Package 'cpfa'

January 15, 2026

Type Package

Title Classification with Parallel Factor Analysis

Version 1.2-5

Date 2026-01-15

Maintainer Matthew A. Asisgress <mattgress@protonmail.ch>

Depends multiway

Imports glmnet, e1071, randomForest, nnet, rda, xgboost, foreach, doParallel, doRNG

Suggests knitr, rmarkdown

Description Classification using Richard A. Harshman's Parallel Factor Analysis-1 (Parafac) model or Parallel Factor Analysis-2 (Parafac2) model fit to a three-way or four-way data array. See Harshman and Lundy (1994): <doi:10.1016/0167-9473(94)90132-5>. Uses component weights from one mode of a Parafac or Parafac2 model as features to tune parameters for one or more classification methods via a k-fold cross-validation procedure. Allows for constraints on different tensor modes. Supports penalized logistic regression, support vector machine, random forest, feed-forward neural network, regularized discriminant analysis, and gradient boosting machine. Supports binary and multiclass classification. Predicts class labels or class probabilities and calculates multiple classification performance measures. Implements parallel computing via the 'parallel', 'doParallel', and 'doRNG' packages.

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Matthew A. Asisgress [aut, cre]

Repository CRAN

Date/Publication 2026-01-15 03:40:02 UTC

Contents

`cpfa` *Classification with Parallel Factor Analysis*

Description

Fits Richard A. Harshman's Parallel Factor Analysis-1 (Parafac) model or Parallel Factor Analysis-2 (Parafac2) model to a three-way or four-way data array. Allows for different constraint options on multiple tensor modes. Uses Parafac component weights from a single mode of this model as predictors to tune parameters for one or more classification methods via a k-fold cross-validation procedure. Predicts class labels and calculates multiple performance measures for binary or multi-class classification across multiple replications with different train-test splits. Provides descriptive statistics to pool output across replications.

Usage

```
cpfa(x, y, model = c("parafac", "parafac2"), nfac = 1, nrep = 5, ratio = 0.8,
     nfolds = 10, method = c("PLR", "SVM", "RF", "NN", "RDA", "GBM"),
     family = c("binomial", "multinomial"), parameters = list(),
     type.out = c("measures", "descriptives"), foldid = NULL,
     prior = NULL, cmode = NULL, seeds = NULL, plot.out = FALSE,
     plot.measures = NULL, parallel = FALSE, cl = NULL, verbose = TRUE, ...)
```

Arguments

`x`	A three-way or four-way data array. For Parafac2, can be a list where each element is a matrix or three-way array. Array or list must contain only real numbers. See note below.
`y`	A vector containing at least two unique class labels. Should be a factor that contains two or more levels. For binary case, ensure the order of factor levels (left to right) is such that negative class is first and positive class is second.
`model`	Character designating the Parafac model to use, either `model = "parafac"` to fit the Parafac model or `model = "parafac2"` to fit the Parafac2 model.
`nfac`	Number of components for each Parafac or Parafac2 model to fit. Default is `nfac = 1`.
`nrep`	Number of replications to repeat the procedure. Default is `nrep = 5`.
`ratio`	Split ratio for dividing data into train and test sets. Default is `ratio = 0.8`.
`nfolds`	Numeric value specifying number of folds for k-fold cross-validation. Must be 2 or greater. Default is `nfolds = 10`.

`method` Character vector indicating classification methods to use. Possible methods include penalized logistic regression (PLR); support vector machine (SVM); random forest (RF); feed-forward neural network (NN); regularized discriminant analysis (RDA); and gradient boosting machine (GBM). If none are selected, default is to use all methods with `method = c("PLR", "SVM", "RF", "NN", "RDA", "GBM")`.

`family` Character value specifying binary classification (`family = "binomial"`) or multiclass classification (`family = "multinomial"`). If not provided, number of levels of input `y` is used, where two levels is binary, and where three or more levels is multiclass.

`parameters` List containing arguments related to classification methods. When specified, must contain one or more of the following:

alpha Values for penalized logistic regression alpha parameter; default is `alpha = seq(0, 1, length = 6)`. Must be numeric and contain only real numbers between 0 and 1, inclusive.

lambda Optional user-supplied lambda sequence for `cv.glmnet` for penalized logistic regression. Default is NULL.

cost Values for support vector machine cost parameter; default is `cost = c(1, 2, 4, 8, 16, 32, 64)`. Must be numeric and contain only real numbers greater than 0.

gamma Values for support vector machine gamma parameter; default is `gamma = c(0, 0.01, 0.1, 1, 10, 100, 1000)`. Must be numeric and greater than or equal to 0.

ntree Values for random forest number of trees parameter; default is `ntree = c(100, 200, 400, 600, 800, 1600, 3200)`. Must be numeric and contain only integers greater than or equal to 1.

nodesize Values for random forest node size parameter; default is `nodesize = c(1, 2, 4, 8, 16, 32, 64)`. Must be numeric and contain only integers greater than or equal to 1.

size Values for neural network size parameter; default is `size = c(1, 2, 4, 8, 16, 32, 64)`. Must be numeric and contain only integers greater than or equal to 0.

decay Values for neural network decay parameter; default is `decay = c(0.001, 0.01, 0.1, 1, 2, 4, 8, 16)`. Must be numeric and contain only real numbers.

rda.alpha Values for regularized discriminant analysis alpha parameter; default is `rda.alpha = seq(0, 0.999, length = 6)`. Must be numeric and contain only real numbers between 0 (inclusive) and 1 (exclusive).

delta Values for regularized discriminant analysis delta parameter; default is `delta = c(0, 0.1, 1, 2, 3, 4)`. Must be numeric and contain only real numbers greater than or equal to 0.

eta Values for gradient boosting machine eta parameter; default is `eta = c(0.1, 0.3, 0.5, 0.7, 0.9)`. Must be numeric and contain only real numbers greater than 0 and less than 1.

max.depth Values for gradient boosting machine max.depth parameter; default is `max.depth = c(1, 2, 3, 4)`. Must be numeric and contain only integers greater than or equal to 1.

subsample Values for gradient boosting machine subsample parameter; default is `subsample = c(0.6, 0.7, 0.8, 0.9)`. Must be numeric and contain only real numbers greater than 0 and less than or equal to 1.

nrounds Values for gradient boosting machine nrounds parameter; default is `nrounds = c(100, 200, 300, 500)`. Must be numeric and contain only integers greater than or equal to 1.

`type.out` Type of output desired: `type.out = "measures"` gives array containing classification performance measures for all replications while `type.out = "descriptives"` gives list of descriptive statistics calculated across all replications for each performance measure. Both options also provide the estimated training weights and classification weights. Defaults to `type.out = "descriptives"`.

`foldid` Integer vector containing fold IDs for k-fold cross-validation. If not provided, fold IDs are generated randomly for number of folds `nfolds`.

`prior` Prior probabilities of class membership. If specified, the probabilities should be in the order of the factor levels of input `y`. If unspecified, the observed class proportions for input `y` are used. Based on `prior`, inverse probability weights are calculated to account for class imbalance. Note that RF and RDA ignore `prior` and use uniform priors to handle imbalance.

`cmode` Integer value of 1, 2, or 3 (or 4 if `x` is a four-way array) specifying the mode whose component weights will be predictors for classification. Defaults to the last mode of the input array (i.e., defaults to 3 for three-way array, and to 4 for four-way array). If `model = "parafac2"`, last mode will be used.

`seeds` Random seeds to be associated with each replication. Default is `seeds = 1:nrep`.

`plot.out` Logical indicating whether to output one or more box plots of classification performance measures that are plotted across classification methods and number of components.

`plot.measures` Character vector containing values that specify for plotting one or more of 11 possible classification performance measures. Only relevant when `plot.out = TRUE`. Should contain one or more of the following labels: `c("err", "acc", "tpr", "fpr", "tnr", "fnr", "ppv", "npv", "fdr", "fom", "fs")`. A box plot will be created for each measure that is specified, summarizing output across replications. Note that additional information about each label is available in the Details section of the help file for function `cpm`. Note also that there are a few cases where the x-axis tick labels for a plot might not appear. A future update is planned to fix this issue.

`parallel` Logical indicating if parallel computing should be implemented. If TRUE, the package **parallel** is used for parallel computing. For all classification methods except penalized logistic regression, the **doParallel** package is used as a wrapper. Defaults to FALSE, which implements sequential computing.

`cl` Cluster for parallel computing, which is used when `parallel = TRUE`. Note that if `parallel = TRUE` and `cl = NULL`, then the cluster is defined as `makeCluster(detectCores())`.

`verbose` If TRUE, progress is printed.

`...` Additional arguments to be passed to function `parafac` for fitting a Parafac model or function `parafac2` for fitting a Parafac2 model. Example: can impose different constraints on different modes of the input array using the argument

const. See help file for function parafac or for function parafac2 for additional details.

Details

Data are split into a training set and a testing set. After fitting a Parafac or Parafac2 model with the training set using package **multiway** (see parafac or parafac2 in **multiway** for details), the estimated classification mode weight matrix is passed to one or more classification methods. The methods include: penalized logistic regression (PLR); support vector machine (SVM); random forest (RF); feed-forward neural network (NN); regularized discriminant analysis (RDA); and gradient boosting machine (GBM).

Package **glmnet** fits models for PLR. PLR tunes penalty parameter lambda while the elastic net parameter alpha is set by the user (see the help file for function cv.glmnet in package **glmnet**). For SVM, package **e1071** is used with a radial basis kernel. Penalty parameter cost and radial basis parameter gamma are used (see svm in package **e1071**). For RF, package **randomForest** is used and implements Breiman's random forest algorithm. The number of predictors sampled at each node split is set at the default of sqrt(R), where R is the number of Parafac or Parafac2 components. Two tuning parameters allowed are ntree, the number of trees to be grown, and nodesize, the minimum size of terminal nodes (see randomForest in package **randomForest**). For NN, package **nnet** fits a single-hidden-layer, feed-forward neural network model. Penalty parameters size (i.e., number of hidden layer units) and decay (i.e., weight decay) are used (see **nnet**). For RDA, package **rda** fits a shrunken centroids regularized discriminant analysis model. Tuning parameters include rda.alpha, the shrinkage penalty for the within-class covariance matrix, and delta, the shrinkage penalty of class centroids towards the overall dataset centroid. For GBM, package **xgboost** fits a gradient boosting machine model. Four tuning parameters are allowed: (1) eta, the learning rate; (2) max.depth, the maximum tree depth; (3) subsample, the fraction of samples per tree; and (4) nrounds, the number of boosting trees to build.

For all six methods, k-fold cross-validation is implemented to tune classification parameters where the number of folds is set by argument nfolds. Separately, the trained Parafac or Parafac2 model is used to predict the classification mode's component weights using the testing set data. The predicted component weights and the optimized classification method are then used to predict class labels. Finally, classification performance measures are calculated. The process is repeated over a number of replications with different random splits of the input array and of the class labels at each replication.

Value

Returns an object of class wrapcpfa either with a three-way array with classification performance measures for each model and for each replication, or with a list containing matrices with descriptive statistics for performance measures calculated across all replications. Specify type.out = "measures" to output the array of performance measures. Specify type.out = "descriptives" to output descriptive statistics across replications. In addition, for both options, the following are also provided:

predweights	List of predicted classification weights for each Parafac or Parafac2 model and for each replication.
train.weights	List of lists of training weights for each Parafac or Parafac2 model and for each replication.

opt.tune	List of optimal tuning parameters for classification methods for each Parafac or Parafac2 model and for each replication.
mean.opt.tune	Mean across all replications of optimal tuning parameters for classification methods for each Parafac or Parafac2 model.
X	Three-way or four-way data array or list used in argument x.
y	Vector of class labels used in input argument y.
nfac	Number of components used to fit each Parafac or Parafac2 model.
model	Character designating the Parafac model that was used, either model = "parafac" for the Parafac model or model = "parafac2" for the Parafac2 model.
method	Classification methods used.
const	Constraints used in fitting Parafac or Parafac2 models.
cmode	Integer value used to specify the mode whose component weights were predictors for classification.
family	Character value used to specify binary classification (family = "binomial") or multiclass classification (family = "multinomial").

Note

If argument cmode is not null, input array x is reshaped with function aperm such that the cmode dimension of x is ordered last. Estimated mode A and B (and mode C for a four-way array) weights that are outputted as Aweights and Bweights (and Cweights) reflect this permutation. For example, if x is a four-way array and cmode = 2, the original input modes 1, 2, 3, and 4 will correspond to output modes 1, 3, 4, 2. Here, output A = input 1; B = 3, and C = 4 (i.e., the second mode specified by cmode has been moved to the D mode/last mode). For model = "parafac2", classification mode is assumed to be the last mode (i.e., mode C for three-way array and mode D for four-way array).

In addition, note that the following combination of arguments will give an error: nfac = 1, family = "multinomial", method = "PLR". The issue arises from providing glmnet::cv.glmnet input x an input matrix that has a single column. The issue is resolved for family = "binomial" because a column of 0s is appended to the single column, but this solution does not appear to work for the multiclass case. As such, this combination of arguments is not currently allowed. Future updates are planned to resolve this issue.

Applications of this function to real datasets can be explored at the following repository: https://github.com/matthewasisgress/multiway-classification/.

Author(s)

Matthew A. Asisgress <mattgress@protonmail.ch>

References

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., Li, Y., Yuan, J. (2025). xgboost: Extreme gradient boosting. R Package Version 1.7.9.1.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, 29(5), 1189-1232.

Friedman, J. H. (1989). Regularized discriminant analysis. Journal of the American Statistical Association, 84(405), 165-175.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software, 33(1), 1-22.

Gaujoux, R. (2025). doRNG: Generic reproducible parallel backend for 'foreach' loops. R Package Version 1.8.6.2.

Guo, Y., Hastie, T., and Tibshirani, R. (2007). Regularized linear discriminant analysis and its application in microarrays. Biostatistics, 8(1), 86-100.

Guo, Y., Hastie, T., and Tibshirani, R. (2023). rda: Shrunken centroids regularized discriminant analysis. R Package Version 1.2-1.

Harshman, R. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. UCLA Working Papers in Phonetics, 16, 1-84.

Harshman, R. (1972). PARAFAC2: Mathematical and technical notes. UCLA Working Papers in Phonetics, 22, 30-44.

Harshman, R. and Lundy, M. (1994). PARAFAC: Parallel factor analysis. Computational Statistics and Data Analysis, 18, 39-72.

Helwig, N. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. Biometrical Journal, 59(4), 783-803.

Helwig, N. (2025). multiway: Component models for multi-way data. R Package Version 1.0-7.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomForest. R News 2(3), 18–22.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2024). e1071: Misc functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R Package Version 1.7-16.

Ripley, B. (1994). Neural networks and related methods for classification. Journal of the Royal Statistical Society: Series B (Methodological), 56(3), 409-437.

Venables, W. and Ripley, B. (2002). Modern applied statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.

Examples

```
########## Parafac2 example with 4-way array and multiclass response ##########
## Not run:
# set seed
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights
```

```
# define target correlation matrix for columns of D mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a four-way ragged array connected to a response
data <- simcpfa(arraydim = c(10, 11, 12, 100), model = "parafac2", nfac = 3,
                nclass = 3, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 4, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 20)
gamma <- c(0, 1)
cost <- c(0.1, 5)
ntree <- c(200, 300)
nodesize <- c(1, 2)
size <- c(1, 2)
decay <- c(0, 1)
rda.alpha <- seq(0.1, 0.9, length = 2)
delta <- c(0.1, 2)
eta <- c(0.3, 0.7)
max.depth <- c(1, 2)
subsample <- c(0.75)
nrounds <- c(100)
method <- c("PLR", "SVM", "RF", "NN", "RDA", "GBM")
family <- "multinomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost, ntree = ntree,
                   nodesize = nodesize, size = size, decay = decay,
                   rda.alpha = rda.alpha, delta = delta, eta = eta,
                   max.depth = max.depth, subsample = subsample,
                   nrounds = nrounds)
model <- "parafac2"
nfolds <- 10
nstart <- 10

# constrain first mode weights to be orthogonal, fourth mode to be nonnegative
const <- c("orthog", "uncons", "uncons", "nonneg")

# fit Parafac2 model and use fourth mode weights to tune classification
# methods, to predict class labels, and to return classification
# performance measures pooled across multiple train-test splits
output <- cpfa(x = data$X, y = as.factor(data$y), model = model, nfac = 3,
               nrep = 5, ratio = 0.9, nfolds = nfolds, method = method,
               family = family, parameters = parameters,
               type.out = "descriptives", seeds = NULL, plot.out = TRUE,
               parallel = FALSE, const = const, nstart = nstart)

# print performance measure means across train-test splits
output$descriptive$mean

## End(Not run)
```

cpm *Classification Performance Measures*

Description

Calculates multiple performance measures for binary or multiclass classification. Uses known class labels and evaluates against predicted labels.

Usage

```
cpm(x, y, level = NULL, fbeta = NULL, prior = NULL)
```

Arguments

x	Known class labels of class numeric, factor, or integer. If factor, converted to class integer in the order of factor levels with integers beginning at 0 (i.e., for binary classification, factor levels become 0 and 1; for multiclass, levels become 0, 1, 2, etc.).
y	Predicted class labels of class numeric, factor, or integer. If factor, converted to class integer in the order of factor levels with integers beginning at 0 (i.e., for binary classification, factor levels become 0 and 1; for multiclass, levels become 0, 1, 2, etc.).
level	Optional argument specifying possible class labels. For cases where x or y do not contain all possible classes. Can be of class numeric, integer, or character. Must contain two elements for binary classification, and contain three or more elements for multiclass classification. If integer, integers should be ordered (e.g., binary with `c(0, 1)`; or three-class with `c(0, 1, 2)`). Note: if both x and y jointly contain only a single value (e.g., 1), must specify argument `level` in order to identify classification as binary or multiclass.
fbeta	Optional numeric argument specifying beta value for F-score. Defaults to `fbeta = 1`, providing an F1-score (i.e., the balanced harmonic mean between precision and recall). Can be any real number.
prior	Optional numeric argument specifying weights for classes. Currently only implemented with multiclass problems. Defaults to `prior = c(rep(1/llev, llev))`, where `llev` is the number of classes, providing equal importance across classes.

Details

Selecting one class as a negative class and one class as a positive class, binary classification generates four possible outcomes: (1) negative cases classified as positives, called false positives (FP); (2) negative cases classified as negatives, called true negatives (TN); (3) positive cases classified as negatives, called false negatives (FN); and (4) positive cases classified as positives, called true positives (TP).

Multiple evaluation measures are calculated using these four outcomes. Measures include: overall error (ERR), also called fraction incorrect; overall accuracy (ACC), also called fraction correct; true positive rate (TPR), also called recall, hit rate, or sensitivity; false negative rate (FNR), also

called miss rate; false positive rate (FPR), also called fall-out; true negative rate (TNR), also called specificity or selectivity; positive predictive value (PPV), also called precision; false discovery rate (FDR); negative predictive value (NPV); false omission rate (FOR); and F-score (FS).

In multiclass classification, the four outcomes are possible for each individual class in macro-averaging, and performance measures are averaged over classes. Macro-averaging gives equal importance to all classes. For multiclass classification, calculated measures are currently only macro-averaged. See the listed reference in this help file for additional details on micro-averaging.

For binary classification, this function assumes a negative class and a positive class (i.e., it contains a reference group) and is ordered. Multiclass classification is currently assumed to be unordered.

Computational details:

ERR = (FP + FN) / (TP + TN + FP + FN).

ACC = (TP + TN) / (TP + TN + FP + FN), and ACC = 1 - ERR.

TPR = TP / (TP + FN).

FNR = FN / (FN + TP), and FNR = 1 - TPR.

FPR = FP / (FP + TN).

TNR = TN / (TN + FP), and TNR = 1 - FPR.

PPV = TP / (TP + FP).

FDR = FP / (FP + TP), and FDR = 1 - PPV.

NPV = TN / (TN + FN).

FOR = FN / (FN + TN), and FOR = 1 - NPV.

FS = (1 + beta^2) * ((PPV * TPR) / (((beta^2)*PPV) + TPR)).

All performance measures calculated are between 0 and 1, inclusive. For multiclass classification, macro-averaged values are provided for each performance measure. Note that 'beta' in FS represents the relative weight such that recall (TPR) is beta times more important than precision (PPV). See reference for more details.

Value

Returns list where first element is a full confusion matrix `cm` and where the second element is a data frame containing performance measures. For multiclass classification, macro-averaged values are provided (i.e., each measure is calculated for each class, then averaged over all classes; the average is weighted by argument `prior` if provided). The second list element contains the following performance measures:

`cm`	A confusion matrix with counts for each of the possible outcomes.
`err`	Overall error (ERR). Also called fraction incorrect.
`acc`	Overall accuracy (ACC). Also called fraction correct.
`tpr`	True positive rate (TPR). Also called recall, hit rate, or sensitivity.
`fpr`	False positive rate (FPR). Also called fall-out.
`tnr`	True negative rate (TNR). Also called specificity or selectivity.
`fnr`	False negative rate (FNR). Also called miss rate.
`ppv`	Positive predictive value (PPV). Also called precision.

npv	Negative predictive value (NPV).
fdr	False discovery rate (FDR).
fom	False omission rate (FOR).
fs	F-score. Mean between TPR (recall) and PPV (precision) varying by importance given to recall over precision (see Details section and argument fbeta).

Author(s)

Matthew Asisgress <mattgress@protonmail.ch>

References

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing and Management, 45(4), 427-437.

Examples

```
########## Parafac example with 3-way array and binary response ##########
## Not run:
# set seed and simulate a three-way array related to a binary response
set.seed(5)

# define target correlation matrix for columns of C mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# simulate a three-way array connected to a response
data <- simcpfa(arraydim = c(11, 12, 100), model = "parafac", nfac = 3,
                nclass = 2, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 3, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 2)
gamma <- c(0, 0.01)
cost <- c(1, 2)
method <- c("PLR", "SVM")
family <- "binomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost)
model <- "parafac"
nfolds <- 3
nstart <- 3

# constrain first mode weights to be orthogonal
const <- c("orthog", "uncons", "uncons")
```

```
# fit Parafac models and use third mode to tune classification methods
tune.object <- tunecpfa(x = data$X[, , 1:80], y = as.factor(data$y[1:80, ]),
                        model = model, nfac = 3, nfolds = nfolds,
                        method = method, family = family,
                        parameters = parameters, parallel = FALSE,
                        const = const, nstart = nstart)

# predict class labels
predict.labels <- predict(object = tune.object, newdata = data$X[, , 81:100],
                          type = "response")

# calculate performance measures for predicted class labels
y.pred <- predict.labels[, 1]
evalmeasure <- cpm(x = as.numeric(data$y[81:100, ]), y = y.pred)

# print performance measures
evalmeasure

## End(Not run)
```

`cpm.all` *Wrapper for Calculating Classification Performance Measures*

Description

Applies function `cpm` to multiple sets of class labels. Each set of class labels is evaluated against the same set of predicted labels. Works with output from function `predict.tunecpfa` and calculates classification performance measures for multiple classifiers or numbers of components.

Usage

```
cpm.all(x, y, ...)
```

Arguments

`x`	A data frame where each column contains a set of class labels of class numeric, factor, or integer. If a set is of class factor, that set is converted to class integer in the order of factor levels with integers beginning at 0 (i.e., for binary classification, factor levels become 0 and 1; for multiclass, levels become 0, 1, 2, etc.).
`y`	Class labels of class numeric.
`...`	Additional arguments passed to function `cpm` for calculating classification performance measures.

Details

Wrapper function that applies function `cpm` to multiple sets of class labels and one other set of labels. See help file for function `cpm` for additional details.

Value

Returns a list with the following two elements:

`cm.list`	A list of confusion matrices, denoted `cm`, where each confusion matrix is associated with one comparison.
`cpms`	A data frame containing classification performance measures where each row contains measures for one comparison.

Author(s)

Matthew Asisgress <mattgress@protonmail.ch>

References

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing and Management, 45(4), 427-437.

Examples

```
########## Parafac example with 3-way array and binary response ##########
## Not run:
# set seed and simulate a three-way array related to a binary response
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# define target correlation matrix for columns of C mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a three-way array connected to a response
data <- simcpfa(arraydim = c(11, 12, 100), model = "parafac", nfac = 3,
                nclass = 2, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 3, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 2)
gamma <- c(0, 0.01)
cost <- c(1, 2)
method <- c("PLR", "SVM")
family <- "binomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost)
model <- "parafac"
nfolds <- 3
nstart <- 3

# constrain first mode weights to be orthogonal
```

```
const <- c("orthog", "uncons", "uncons")

# fit Parafac models and use third mode to tune classification methods
tune.object <- tunecpfa(x = data$X[, , 1:80], y = as.factor(data$y[1:80, ]),
                        model = model, nfac = 3, nfolds = nfolds,
                        method = method, family = family,
                        parameters = parameters, parallel = FALSE,
                        const = const, nstart = nstart)

# predict class labels
predict.labels <- predict(object = tune.object, newdata = data$X[, , 81:100],
                          type = "response")

# calculate performance measures for predicted class labels
evalmeasure <- cpm.all(x = predict.labels, y = as.numeric(data$y[81:100, ]))

# print performance measures
evalmeasure

## End(Not run)
```

`plotcpfa` *Plot Optimal Model from Classification with Parallel Factor Analysis*

Description

Plots optimal model based on results from a 'wrapcpfa' object generated by function `cpfa`.

Usage

```
plotcpfa(object, cmeasure = "acc", meanvalue = TRUE, supNum = FALSE,
         parallel = FALSE, cl = NULL, scale.remode = NULL, newscales = 1,
         scale.abmode = NULL, sign.remode = NULL, newsigns = 1,
         sign.abmode = NULL, ...)
```

Arguments

`object`	An object of class 'wrapcpfa' from function `cpfa`.
`cmeasure`	Classification performance measure used to select the optimal number of components. Options include `c("err", "acc", "tpr", "fpr", "tnr", "fnr", "ppv", "npv", "fdr", "fom", "fs")`. If `cmeasure` is in `c("err", "fpr", "fnr", "fdr", "fom")`, the number of components that minimized `cmeasure` is selected among all classification methods. Otherwise, the number that maximized `cmeasure` is selected.
`meanvalue`	Logical indicating whether to find the optimal number of components based on the mean performance across replications from the results generated by `cpfa`. If `meanvalue = FALSE`, the median is used.

supNum	Logical indicating whether to suppress text displaying component weight values within plot cells. If TRUE, values are not displayed.
parallel	Logical indicating whether parallel computing should be used. If TRUE, parallel computing is used.
cl	Cluster for parallel computing, which is used when parallel = TRUE. Note that if parallel = TRUE and cl = NULL, then the cluster is defined as makeCluster(detectCores()).
scale.remode	Character that indicates a mode to rescale. Must be one of c("A", "B", "C", "D"). Sent directly to argument mode in function rescale from package **multiway**. See help file for rescale for additional details.
newscales	The root mean-square for columns of the mode indicated by scale.remode. See help file for rescale for additional details.
scale.abmode	Character that indicates the mode that absorbs the inverse of rescalings applied to the mode indicated by scale.remode. Must be one of c("A", "B", "C", "D"). Sent directly to argument absorb in function rescale from package **multiway**. See help file for rescale for additional details.
sign.remode	Character that indicates a mode to resign. Must be one of c("A", "B", "C", "D"). Sent directly to argument mode in function resign from package **multiway**. See help file for resign for additional details.
newsigns	Scalar or vector indicating resignings for columns of the mode indicated by sign.remode. See help file for resign for additional details.
sign.abmode	Character that indicates the mode that absorbs the negation of the resignings applied to the mode indicated by sign.remode. Must be one of c("A", "B", "C", "D"). Sent directly to argument absorb in function resign from package **multiway**. See help file for resign for additional details.
...	Additional arguments to be passed to function parafac for fitting a Parafac model or function parafac2 for fitting a Parafac2 model. See help file for function parafac or for function parafac2 for additional details.

Details

Selects the number of components that optimized a performance measure across all classification methods used by cpfa. With this optimal number of components, fits the Parafac or Parafac2 model that was used by cpfa to create the input 'wrapcpfa' object. Uses same constraints used in cpfa. Plots component weights for this optimal model using heatmaps. Darker red indicates component weights that are more negative while darker green indicates component weights that are more positive. For three-way Parafac, plots A and B weights. For four-way Parafac, plots A, B, and C weights. For three-way Parafac2, plots B weights. For four-way Parafac2, plots B and C weights.

Value

Returns one or more heatmap plots of component weights for the optimal Parafac or Parafac2 model. Returns list of estimated component weights from the optimal model.

Note

Applications of this function to real datasets can be explored at the following repository: https://github.com/matthewasisgress/multiway-classification/.

Author(s)

Matthew Asisgress <mattgress@protonmail.ch>

References

See help file for function cpfa for a list of references.

Examples

```
########## Parafac2 example with 4-way array and multiclass response ##########
## Not run:
# set seed and simulate a four-way ragged array related to a multiclass response
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# define target correlation matrix for columns of D mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a four-way ragged array connected to a response
data <- simcpfa(arraydim = c(10, 11, 12, 100), model = "parafac2", nfac = 3,
                nclass = 3, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 4, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 2)
gamma <- c(0, 1)
cost <- c(0.1, 5)
rda.alpha <- seq(0.1, 0.9, length = 2)
delta <- c(0.1, 2)
method <- c("PLR", "SVM", "RDA")
family <- "multinomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost,
                   rda.alpha = rda.alpha, delta = delta)
model <- "parafac2"
nfolds <- 3
nstart <- 3

# constrain first mode weights to be orthogonal, fourth mode to be nonnegative
const <- c("orthog", "uncons", "uncons", "nonneg")

# fit Parafac2 model and use fourth mode weights to tune classification
# methods, to predict class labels, and to return classification
# performance measures pooled across multiple train-test splits
output <- cpfa(x = data$X, y = as.factor(data$y), model = model, nfac = 3,
               nrep = 2, ratio = 0.8, nfolds = nfolds, method = method,
               family = family, parameters = parameters,
```

```
                type.out = "descriptives", seeds = NULL, plot.out = TRUE,
                parallel = FALSE, const = const, nstart = nstart, ctol = 1e-2)

# plot heatmap of component weights for optimal model
plotcpfa(output, nstart = nstart, ctol = 1e-3)

## End(Not run)
```

`predict.tunecpfa` *Predict Method for Tuning for Classification with Parallel Factor Analysis*

Description

Obtains predicted class labels from a 'tunecpfa' model object generated by function `tunecpfa`.

Usage

```
## S3 method for class 'tunecpfa'
predict(object, newdata = NULL, method = NULL,
        type = c("response", "prob", "classify.weights"),
        threshold = NULL, ...)
```

Arguments

`object`	A fit object of class 'tunecpfa' produced by function `tunecpfa`.
`newdata`	An optional three-way or four-way data array used to predict Parafac or Parafac2 component weights using estimated Parafac or Parafac2 model component weights from the input object. For Parafac2, can be a list of length `K` where the `k`-th element is a matrix or three-way array associated with the `k`-th element. Array or list must contain only real numbers. Dimensions must match dimensions of original data for all modes except the classification mode. If omitted, the original data are used.
`method`	Character vector indicating classification methods to use. Possible methods include penalized logistic regression (PLR); support vector machine (SVM); random forest (RF); feed-forward neural network (NN); regularized discriminant analysis (RDA); and gradient boosting machine (GBM). If none selected, default is to use all methods.
`type`	Character vector indicating type of prediction to return. Possible values include: (1) `"response"`, returning predicted class labels; (2) `"prob"`, returning predicted class probabilities; or (3) `"classify.weights"`, returning predicted component weights used for classification in the specified Parafac models. Defaults to `"response"`.
`threshold`	For binary classification, value indicating prediction threshold over which observations are classified as the positive class. If not provided, calculates threshold using class proportions in original data. For multiclass classification, `threshold` is not currently implemented.
`...`	Additional predict arguments. Currently ignored.

Details

Predicts class labels for a binary or a multiclass outcome. Specifically, predicts component weights for one mode of a Parallel Factor Analysis-1 (Parafac) model or a Parallel Factor Analysis-2 (Parafac2) model using new data and previously estimated mode weights from original data. Passes predicted component weights to one or several classification methods as new data for predicting class labels.

Tuning parameters optimized by k-fold cross-validation are used for each classification method (see help for `tunecpfa`). If not supplied in argument `threshold`, prediction threshold for all classification methods is calculated using proportions of class labels for original data in the binary case (and the positive class proportion is set as the threshold). For multiclass case, class with highest probability is chosen.

Value

Returns one of the following, depending on the choice for argument `type`:

`type = "response"`
: A data frame containing predicted class labels for each Parafac model and classification method selected (see argument `type`). Number of columns is equal to number of methods times number of Parafac models. Number of rows is equal to number of predicted observations.

`type = "prob"`
: A list containing predicted probabilities for each Parafac model and classification method selected (see argument `type`). The number of list elements is equal to the number of methods times the number of Parafac models.

`type = "classify.weights"`
: List containing predicted component weights for each Parafac or Parafac2 model. Length is equal to number of Parafac models that were fit.

Note

Applications of this function to real datasets can be explored at the following repository: https://github.com/matthewasisgress/multiway-classification/.

Author(s)

Matthew Asisgress <mattgress@protonmail.ch>

References

See help file for function `tunecpfa` for a list of references.

Examples

```
########## Parafac example with 3-way array and binary response ##########
## Not run:
# set seed and simulate a three-way array related to a binary response
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
```

```
                              lambda = 3),                # for A weights
                  distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# define target correlation matrix for columns of C mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a three-way array connected to a response
data <- simcpfa(arraydim = c(11, 12, 100), model = "parafac", nfac = 3,
                nclass = 2, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 3, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 2)
gamma <- c(0, 0.01)
cost <- c(1, 2)
method <- c("PLR", "SVM")
family <- "binomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost)
model <- "parafac"
nfolds <- 3
nstart <- 3

# constrain first mode weights to be orthogonal
const <- c("orthog", "uncons", "uncons")

# fit Parafac models and use third mode to tune classification methods
tune.object <- tunecpfa(x = data$X[, , 1:80], y = as.factor(data$y[1:80, ]),
                        model = model, nfac = 3, nfolds = nfolds,
                        method = method, family = family,
                        parameters = parameters, parallel = FALSE,
                        const = const, nstart = nstart)

# predict class labels
predict.labels <- predict(object = tune.object, newdata = data$X[, , 81:100],
                          type = "response")

# print predicted labels
predict.labels

## End(Not run)
```

`print.tunecpfa` *Print Method for Tuning for Classification with Parallel Factor Analysis*

Description

Prints summary of a 'tunecpfa' model object generated by function `tunecpfa`.

Usage

```
## S3 method for class 'tunecpfa'
print(x, ...)
```

Arguments

x	A fit object of class 'tunecpfa' from function tunecpfa.
...	Additional print arguments.

Details

Prints names of the models and methods used to create the input 'tunecpfa' model object. Prints misclassification error rates and estimation times in seconds.

Value

Returns a summary of the 'tunecpfa' model object.

Author(s)

Matthew Asisgress <mattgress@protonmail.ch>

References

See help file for function tunecpfa for a list of references.

Examples

```
########## Parafac example with 3-way array and binary response ##########
## Not run:
# set seed and simulate a three-way array connected to a binary response
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# define target correlation matrix for columns of C mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a three-way array connected to a response
data <- simcpfa(arraydim = c(11, 12, 100), model = "parafac", nfac = 3,
                nclass = 2, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 3, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 2)
gamma <- c(0, 0.01)
```

```
cost <- c(1, 2)
method <- c("PLR", "SVM")
family <- "binomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost)
model <- "parafac"
nfolds <- 3
nstart <- 3

# constrain first mode weights to be orthogonal
const <- c("orthog", "uncons", "uncons")

# fit Parafac models and use third mode to tune classification methods
tune.object <- tunecpfa(x = data$X, y = as.factor(data$y), model = model,
                        nfac = 3, nfolds = nfolds, method = method,
                        family = family, parameters = parameters,
                        parallel = FALSE, const = const, nstart = nstart)

# print summary of output
print(tune.object)

## End(Not run)
```

`simcpfa` *Simulate Data for Classification with Parallel Factor Analysis*

Description

Simulates a three-way or four-way data array and a set of class labels that are related to the simulated array through one mode of the array. Data array is simulated using either a Parafac or Parafac2 model with no constraints. Weights for mode weight matrices can be drawn from 12 common probability distributions. Alternatively, custom weights can be provided for any mode.

Usage

```
simcpfa(arraydim = NULL, model = "parafac", nfac = 2, nclass = 2,
        smethod = "logistic", nreps = 100, onreps = 10, props = NULL,
        corresp = c(0.3, -0.3), meanpred = c(0, 0), modes = 3,
        corrpred = matrix(c(1, 0.2, 0.2, 1), nrow = 2), pf2num = NULL,
        Amat = NULL, Bmat = NULL, Cmat = NULL, Dmat = NULL, Gmat = NULL,
        Emat = NULL, technical = list())
```

Arguments

`arraydim`	Numeric vector containing the number of dimensions for each mode of the simulated data array. Must contain integers greater than or equal to 2.
`model`	Character specifying the model to use for simulating the data array. Must be either 'parafac' or 'parafac2'.
`nfac`	Number of components in the Parafac or Parafac2 model. Must be an integer greater than or equal to 1.

`nclass` Number of classes in simulated class labels. Must be an integer greater than or equal to 2.

`smethod` Simulation method, either "logistic" or "eigende". The former implements an iterative Monte Carlo rejection sampling technique based on the generalized linear model (slower). The latter uses a multivariate normal distribution and constructs a joint covariance matrix, employing a eigendecomposition and assuming class labels arise from discretizing continuous latent variables (faster).

`nreps` Number of replications for simulating class labels for a given set of classification mode component weights.

`onreps` Number of replications for simulating a set of classification mode component weights.

`props` Target proportions for simulated class labels in output 'y'.

`corresp` Numeric vector of target correlations between simulated class labels and columns of the classification mode component weight matrix. Must have length equal to 'nfac'.

`meanpred` Numeric vector of means used to generate the classification mode component weights. Must be real numbers. Operates as the mean vector parameterizing a multivariate normal distribution from which classification mode component weights are generated. Length must be equal to input `nfac`.

`modes` Single integer of either 3 or 4, indicating whether to simulate a three-way or four-way data array, respectively.

`corrpred` A positive definite correlation matrix containing the target correlations for the classification mode component weights. Must have number of rows and columns equal to input 'nfac'. Operates as the covariance matrix parameterizing a multivariate normal distribution from which classification mode component weights are generated.

`pf2num` When `model = 'parafac2'`, number of rows for each simulated matrix in the list of matrices `Amat`. Replaces the first element of input `arraydim` because, for the Parafac2 model, the number of rows in each simulated matrix can vary. If not specified when `model = 'parafac2'`, defaults to `rep(c((nfac + 1), (nfac + 2), (nfac + 3)), length.out = arraydim[modes])`.

`Amat` When `model = 'parafac'`, a matrix of A mode weights with number of rows equal to the first element of input 'arraydim' and with number of columns equal to input 'nfac'. When `model = 'parafac2'`, a list with length equal to the last element of input 'arraydim', where each list element contains a matrix with number of rows of at least 2 and with number of columns equal to input 'nfac'. When provided, replaces a simulated `Amat`.

`Bmat` A matrix of B mode weights with number of rows equal to the second element of input 'arraydim' and with number of columns equal to the input 'nfac'. When provided, replaces a simulated `Bmat`.

`Cmat` A matrix of C mode weights with number of rows equal to the third element of input 'arraydim' and with number of columns equal to the input 'nfac'. When provided, replaces a simulated `Cmat` when `modes = 4`. When `modes = 3`, replaces the simulated classification mode weight matrix. If provided when `modes = 3`, `onreps` is reduced to one.

`Dmat` A matrix of D mode weights with number of rows equal to the fourth element of input 'arraydim' and with number of columns equal to the input 'nfac'. When `modes = 4`, replaces the simulated classification mode weight matrix. When `modes = 3`, this argument is ignored. If provided when `modes = 4`, `onreps` is reduced to one.

`Gmat` When `model = 'parafac2'`, a matrix of G mode weights with number of rows equal to input 'nfac' and with number of columns equal to input 'nfac'. When provided, replaces a simulated `Gmat`.

`Emat` When `model = 'parafac'`, an array containing noise to be added to the corresponding elements in the simulated data array. Error array dimensions must be equal to the values contained in `arraydim`. When `model = 'parafac2'`, a list containing either matrices (i.e., when `modes = 3`) or three-way arrays (i.e., when `modes = 4`) whose elements contain noise to be added to corresponding elements in the simulated data array. When provided, replaces a simulated `Emat`.

`technical` List containing arguments related to distributions from which to simulate data. When specified, must contain one or more of the following:

distA List containing arguments specifying the distribution from which deviates are drawn for A mode weights contained in `Amat`. Defaults to standard normal distribution when not specified. See Details section for additional information on acceptable arguments.

distB List containing arguments specifying the distribution from which deviates are drawn for B mode weights contained in `Bmat`. Defaults to standard normal distribution when not specified. See Details section for additional information on acceptable arguments.

distC For when `modes = '4'`, list containing arguments specifying the distribution from which deviates are drawn for C mode weights contained in `Cmat`. Defaults to standard normal distribution when not specified. See Details section for additional information on acceptable arguments.

distG For when `model = 'parafac2'`, list containing arguments specifying the distribution from which deviates are drawn for G weights contained in `Gmat`. Defaults to standard normal distribution when not specified. See Details section for additional information on acceptable arguments.

distE List containing arguments specifying the distribution from which deviates are drawn for error contained in `Emat`. Defaults to standard normal distribution when not specified. See Details section for additional information on acceptable arguments.

Details

By selecting `smethod = "logistic"`, the data array simulation consists of two steps. First, a Monte Carlo simulation is conducted to simulate class labels using a binomial logistic (i.e., in the binary case) or multinomial logistic (i.e., in the multiclass case) regression model. Specifically, columns of the classification mode weights matrix (e.g., `Cmat` when `modes = 3`) are generated from a multivariate normal distribution with mean vector `meanpred` and with covariance matrix `corrpred`. Values are then drawn randomly from a uniform or a normal distribution and serve as beta coefficients. A linear combination of these beta coefficients and the generated classification weights produces a linear systematic part, which is passed through the logistic function (i.e., the sigmoid) in the binary

case or through the softmax function in the multiclass case. Resulting probabilities are used to assign class labels. The simulation repeats classification weights generation `onreps` times and repeats class label generation, within each `onreps` iteration, a total of `nreps` times. The generated class labels that correlate best with the generated classification weights (i.e., with correlations closest to `corresp`) are retained as the final class labels with corresponding final classification weights. An adaptive sampling technique is used during the simulation such that optimal beta coefficients from previous iterations are used to parameterize a normal distribution, from which new coefficients are drawn in subsequent iterations. Note that, if any simulation replicate produces a set of class labels where all labels are the same (i.e., have no variance), that replicate is discarded. Note also that `onreps` is ignored when the classification mode weight matrix (i.e., `Cmat` when `modes = 3` or `Dmat` when `modes = 4`) is provided; in this case, class labels are simulated with respect to the provided classification mode weight matrix.

Second, depending on the chosen model (i.e., Parafac or Parafac2) specified via `model`, and depending on the number of modes specified via `modes`, component matrices are randomly generated for each mode of the data array. A data array is then constructed using a Parafac or Parafac2 structure from these weight matrices, including the generated classification mode weight matrix (i.e., `Cmat` or `Dmat`) from the first step. Alternatively, weight matrices can be provided to override random generation for any weight matrix with the exception of the classification mode. When provided, weight matrices are used to form the final data array. Finally, random noise is added to each value in the array. The resulting output is a synthetic multiway data array paired, through one mode of the array, with a simulated binary or multiclass response.

Alternatively, by selecting `smethod = "eigende"`, the function simulates component weights and a latent response variable simultaneously from a joint multivariate normal distribution using an eigendecomposition. The covariance structure is defined by `corrpred` and a corrected version of `corresp` that accounts for the attenuation in correlation caused by discretizing the latent response. This continuous latent response vector is then discretized into class labels using quantile cuts defined by the cumulative sum of `props`. This method offers an efficient, non-iterative alternative that satisfies the target covariance structure without rejection sampling.

The `technical` argument controls the probability distributions used to simulate weights for different modes. Currently, `technical` is highly structured. In particular, `technical` must be provided as a named list whose elements must be one of 'distA', 'distB', 'distC', 'distG', or 'distE', with the last letter of each name designating a mode or, in the case of 'distE', designating error. Each element provided must itself be a list where the first inner list element is named 'dname', specifying the distribution to be used to generate weights for a given mode or for error. There are 12 'dname' options: 'normal', 'uniform', 'gamma', 'beta', 'binomial', 'poisson', 'exponential', 'geometric', 'negbinomial', 'hypergeo', 'lognormal', and 'cauchy'. Additional arguments can be added to each inner list to parameterize the probability distribution being used. These arguments can be one of the following, for each distribution allowed:

For `dname = 'normal'`, allowed arguments are `mean` or `sd` (i.e., function `rnorm` is called).

For `dname = 'uniform'`, allowed arguments are `min` or `max` (i.e., function `runif` is called).

For `dname = 'gamma'`, allowed arguments are `shape` or `scale` (i.e., function `rgamma` is called).

For `dname = 'beta'`, allowed arguments are `shape1` or `shape2` (i.e., function `rbeta` is called).

For `dname = 'binomial'`, allowed arguments are `size` or `prob` (i.e., function `rbinom` is called).

For `dname = 'poisson'`, allowed argument is `lambda` (i.e., function `rpois` is called).

For `dname = 'exponential'`, allowed argument is `rate` (i.e., function `rexp` is called).

For dname = 'geometric', allowed argument is prob (i.e., function rgeom is called).

For dname = 'negbinomial', allowed arguments are size or prob (i.e., function rnbinom is called).

For dname = 'hypergeo', allowed arguments are m, n, or k (i.e., function rhyper is called).

For dname = 'lognormal', allowed arguments are meanlog or sdlog (i.e., function rlnorm is called).

For dname = 'cauchy', allowed arguments are location or scale (i.e., function rcauchy is called).

Note that if a weight matrix and technical information are both provided for a given mode (or for error), the weight matrix is used while technical information is ignored. See Examples below for an example of how to set up technical.

Value

X	Simulated data array with dimensions specified by arraydim and, when model = 'parafac2', also by pf2num. When model = 'parafac', X is an object of class 'array'. When model = 'parafac2', X is an object of class 'list'.
y	Simulated class labels provided as an object of class 'matrix', with number of rows equal to the last element of arraydim and with number of columns equal to 1.
model	Character value indicating whether Parafac or Parafac2 model was used to simulate the data array.
Amat	Simulated A mode weights. When model = 'parafac', output is a matrix with number of rows equal to the first element of arraydim and with number of columns equal to the number of components nfac. When model = 'parafac2', output is a list of matrices with number of rows for each matrix equal to those specified by pf2num and with number of columns equal to nfac. If Amat was supplied, returns original Amat instead of a simulated Amat.
Bmat	Simulated B mode weights provided as a matrix with number of rows equal to the second element of arraydim and with number of columns equal to the number of components nfac. If Bmat was supplied, returns original Bmat instead of a simulated Bmat.
Cmat	Simulated C mode weights provided as a matrix with number of rows equal to the third element of arraydim and with number of columns equal to the number of components nfac. If Cmat was supplied when modes = 4, returns original Cmat instead of a simulated Cmat.
Dmat	Simulated D mode weights provided when modes = 4. Output is a matrix with number of rows equal to the fourth element of arraydim and with number of columns equal to the number of components nfac.
Gmat	Simulated G weights provided when model = 'parafac2'. Provided as a matrix with number of rows and columns equal to nfac. If Gmat was supplied, returns original Gmat instead of a simulated Gmat.
Emat	Error array or list containing noise added to corresponding elements of simulated data array. Output has dimensions specified by arraydim and, when model = 'parafac2', also by pf2num. When model = 'parafac', Emat is an object of class 'array'. When model = 'parafac2', Emat is an object of class 'list'.

Author(s)

Matthew Asisgress <mattgress@protonmail.ch>

References

See help file for function cpfa for a list of references.

Examples

```
########## Parafac2 example with 4-way array and multiclass response ##########
## Not run:
# set seed for reproducibility
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# define target correlation matrix for columns of D mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a four-way ragged array connected to a response
data <- simcpfa(arraydim = c(10, 11, 12, 100), model = "parafac2", nfac = 3,
                nclass = 3, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 4, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# examine correlations among columns of classification mode matrix Dmat
cor(data$Dmat)

# examine correlations between columns of classification mode matrix Dmat and
# simulated class labels
cor(data$Dmat, data$y)

## End(Not run)
```

---

tunecpfa *Tuning for Classification with Parallel Factor Analysis*

Description

Fits Richard A. Harshman's Parallel Factor Analysis-1 (Parafac) model or Parallel Factor Analysis-2 (Parafac2) model to a three-way or four-way data array. Allows for multiple constraint options on tensor modes. Uses component weights from a single mode of the model as predictors to tune parameters for one or more classification methods via a k-fold cross-validation procedure. Supports binary and multiclass classification.

Usage

```
tunecpfa(x, y, model = c("parafac", "parafac2"), nfac = 1, nfolds = 10,
         method = c("PLR", "SVM", "RF", "NN", "RDA", "GBM"),
         family = c("binomial", "multinomial"), parameters = list(),
         foldid = NULL, prior = NULL, cmode = NULL, parallel = FALSE,
         cl = NULL, verbose = TRUE, ...)
```

Arguments

x	For Parafac or Parafac2, a three-way or four-way data array. For Parafac2, can be a list where each element is a matrix or three-way array. Array or list must contain real numbers. See note below.
y	A vector containing at least two unique class labels. Should be a factor that contains two or more levels. For binary case, ensure the order of factor levels (left to right) is such that negative class is first and positive class is second.
model	Character designating the Parafac model to use, either `model = "parafac"` to fit the Parafac model or `model = "parafac2"` to fit the Parafac2 model.
nfac	Number of components for each Parafac or Parafac2 model to fit. Default is `nfac = 1`.
nfolds	Numeric value specifying the number of folds for k-fold cross-validation. Must be 2 or greater. Default is `nfolds = 10`.
method	Character vector indicating classification methods to use. Possible methods include penalized logistic regression (PLR); support vector machine (SVM); random forest (RF); feed-forward neural network (NN); regularized discriminant analysis (RDA); and gradient boosting machine (GBM). If none selected, default is to use all methods.
family	Character value specifying binary classification (`family = "binomial"`) or multiclass classification (`family = "multinomial"`). If not provided, number of levels of input `y` is used, where two levels is binary, and where three or more levels is multiclass.
parameters	List containing arguments related to classification methods. When specified, must contain one or more of the following:

alpha Values for penalized logistic regression alpha parameter; default is `alpha = seq(0, 1, length = 6)`. Must be numeric and contain only real numbers between 0 and 1, inclusive.

lambda Optional user-supplied lambda sequence for `cv.glmnet` for penalized logistic regression. Default is NULL.

cost Values for support vector machine cost parameter; default is `cost = c(1, 2, 4, 8, 16, 32, 64)`. Must be numeric and contain only real numbers greater than or equal to 0.

gamma Values for support vector machine gamma parameter; default is `gamma = c(0, 0.01, 0.1, 1, 10, 100, 1000)`. Must be numeric and greater than or equal to 0.

ntree Values for random forest number of trees parameter; default is `ntree = c(100, 200, 400, 600, 800, 1600, 3200)`. Must be numeric and contain only integers greater than or equal to 1.

nodesize Values for random forest node size parameter; default is `nodesize = c(1, 2, 4, 8, 16, 32, 64)`. Must be numeric and contain only integers greater than or equal to 1.

size Values for neural network size parameter; default is `size = c(1, 2, 4, 8, 16, 32, 64)`. Must be numeric and contain only integers greater than or equal to 0.

decay Values for neural network decay parameter; default is `decay = c(0.001, 0.01, 0.1, 1, 2, 4, 8, 16)`. Must be numeric and contain only real numbers.

rda.alpha Values for regularized discriminant analysis alpha parameter; default is `rda.alpha = seq(0, 0.999, length = 6)`. Must be numeric and contain only real numbers between 0 (inclusive) and 1 (exclusive).

delta Values for regularized discriminant analysis delta parameter; default is `delta = c(0, 0.1, 1, 2, 3, 4)`. Must be numeric and contain only real numbers greater than or equal to 0.

eta Values for gradient boosting machine eta parameter; default is `eta = c(0.1, 0.3, 0.5, 0.7, 0.9)`. Must be numeric and contain only real numbers greater than 0 and less than 1.

max.depth Values for gradient boosting machine max.depth parameter; default is `max.depth = c(1, 2, 3, 4)`. Must be numeric and contain only integers greater than or equal to 1.

subsample Values for gradient boosting machine subsample parameter; default is `subsample = c(0.6, 0.7, 0.8, 0.9)`. Must be numeric and contain only real numbers greater than 0 and less than or equal to 1.

nrounds Values for gradient boosting machine nrounds parameter; default is `nrounds = c(100, 200, 300, 500)`. Must be numeric and contain only integers greater than or equal to 1.

`foldid` Vector containing fold IDs for k-fold cross-validation. Can be of class integer, numeric, or data frame. Should contain integers from 1 through the number of folds. If not provided, fold IDs are generated randomly for observations using 1 through the number of folds `nfolds`.

`prior` Prior probabilities of class membership. If specified, the probabilities should be in the order of the factor levels of input `y`. If unspecified, the observed class proportions for input `y` are used. Based on `prior`, inverse probability weights are calculated to account for class imbalance. Note that RF and RDA ignore `prior` and use uniform priors to handle imbalance.

`cmode` Integer value of 1, 2, or 3 (or 4 if `x` is a four-way array) specifying the mode whose component weights will be predictors for classification. Defaults to the last mode of the input array (i.e., defaults to 3 for three-way array, and to 4 for four-way array). If `model = "parafac2"`, last mode will be used.

`parallel` Logical indicating whether to use parallel computing. If TRUE, the package **parallel** is used for parallel computing. For all classification methods except penalized logistic regression, the **doParallel** package is used as a wrapper. Defaults to FALSE, which implements sequential computing.

`cl` Cluster for parallel computing, which is used when `parallel = TRUE`. Note that if `parallel = TRUE` and `cl = NULL`, then the cluster is defined as `makeCluster(detectCores())`.

verbose	If TRUE, progress is printed.
...	Additional arguments to be passed to function parafac for fitting a Parafac model or function parafac2 for fitting a Parafac2 model. Example: can impose different constraints on different modes of the input array using the argument const. See help file for function parafac or for function parafac2 for additional details.

Details

After fitting a Parafac or Parafac2 model using package **multiway** (see parafac or parafac2 in **multiway** for details), the estimated classification mode weight matrix is passed to one or more classification methods–including penalized logistic regression (PLR); support vector machine (SVM); random forest (RF); feed-forward neural network (NN); regularized discriminant analysis (RDA); and gradient boosting machine (GBM).

Package **glmnet** fits models for PLR. PLR tunes penalty parameter lambda while the elastic net parameter alpha is set by the user (see the help file for function cv.glmnet in package **glmnet**). For SVM, package **e1071** is used with a radial basis kernel. Penalty parameter cost and radial basis parameter gamma are used (see svm in package **e1071**). For RF, package **randomForest** is used and implements Breiman's random forest algorithm. The number of predictors sampled at each node split is set at the default of sqrt(R), where R is the number of Parafac or Parafac2 components. Two tuning parameters allowed are ntree, the number of trees to be grown, and nodesize, the minimum size of terminal nodes (see randomForest in package **randomForest**). For NN, package **nnet** fits a single-hidden-layer, feed-forward neural network model. Penalty parameters size (i.e., number of hidden layer units) and decay (i.e., weight decay) are used (see **nnet**). For RDA, package **rda** fits a shrunken centroids regularized discriminant analysis model. Tuning parameters include rda.alpha, the shrinkage penalty for the within-class covariance matrix, and delta, the shrinkage penalty of class centroids towards the overall dataset centroid. For GBM, package **xgboost** fits a gradient boosting machine model. Four tuning parameters are allowed: (1) eta, the learning rate; (2) max.depth, the maximum tree depth; (3) subsample, the fraction of samples per tree; and (4) nrounds, the number of boosting trees to build.

For all six methods, k-fold cross-validation is implemented to tune classification parameters where the number of folds is set by argument nfolds.

Value

Returns an object of class tunecpfa with the following elements:

opt.model	List containing optimal model for tuned classification methods for each Parafac or Parafac2 model that was fit.
opt.param	Data frame containing optimal parameters for tuned classification methods.
kcv.error	Data frame containing KCV misclassification error for optimal parameters for tuned classification methods.
est.time	Data frame containing times for fitting Parafac or Parafac2 model and for tuning classification methods.
method	Numeric indicating classification methods used. Value of '1' indicates 'PLR'; value of '2' indicates 'SVM'; value of '3' indicates 'RF'; value of '4' indicates 'NN'; value of '5' indicates 'RDA'; and value of '6' indicates 'GBM'.

x	Three-way or four-way array used. If a list was used with model = "parafac2", returns list of matrices or three-way arrays used.
y	Factor containing class labels used. Note that output y is recoded such that the input labels of y are converted to numeric integers from 0 through the number of levels, which are then applied as labels for output y.
Aweights	List containing estimated A weights for each Parafac or Parafac2 model that was fit.
Bweights	List containing estimated B weights for each Parafac or Parafac2 model that was fit.
Cweights	List containing estimated C weights for each Parafac or Parafac2 model that was fit. Null if input argument x was a three-way array.
Phi	If model = "parafac2", a list containing estimated Phi from the Parafac2 model. Phi is the common cross product matrix shared by all levels of the last mode (see help file for function parafac2 in package **multiway** for additional details). NULL if model = "parafac".
const	Constraints used in fitting Parafac or Parafac2 models. If argument const was not provided, no constraints will be used.
cmode	Integer value of 1, 2, or 3 (or 4 if x is a four-way array) specifying mode whose component weights were predictors for classification.
family	Character value specifying whether classification was binary (family = "binomial") or multiclass (family = "multinomial").
xdim	Numeric value specifying number of levels for each mode of input x. If model = "parafac2", number of levels for first mode is designated as NA because the number of levels can differ across levels of the last mode.
lxdim	Numeric value specifying number of modes of input x.
train.weights	List containing classification component weights for each fit Parafac or Parafac2 model, for possibly different numbers of components. The weights used to train classifiers.
priorweights	List containing three elements based on input prior: (1) weight, inverse probability weights per observation, used to balance classes for PLR, NN, and GBM; (2) frac, inverse probability weights per class, used to balance classes for SVM; and (3) pricorrect, uniform priors used to balance classes for RF and RDA.

Note

For fitting the Parafac model, if argument cmode is not NULL, input array x is reshaped with function aperm such that the cmode dimension of x is ordered last. Estimated mode A and B (and mode C for a four-way array) weights that are outputted as Aweights and Bweights (and Cweights) reflect this permutation. For example, if x is a four-way array and cmode = 2, the original input modes 1, 2, 3, and 4 will correspond to output modes 1, 3, 4, 2. Here, output A = input 1; B = 3, and C = 4 (i.e., the second mode specified by cmode has been moved to the D mode/last mode). For model = "parafac2", classification mode is assumed to be the last mode (i.e., mode C for three-way array and mode D for four-way array).

In addition, note that the following combination of arguments will give an error: nfac = 1, family = "multinomial", method = "PLR". The issue arises from providing glmnet::cv.glmnet input x

with a matrix that has a single column. The issue is resolved for `family = "binomial"` because a column of 0s is appended to the single column, but this solution does not appear to work for the multiclass case. As such, this combination of arguments is not currently allowed. This issue will be resolved in a future update.

Applications of this function to real datasets can be explored at the following repository: https://github.com/matthewasisgress/multiway-classification/.

Author(s)

Matthew A. Asisgress <mattgress@protonmail.ch>

References

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., Li, Y., Yuan, J. (2025). xgboost: Extreme gradient boosting. R Package Version 1.7.9.1.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273-297.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. Annals of Statistics, 29(5), 1189-1232.

Friedman, J. H. (1989). Regularized discriminant analysis. Journal of the American Statistical Association, 84(405), 165-175.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software, 33(1), 1-22.

Gaujoux, R. (2025). doRNG: Generic reproducible parallel backend for 'foreach' loops. R Package Version 1.8.6.2.

Guo, Y., Hastie, T., and Tibshirani, R. (2007). Regularized linear discriminant analysis and its application in microarrays. Biostatistics, 8(1), 86-100.

Guo, Y., Hastie, T., and Tibshirani, R. (2023). rda: Shrunken centroids regularized discriminant analysis. R Package Version 1.2-1.

Harshman, R. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. UCLA Working Papers in Phonetics, 16, 1-84.

Harshman, R. (1972). PARAFAC2: Mathematical and technical notes. UCLA Working Papers in Phonetics, 22, 30-44.

Harshman, R. and Lundy, M. (1994). PARAFAC: Parallel factor analysis. Computational Statistics and Data Analysis, 18, 39-72.

Helwig, N. (2017). Estimating latent trends in multivariate longitudinal data via Parafac2 with functional and structural constraints. Biometrical Journal, 59(4), 783-803.

Helwig, N. (2025). multiway: Component models for multi-way data. R Package Version 1.0-7.

Liaw, A. and Wiener, M. (2002). Classification and regression by randomForest. R News 2(3), 18–22.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. (2024). e1071: Misc functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R Package Version 1.7-16.

Ripley, B. (1994). Neural networks and related methods for classification. Journal of the Royal Statistical Society: Series B (Methodological), 56(3), 409-437.

Venables, W. and Ripley, B. (2002). Modern applied statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.

Examples

```
########## Parafac example with 3-way array and binary response ##########
## Not run:
# set seed and simulate a three-way array connected to a binary response
set.seed(5)

# define list of arguments specifying distributions for A and G weights
techlist <- list(distA = list(dname = "poisson",
                              lambda = 3),                # for A weights
                 distG = list(dname = "gamma", shape = 2,
                              scale = 4))                 # for G weights

# define target correlation matrix for columns of C mode weights matrix
cormat <- matrix(c(1, .6, .6, .6, 1, .6, .6, .6, 1), nrow = 3, ncol = 3)

# simulate a three-way array connected to a response
data <- simcpfa(arraydim = c(11, 12, 100), model = "parafac", nfac = 3,
                nclass = 2, nreps = 1e2, onreps = 10, corresp = rep(.6, 3),
                meanpred = rep(2, 3), modes = 3, corrpred = cormat,
                technical = techlist, smethod = "eigende")

# initialize
alpha <- seq(0, 1, length = 2)
gamma <- c(0, 0.01)
cost <- c(1, 2)
ntree <- c(100, 200)
nodesize <- c(1, 2)
size <- c(1, 2)
decay <- c(0, 1)
rda.alpha <- c(0.1, 0.6)
delta <- c(0.1, 2)
eta <- c(0.3, 0.7)
max.depth <- c(1, 2)
subsample <- c(0.75)
nrounds <- c(100)
method <- c("PLR", "SVM", "RF", "NN", "RDA", "GBM")
family <- "binomial"
parameters <- list(alpha = alpha, gamma = gamma, cost = cost, ntree = ntree,
                   nodesize = nodesize, size = size, decay = decay,
                   rda.alpha = rda.alpha, delta = delta, eta = eta,
                   max.depth = max.depth, subsample = subsample,
                   nrounds = nrounds)
model <- "parafac"
nfolds <- 3
```

```
nstart <- 3

# constrain first mode weights to be orthogonal
const <- c("orthog", "uncons", "uncons")

# fit Parafac models and use third mode to tune classification methods
tune.object <- tunecpfa(x = data$X, y = as.factor(data$y), model = model,
                        nfac = 3, nfolds = nfolds, method = method,
                        family = family, parameters = parameters,
                        parallel = FALSE, const = const, nstart = nstart)

# print tuning object
tune.object

## End(Not run)
```

Index