evidence Balance Sheet",
       col = rev(colorspread("slategray",ncol(hist),maxsat=TRUE)),
       posCol="cyan", negCol="red", stripCol=c("white","lightgray"),
       lcex = 0.65)
```

Arguments

hist	A matrix whose rows represent time points (after tests) and columns represent probabilities.
pos	An expression for selecting rows of the cpf which corresponds to the hypothesis.
neg	An expression for selecting the rows corresponding to the complement of the hypothesis. (The default value is -pos if pos is numeric; !pos if pos is logical, and setdiff(rownames(cpf),pos if pos is a character vector.
obs	An optional character vector of the same length as the number of rows of hist giving the value observed at each step
title	Title for plot
col	A list of color values for probability bars.
posCol	The color to be used for bars showing positive weights of evidence.
negCol	The color to be used for bars showing negative weights of evidence.
stripCol	The colors to be used for the time step labels. Setting this to a vector of two colors creates alternate color stripes. Set this to "white" to disable that effect.
lcex	Character expansion size for labels.

Details

This constructs a weight of evidence balance sheet (Madigan, Mosurski, and Almond, 1997) showing the changes to the probability distribution and weight of evidence for each change in the probability. The probabilities are given in the hist argument in which each row should be a probability distribution for the target variable. The labels for the plot are taken from the row labels of the hist argument.

Madigan, Mosurski and Almond (1997) note that the definition of weight of evidence is somewhat problematic if the hypothesis variable is not binary. In that case, they recommend partitioning the states into a *positive* and *negative* set. The pos and neg are meant to describe that partition. They can be any expression suitable for selecting columns from the hist matrix. This function calls woeHist() to calculate weights of evidence.

The row names of hist are printed left-justified in the leftmost column. If observed values (obs) are supplied, they are printed right justified in the same column.

Value

The midpoints of the bars (see barplot) are returned invisibly.

Side Effects

Starts a new plotting page and creates three side-by-side plots, one for the labels, one for the probability bars and one for the weight of evidence bars.

Author(s)

Russell Almond

References

Good, I. (1971) The probabilistic explication of information, evidence, surprise, causality, explanation and utility. In *Proceedings of a Symposium on the Foundations of Statistical Inference*. Holt, Rinehart and Winston, 108-141.

Madigan, D., Mosurski, K. and Almond, R. (1997) Graphical explanation in belief networks. *Journal of Computational Graphics and Statistics*, **6**, 160-181.

Almond, R. G., Kim, Y. J., Shute, V. J. and Ventura, M. (2013). Debugging the Evidence Chain. In Almond, R. G. and Mengshoel, O. (Eds.) *Proceedings of the 2013 UAI Application Workshops: Big Data meet Complex Models and Models for Spatial, Temporal and Network Data (UAI2013AW)*, 1-10. http://ceur-ws.org/Vol-1024/paper-01.pdf

Almond, R.G., Mislevy, R.J., Steinberg, L.S., Williamson, D.M. and Yan, D. (2015) *Bayesian Networks in Educational Assessment.* Springer. Chapter 7.

See Also

readHistory, woeHist, barplot, Colors

Examples

```
sampleSequence <- read.csv(system.file("testFiles","SampleStudent.csv",
                                       package="CPTtools"),
                           header=TRUE,row.names=1)

woeBal(sampleSequence[,c("H","M","L")],c("H"),c("M","L"),lcex=1.25)
woeBal(sampleSequence[,c("H","M","L")],c("H"),c("M","L"),
      obs=sampleSequence[,"Acc"],lcex=1.25)
```

---

woeHist *Creates weights of evidence from a history matrix.*

Description

Takes a matrix providing the probability distribution for the target variable at several time points and returns a weight of evidence for all time points except the first.

Usage

```
woeHist(hist, pos=1L, neg=NULL)
```

Arguments

hist	A matrix whose rows represent time points (after tests) and columns represent probabilities.
pos	An expression for selecting rows of the cpf which corresponds to the hypothesis.

neg — An expression for selecting the rows corresponding to the complement of the hypothesis. (The default value is `-pos` if `pos` is numeric; `!pos` if `pos` is logical, and `setdiff(rownames(cpf),pos` if `pos` is a character vector.

Details

Good (1971) defines the *Weight Of Evidence (WOE)* as:

$$100 \log_{10} \frac{\Pr(E|H)}{\Pr(E|\overline{H})} = 100 \left[\log_{10} \frac{\Pr(H|E)}{\Pr(\overline{H}|E)} - \log_{10} \frac{\Pr(H)}{\Pr(\overline{H})} \right]$$

Where $\overline{H}$ is used to indicate the negation of the hypothesis. Good recommends taking the log base 10 and multiplying by 100, and calls the resulting units *centibans*. The second definition of weight of evidence as a difference in log odd leads naturally to the idea of an incremental weight of evidence for each new observation.

Following Madigan, Mosurski and Almond (1997), all that is needed to calculate the WOE is the marginal distribution for the hypothesis variable at each time point. They also note that the definition is somewhat problematic if the hypothesis variable is not binary. In that case, they recommend partitioning the states into a *positive* and *negative* set. The `pos` and `neg` are meant to describe that partition. They can be any expression suitable for selecting columns from the `hist` matrix.

Value

A vector of weights of evidence of length one less than the number of rows of `hist` (i.e., the result of applying `diff()` to the vector of log odds.)

Author(s)

Russell Almond

References

Good, I. (1971) The probabilistic explication of information, evidence, surprise, causality, explanation and utility. In *Proceedings of a Symposium on the Foundations of Statistical Inference*. Holt, Rinehart and Winston, 108-141.

Madigan, D., Mosurski, K. and Almond, R. (1997) Graphical explanation in belief networks. *Journal of Computational Graphics and Statistics*, **6**, 160-181.

See Also

`readHistory`, `woeBal`, `diff`

Examples

```
testFiles <- system.file("testFiles",package="CPTtools")
allcorrect <- readHistory(read.csv(file.path(testFiles,
   "CorrectSequence.csv"),as.is=TRUE),
   probcol="Margin.sequences.")
woeHist(allcorrect,"High",c("Medium","Low"))
woeHist(allcorrect,1:2,3)
```

Index