An Introduction to cpfa

Matthew Asisgress

October 2025

Outline

Overview

Package **cpfa** implements a k-fold cross-validation procedure to predict class labels using component weights from a single mode of a Parallel Factor Analysis model-1 (Parafac; Harshman, 1970) or a Parallel Factor Analysis model-2 (Parafac2; Harshman, 1972), which is fit to a three-way or four-way data array. After fitting a Parafac or Parafac2 model with package **multiway** via an alternating least squares algorithm (Helwig, 2025), estimated component weights from one mode of this model are passed to one or more classification methods. For each method, a k-fold cross-validation is conducted to tune classification parameters using estimated Parafac component weights, optimizing class label prediction. This process is repeated over multiple train-test splits in order to improve the generalizability of results. Multiple constraint options are available to impose on any mode of the Parafac model during the estimation step (see Helwig, 2017). Multiple numbers of components can be considered in the primary package function `cpfa`. This vignette describes how to use the **cpfa** package.

Installation

cpfa can be installed directly from CRAN. Type the following command in an R console: `install.packages("cpfa", repos = "https://cran.r-project.org/")`. The argument `repos` can be modified according to user preferences. For more options and details, see `help(install.packages)`. In this case, the package **cpfa** has been downloaded and installed to the default directories. Users can download the package source at https://cran.r-project.org/package=cpfa and use Unix commands for installation.

Example 1: Four-way Array with Multiclass Response

We start by using the simulation function `simcpfa` and examining basic operations and outputs related to this function.

First, we load the **cpfa** package:

```
library(cpfa)
```

We simulate a four-way array where the fourth mode (i.e., the classification mode) of the simulated array is related to a response vector, which is also simulated. To generate data, we specify the data-generating

model as a Parafac2 model via the `model` argument and specify three components for this model with the `nfac` argument. We specify the number of dimensions for the simulated array using the `arraydim` argument. However, because a Parafac2 model is used, the function ignores the first element of `arraydim` and looks for input provided through the argument `pf2num` instead. Argument `pf2num` specifies the number of rows in each three-way array that exists within each level of the fourth mode of the four-way ragged array being simulated. As a demonstration, we set `pf2num <- rep(c(7, 8, 9), length.out = 100)`, which specifies that the number of rows alternates from 7, to 8, to 9, and back to 7, across all 100 levels of the fourth mode of the simulated array. Note that a useful feature of Parafac2 is that it can be fit to ragged arrays directly, while maintaining the intrinsic axis property of Parafac (see Harshman and Lundy, 1994).

For these simulated data, we specify that the response vector should have three classes using `nclass`. Moreover, we set a target correlation matrix, `corrpred`, specifying correlations among the columns of the classification mode's weight matrix (i.e., the fourth mode, in this case). We also specify correlations, contained in `corresp`, between columns of the classification mode's weight matrix and the response vector. Input `modes` sets the number of modes in the array; and we use `meanpred` to specify the target means for the columns of the classification mode weight matrix. Finally, `onreps` specifies the number of classification mode weight matrices to generate while `nreps` specifies, for any one classification mode weight matrix, the number of response vectors to generate (see `help(simcpfa)` for additional details of the simulation procedure). Then, in R we have the following:

```
# set seed for reproducibility
set.seed(500)

# specify correlation
cp <- 0.1

# define target correlation matrix for columns of fourth mode weight matrix
corrpred <- matrix(c(1, cp, cp, cp, 1, cp, cp, cp, 1), nrow = 3, ncol = 3)

# define correlations between fourth mode weight matrix and response vector
corresp <- rep(.85, 3)

# specify number of rows in the three-way array for each level of fourth mode
pf2num <- rep(c(7, 8, 9), length.out = 100)

# 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 = 10, onreps = 10, corresp = corresp,
                pf2num = pf2num, modes = 4, corrpred = corrpred,
                meanpred = c(10, 20, 30))

# define simulated array 'X' and response vector 'y' from the output
X <- data$X
y <- data$y
```

The above creates a four-way array `X` with Parafac2 structure that is connected through its fourth mode to response vector `y`. We confirm the dimensions of `X` and `y`, confirm their classes, and inspect the possible values of `y`:

```
# examine data object X
class(X)
```

```
## [1] "list"
```

```
length(X)
```

```
## [1] 100
```

```
dim(X[[1]])
```

```
## [1]  7 11 12
```

```
dim(X[[2]])
```

```
## [1]  8 11 12
```

```
# examine data object y
class(y)
```

```
## [1] "matrix" "array"
```

```
length(y)
```

```
## [1] 100
```

```
table(y)
```

```
## y
##  0  1  2
## 36 24 40
```

As shown, `X` is a list where each element is a three-way array. The dimensions of `X` match those specified in input arguments `arraydim` and `pf2num`. Likewise, `y` is a vector with length equal to the number of levels of the fourth mode of `X`. As desired, we can see that `y` contains three classes. However, note that no control currently exists to specify the proportions of output classes in `y`, which is a limitation of `simcpfa`. Future enhancements are planned to address this limitation.

We confirm that the columns of the fourth mode's weights are linearly associated with `y`:

```
# examine correlations between columns of fourth mode weights 'Dmat' and
# simulated response vector 'y'
cor(data$Dmat, data$y)
```

```
##           [,1]
## [1,] 0.3955246
## [2,] 0.3557359
## [3,] 0.4321371
```

As shown, the classification mode weight matrix `Dmat` contains columns that have a positive correlation with the response vector `y`. The target correlations (i.e., 0.85) were not achieved, especially given that `nreps = 10` and `onreps = 10` were small values. Nevertheless, achieved positive correlations indicate that building a classifier between `X` and `y` through the fourth mode of a three-component Parafac2 model could prove useful.

We initialize values for tuning parameter α from penalized logistic regression (PLR) implemented through package **glmnet** (Friedman, Hastie, and Tibshirani, 2010; Zou and Hastie, 2005). We specify the classification method as PLR through `method`, the model of interest as Parafac2 through `model`, the number of folds in the k-fold cross-validation step as three through `nfolds`, and the number of random starts for fitting the Parafac2 model as three through `nstart`. Further, we specify `nfac` to be two or three because we wish to explore classification performance for a two-component model and for a three-component model. The classification problem is multiclass, which is specified by setting `multinomial` for input `family`. In this demonstration, we allow for three train-test splits by setting `nrep <- 3` with a split ratio of `ratio <- 0.9`. We also specify pre-determined fold IDs for k-fold cross-validation using `foldid`. Finally, we set a Parafac2 model constraint: the fourth mode must have non-negative weights. We use `const` to set this constraint. In R:

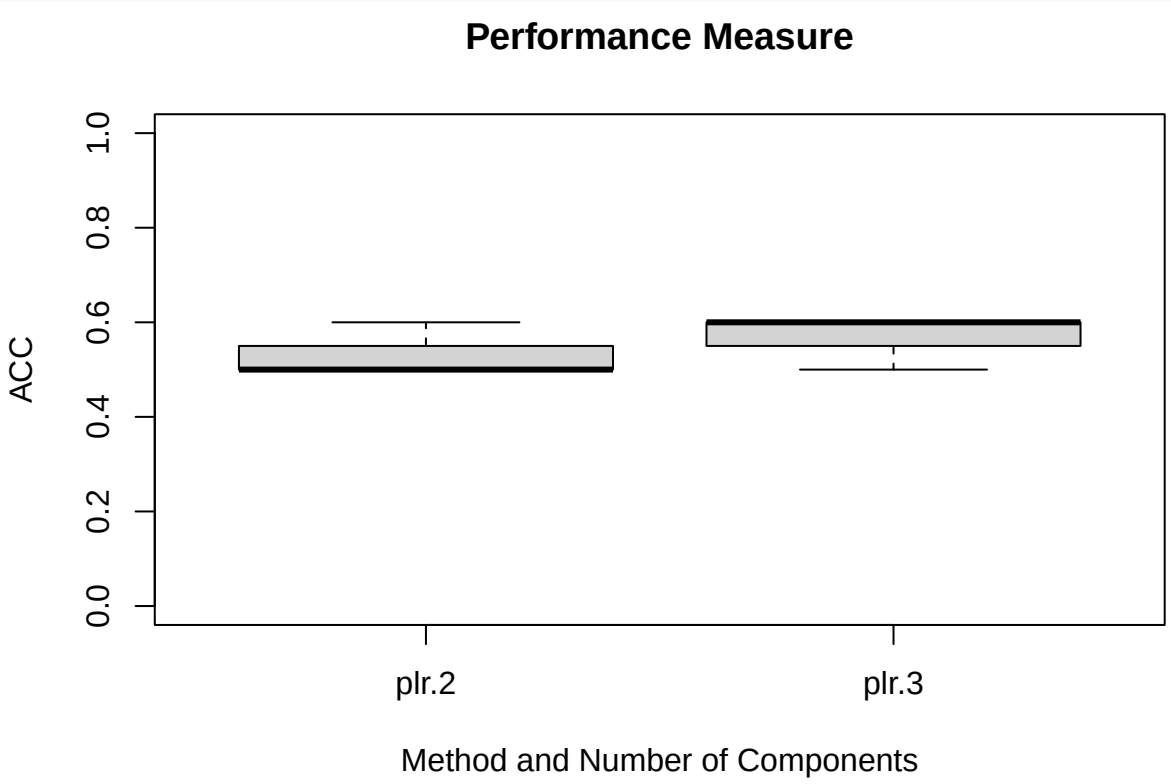
```
# set seed
set.seed(500)

# initialize alpha and store within a list called 'parameters'
```

```r
alpha <- seq(0, 1, length.out = 11)
parameters <- list(alpha = alpha)

# initialize inputs
method <- "PLR"
model <- "parafac2"
nfolds <- 3
nstart <- 3
nfac <- c(2, 3)
family <- "multinomial"
nrep <- 3
ratio <- 0.9
plot.out <- TRUE
const <- c("uncons", "uncons", "uncons", "nonneg")
foldid <- rep(1:nfolds, length.out = ratio * length(y))

# implement train-test splits with inner k-fold CV to optimize classification
output <- cpfa(x = X, y = as.factor(y), model = model, nfac = nfac,
               nrep = nrep, ratio = ratio, nfolds = nfolds, method = method,
               family = family, parameters = parameters, plot.out = plot.out,
               parallel = FALSE, const = const, foldid = foldid,
               nstart = nstart, verbose = FALSE)
```



The function generates box plots of classification accuracy for each number of components and for each classification method. In greater detail, we examine classification performance in the output object:

```r
# examine classification performance measures - median across train-test splits
output$descriptive$median[, 1:2]
```

```
##          err acc
```

```
## fac.2plr 0.5 0.5
## fac.3plr 0.4 0.6
```

As shown, classification accuracy (i.e., 'acc') is relatively good and certainly above baseline (for more details on classification performance measures, see help file for package function `cpm` via `help(cpm)`). In this case, the data-generating model with three components worked for classification purposes. We also examine, averaged across train-test splits, optimal tuning parameters. Note that **glmnet** optimized tuning parameter λ internally.

```
# examine optimal tuning parameters averaged across train-test splits
output$mean.opt.tune
```

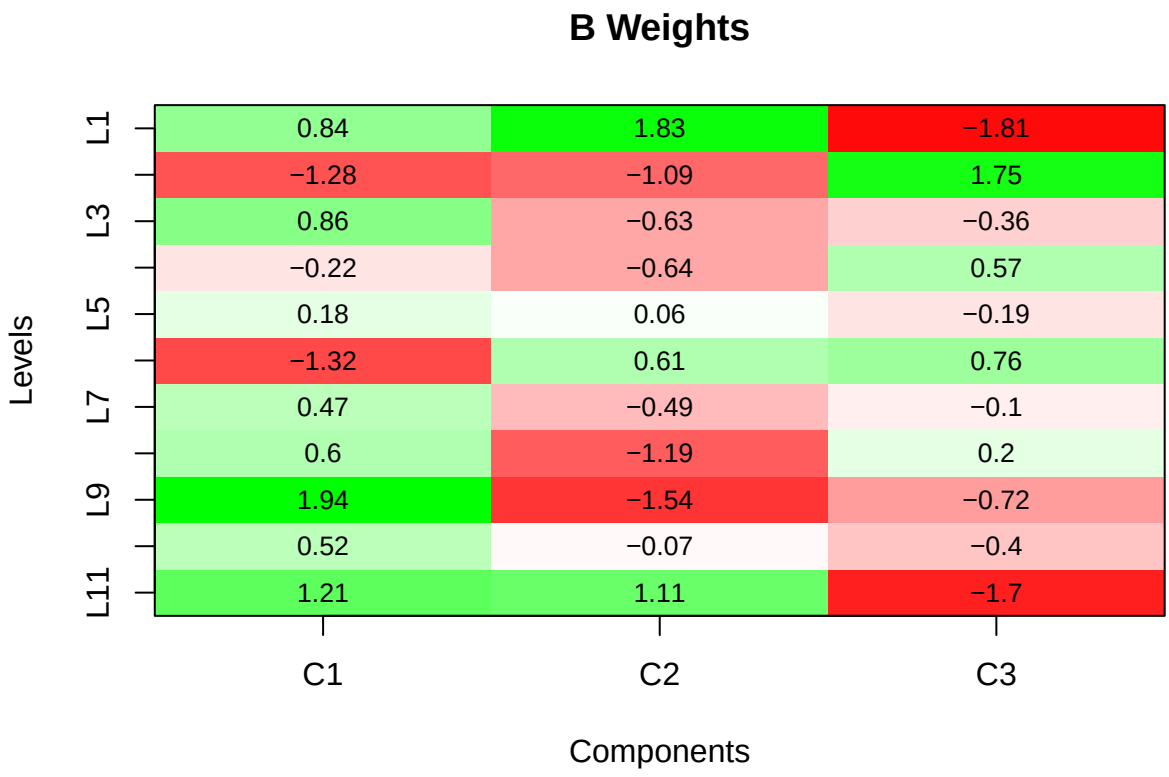
```
##   nfac      alpha     lambda gamma cost ntree nodesize size decay rda.alpha
## 1    2 0.03333333 974.374043    NA   NA    NA       NA   NA    NA        NA
## 2    3 0.06666667   9.291151    NA   NA    NA       NA   NA    NA        NA
##   delta eta max.depth subsample nrounds
## 1    NA  NA        NA        NA      NA
## 2    NA  NA        NA        NA      NA
```

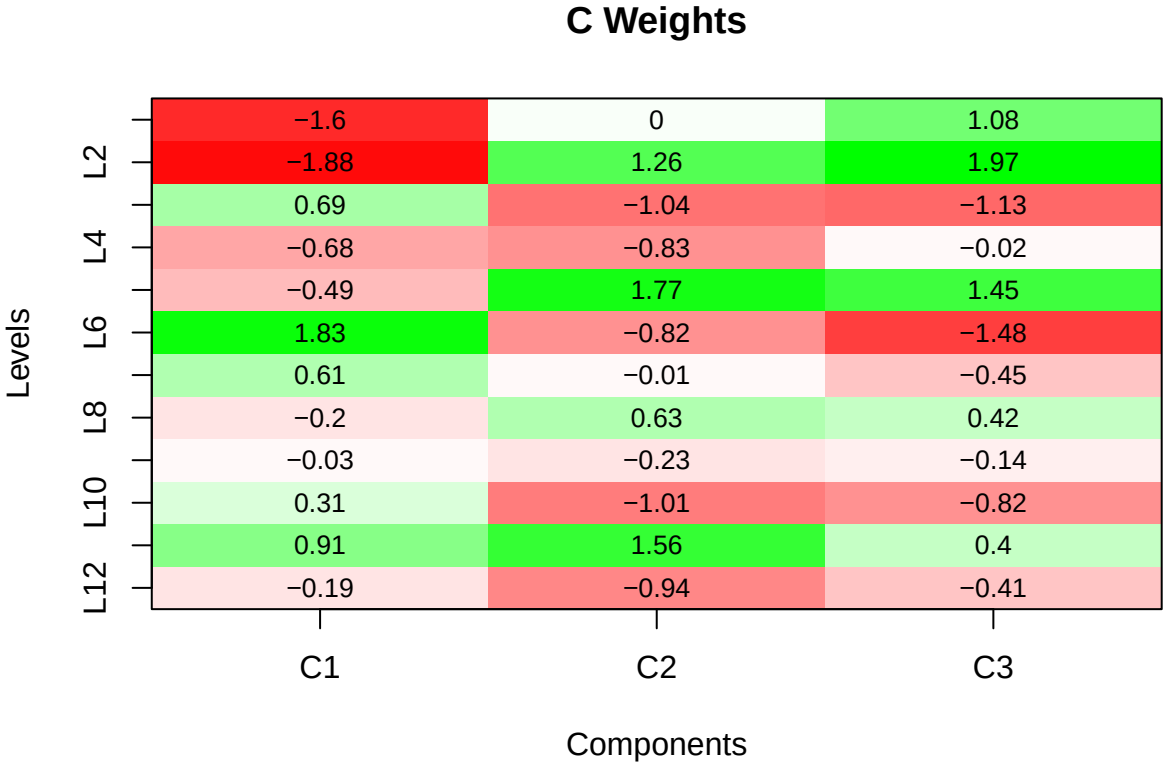
We can see average values for α that worked best. In addition, the best average λ values are also displayed. Note that other classifiers were not used in this demonstration, but their tuning parameters are indicated in the output with placeholders of NA.

We next use package function `plotcpfa` to fit the best (in terms of mean accuracy) Parafac2 model and to plot the results:

```
# set seed
set.seed(500)

# plot heat maps of component weights for optimal model
results <- plotcpfa(output, nstart = 3, ctol = 1e-1, verbose = FALSE)
```





Generated plots are heatmaps displaying the size of estimated component weights for the second (B) and third (C) modes. Because the input array was simulated, these plots do not display meaningful results but serve only to demonstrate the utility of function `plotcpfa` for visualizing the component weights of the optimal classification model (i.e., the model with the best classification performance, based on output from function `cpfa`). Thus, where function `cpfa` can serve as a guide to identify a meaningful number of components and a meaningful set of constraints for different modes, function `plotcpfa` can be used to visualize component weights of the best model to better understand how different levels of each mode map onto the set of components.

Example 2: Three-way Array with Binary Response

We now simulate a three-way array. For this array, the third mode is related linearly to a response vector of class labels, which is also simulated. To generate data using function `simcpfa`, we specify the data-generating model as a Parafac model via the `model` argument and specify two components for this model with the `nfac` argument. We specify the number of dimensions for the simulated array using the `arraydim` argument.

For these simulated data, we specify that the response vector should have two classes using `nclass`. Moreover, we set a target correlation matrix, `corrpred`, specifying correlations among the columns of the classification mode's weight matrix (i.e., in this case, the third mode). We also specify correlations, contained in `corresp`, between columns of the classification mode's weight matrix and the response vector.

```
# set seed for reproducibility
set.seed(400)

# specify correlation
cp <- 0.1

# define target correlation matrix for columns of third mode weight matrix
corrpred <- matrix(c(1, cp, cp, 1), nrow = 2, ncol = 2)

# define correlations between third mode weight matrix and response vector
```

```
corresp <- rep(.9, 2)

# simulate a three-way array connected to a binary response
data <- simcpfa(arraydim = c(10, 11, 100), model = "parafac", nfac = 2,
                nclass = 2, nreps = 10, onreps = 10, corresp = corresp,
                modes = 3, corrpred = corrpred, meanpred = c(10, 20))

# define simulated array 'X' and response vector 'y' from the output
X <- data$X
y <- data$y
```

The above creates a three-way array `X` with Parafac structure that is connected through its third mode to response vector `y`. We confirm the dimensions of `X` and `y`, confirm their classes, and inspect the possible values of `y`:

```
# examine data object X
class(X)
```

```
## [1] "array"
```

```
dim(X)
```

```
## [1]  10  11 100
```

```
# examine data object y
class(y)
```

```
## [1] "matrix" "array"
```

```
length(y)
```

```
## [1] 100
```

```
table(y)
```

```
## y
##  0  1
## 52 48
```

As shown, `X` is a three-way array. The dimensions of `X` match those specified in input argument `arraydim`. Likewise, `y` is a vector with length equal to the number of levels of the third mode of `X`. As desired, we can see that `y` contains two classes. As in the last example, we see that the output classes in `y` are imbalanced.

We confirm that the columns of the third mode's weights are linearly associated with `y`:

```
# examine correlations between columns of third mode weights 'Cmat' and
# simulated response vector 'y'
cor(data$Cmat, data$y)
```

```
##           [,1]
## [1,] 0.4962389
## [2,] 0.3457724
```

As shown, the classification mode weight matrix `Cmat` contains columns that have a positive correlation with the response vector `y`. The target correlations (i.e., 0.9) were not achieved, especially given that `nreps = 10` and `onreps = 10` were small values. Nevertheless, the positive correlations indicate that building a classifier between `X` and `y` through the third mode of a two-component Parafac model could prove useful.

We initialize values for tuning parameter α from PLR implemented through package **glmnet**. We also initialize values for the tuning parameters `ntree` (i.e., number of trees) and `nodesize` (i.e., node size) from

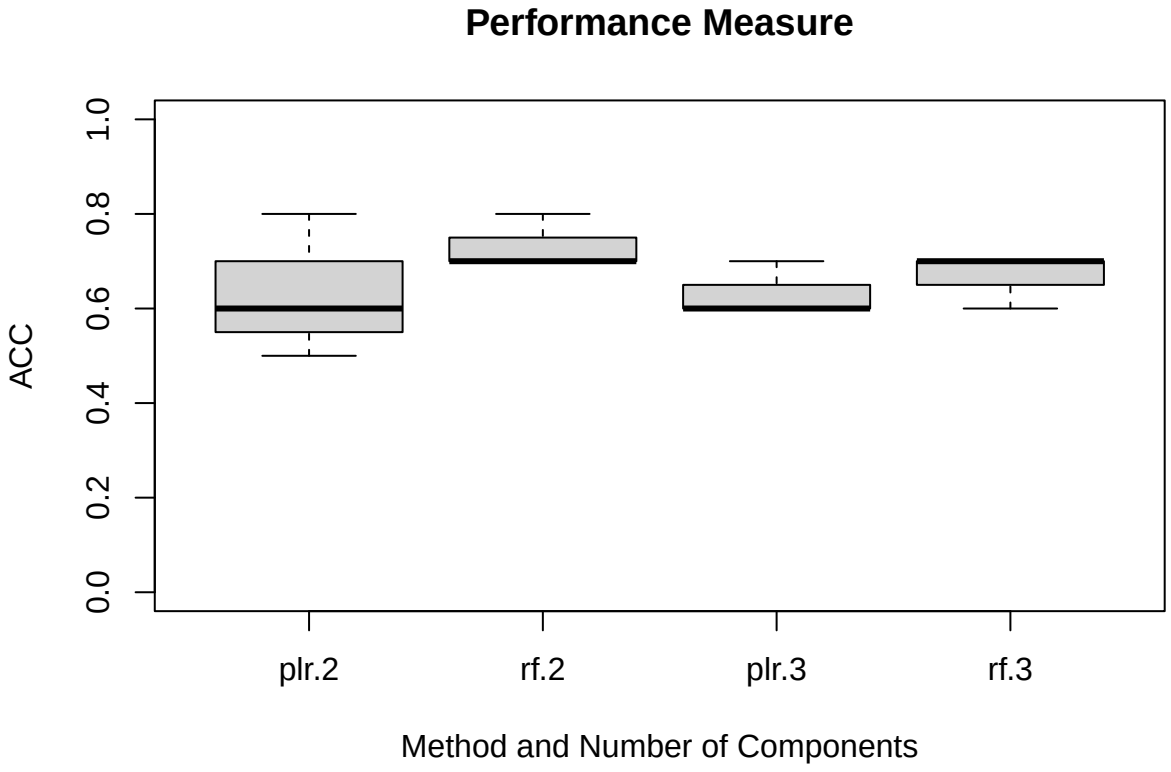
random forest (RF; Breiman, 2001) implemented through package **randomForest** (Liaw and Wiener, 2002). We specify the classification methods as PLR and RF through `method`, the model of interest as Parafac through `model`, the number of folds in the k-fold cross-validation step as three through `nfolds`, and the number of random starts for fitting the Parafac model as three through `nstart`. Further, we specify `nfac` to be two or three because we wish to explore classification performance for a two-component model and for a three-component model. The classification problem is binary, which is specified by setting `binomial` for input `family`. We allow for three train-test splits by setting `nrep <- 3` with a split ratio of `ratio <- 0.9`. Finally, for this demonstration, we set a Parafac model constraint: the second mode must have orthogonal weights. We use `const` to set this constraint. In R:

```r
# set seed
set.seed(300)

# initialize tuning parameters and store within a list called 'parameters'
alpha <- seq(0, 1, length.out = 3)
ntree <- c(200, 400)
nodesize <- c(2, 4)
parameters <- list(alpha = alpha, ntree = ntree, nodesize = nodesize)

# initialize inputs
method <- c("PLR", "RF")
model <- "parafac"
nfolds <- 3
nstart <- 3
nfac <- c(2, 3)
family <- "binomial"
nrep <- 3
ratio <- 0.9
plot.out <- TRUE
const <- c("uncons", "orthog", "uncons")

# implement train-test splits with inner k-fold CV to optimize classification
output <- cpfa(x = X, y = as.factor(y), model = model, nfac = nfac,
               nrep = nrep, ratio = ratio, nfolds = nfolds, method = method,
               family = family, parameters = parameters, plot.out = plot.out,
               parallel = FALSE, const = const, nstart = nstart,
               verbose = FALSE)
```



The function generates box plots of classification accuracy for each number of components and for each classification method. In greater detail, we examine classification performance in the output object:

```
# examine classification performance measures - median across train-test splits
output$descriptive$mean[, 1:2]
```

```
##                err       acc
## fac.2plr 0.3666667 0.6333333
## fac.2rf  0.2666667 0.7333333
## fac.3plr 0.3666667 0.6333333
## fac.3rf  0.3333333 0.6666667
```

As shown, classification accuracy is highest for the two-component RF classifer. In this case, the data-generating model with two components worked best for classification purposes (i.e, compared to the three-component model). We also examine, averaged across train-test splits, optimal tuning parameters.

```
# examine optimal tuning parameters averaged across train-test splits
output$mean.opt.tune
```

```
##   nfac     alpha    lambda gamma cost    ntree nodesize size decay rda.alpha
## 1    2 0.0000000 0.1655723    NA   NA 266.6667 3.333333   NA    NA        NA
## 2    3 0.1666667 0.2013264    NA   NA 266.6667 3.333333   NA    NA        NA
##   delta eta max.depth subsample nrounds
## 1    NA  NA        NA        NA      NA
## 2    NA  NA        NA        NA      NA
```

We can see average tuning values that worked best. For example, PLR favored $\alpha = 0$ for the two-component model (i.e., preferred ridge regression).

We next could use the function `plotcpfa` to fit the best Parafac model and to plot the results. The code looks like this:

```r
# set seed
set.seed(400)

# plot heat maps of component weights for optimal model
results <- plotcpfa(output, nstart = 3, ctol = 1e-1, verbose = FALSE)
```

However, as in the previous example, the simulated array contains mode A and mode B weights whose levels do not have a substantive meaning. As such, we omit the heat maps for this example. However, if these were real data, such heat maps might reveal relationships between model components and the levels of mode A or B. Interested readers might explore the application of this package to popular, real data sets used in classification research, such as the MNIST data set (LeCun et al., 2002).

Concluding Thoughts

Package **cpfa** implements a k-fold cross-validation procedure, connecting Parafac models fit by **multiway** to classification methods implemented through six popular packages used for classification: **glmnet**; **e1071** (Meyer et al., 2024; Cortes and Vapnik, 1995); **randomForest** (Liaw and Wiener, 2002; Breiman, 2001); **nnet** (Ripley, 1994; Venables and Ripley, 2002); **rda** (Guo, Hastie, and Tibshirani, 2007, 2023; Friedman, 1989), and **xgboost** (Chen et al., 2025; Friedman, 2001). Parallel computing is implemented through packages **parallel** (R Core Team, 2025) and **doParallel** (Microsoft Corporation and Weston, 2022). The example above highlights the use of **cpfa** and three of its functions. For more information about the package, see https://CRAN.R-project.org/package=cpfa or examine package help files with `help(simcpfa)`, `help(cpfa)`, or `help(plotcpfa)`.

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

Friedman, J. (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.

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.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (2002). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

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.

Microsoft Corporation and Weston, S. (2022). doParallel: foreach parallel adaptor for the 'parallel' package. R Package Version 1.0.17.

R Core Team (2025). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

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.