An Introduction to cpfa

Matthew Asisgress

June 2026

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. The package also supports principal component analysis (PCA) applied to a two-way data matrix. After fitting a Parafac or Parafac2 model with package **multiway** via an alternating least squares algorithm (Helwig, 2025), or after fitting a PCA model using the singular value decomposition, estimated component weights from one mode of the selected component 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 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 or Parafac2 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:

```r
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. Both of these arguments work when `smethod = logistic`, which implements an iterative Monte Carlo rejection sampling procedure based on the generalized linear model (see `help(simcpfa)` for additional details of the simulation procedure). Then, in R we have the following:

```r
# 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), smethod = "logistic")

# 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] "factor"
```

```
length(y)
```

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

```
table(y)
```

```
## y
##  0  1  2
## 36 23 41
```

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 factor with length equal to the number of levels of the fourth mode of `X`. As desired, we can see that `y` contains three classes.

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$Dmat, (as.numeric(data$y) - 1))
```

```
##           [,1]
## [1,] 0.4215966
## [2,] 0.3753194
## [3,] 0.4235106
```

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`. 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'
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
const <- c("uncons", "uncons", "uncons", "nonneg")

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

The function can generate box plots of classification accuracy for each number of components and for each classification method, but we set `plot.out = FALSE` and do not produce them. We examine classification performance in the output object:

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

```
##          err acc
## fac.2plr 0.3 0.7
## fac.3plr 0.3 0.7
```

As shown, classification accuracy (i.e., 'acc') is relatively fair (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 best 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 delta
## 1    2 0.5666667 0.2325063    NA   NA    NA       NA   NA    NA        NA    NA
## 2    3 0.4666667 0.2136389    NA   NA    NA       NA   NA    NA        NA    NA
##   eta max.depth subsample nrounds
## 1  NA        NA        NA      NA
## 2  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.

Next, we could use the package function `plotcpfa` to fit the best (in terms of mean accuracy) Parafac2 model and to plot the results. The code looks like this:

```
# 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 heat maps displaying the magnitude of estimated component weights for the second (B) and third (C) modes. Because the input array was simulated, these plots would not display meaningful results. As such, we omit the heat maps for this example. For such plots, while `cpfa` can serve as a guide to identify a meaningful number of components and a meaningful set of constraints for different modes, the function `plotcpfa` can be used to visualize component weights of the best model to understand how levels of each mode map onto components. To see heat maps, interested readers might explore the application of this package to several real datasets. See here for examples: https://github.com/matthewasisgress/multiway-classification/.

To obtain a measure of feature importance among features in the model, we calculate permutation feature importance (PFI; see Breiman, 2001). The function `pficpfa` calculates PFI for an object from function `cpfa`.

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

# calculate permutation feature importance for each feature
pfistats <- pficpfa(object = output, nshuffles = 5, type = "marginal")

# examine median pfi for classification accuracy and for nfac = 2
pfistats[which((pfistats$nfac == 2) & (pfistats$metric == "acc")), ]
```

```
##    nfac feature method metric      mean median         sd n nvalid
## 2     2       1    PLR    acc 0.1933333   0.24 0.09865766 3      3
## 13    2       2    PLR    acc 0.0800000   0.12 0.14422205 3      3
```

As shown, we can observe the mean or median drop in classification accuracy, pooled across replications, after permuting observations for each feature.

Alternatively, we can set `type = conditional` to fit a conditioning model to predict each feature from the others; and the model's residuals are permuted and added back to the fitted values, preserving the feature's dependence on the other features (for more information, see Huang, 2025, and O'Gorman, 2005). We can choose random forest as the conditioning model by setting `conditional.model = "rf"`.

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

# calculate permutation feature importance for each feature
pfistats <- pficpfa(object = output, nshuffles = 5, type = "conditional",
                    conditional.model = "rf")

# examine median pfi for classification accuracy and for nfac = 2
pfistats[which((pfistats$nfac == 2) & (pfistats$metric == "acc")), ]
```

```
##    nfac feature method metric       mean median        sd n nvalid
## 2     2       1    PLR    acc 0.08000000   0.04 0.1249000 3      3
## 13    2       2    PLR    acc 0.04666667   0.08 0.1137248 3      3
```

As shown, the mean and median importance calculations are lower. Conditional permutation feature importance is useful when features are correlated.

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),
                smethod = "logistic")

# 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] "factor"
```

```
length(y)
```

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

```
table(y)
```

```
## y
##  0  1
```

```
## 43 57
```

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

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$Cmat, (as.numeric(data$y) - 1))
```

```
##           [,1]
## [1,] 0.4951718
## [2,] 0.3820043
```

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:

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

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

```
                parallel = FALSE, const = const, nstart = nstart,
                verbose = FALSE)
```

We examine classification performance in the output object:

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

```
##                err       acc
## fac.2plr 0.2000000 0.8000000
## fac.2rf  0.2333333 0.7666667
## fac.3plr 0.1666667 0.8333333
## fac.3rf  0.3000000 0.7000000
```

As shown, classification accuracy is highest for the two-component RF classifier. 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 0.3333333 0.12588858    NA   NA 266.6667 2.666667   NA    NA        NA
## 2    3 0.3333333 0.09947448    NA   NA 266.6667 2.666667   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:

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

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

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 could reveal relationships between model components and the levels of mode A or B. As previously described, interested readers might explore the application of this package to several real datasets. See here for more examples: https://github.com/matthewasisgress/multiway-classification/.

Example 3: Two-way Matrix with Multiclass Response

We now simulate a two-way matrix. For this matrix, the second 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 PCA 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 matrix using the `arraydim` argument. For a matrix only, the classification mode is assumed to be the first mode while the mode containing variables/predictors is assumed to be the second mode.

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., in this case, the first mode). We also specify correlations, contained in `corresp`,

between columns of the classification mode's weight matrix and the response vector. For this example, we use the second simulation method `eigende`, which simulates component weights and a latent response variable simultaneously from a joint multivariate normal distribution using an eigendecomposition, discretizing the response into class labels using quantile cuts defined by the cumulative sum of `props`. Without specifying `props`, this argument defaults to equal class proportions. We ensure that argument `modes` is set to 2.

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

# specify correlation
cp <- 0.1

# define target correlation matrix for columns of first mode weight matrix
a <- matrix(cp, nrow = 3, ncol = 3); diag(a) <- 1; corrpred <- a

# define correlations between first mode weight matrix and response vector
corresp <- rep(.5, 3)

# simulate a two-way matrix connected to a multiclass response
data <- simcpfa(arraydim = c(200, 10), model = "pca", nfac = 3,
                nclass = 3, corresp = corresp, modes = 2, corrpred = corrpred,
                smethod = "eigende")

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

The above creates a two-way matrix `X` with PCA structure that is connected through its second 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] "matrix" "array"
```

```
dim(X)
```

```
## [1] 200  10
```

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

```
## [1] "factor"
```

```
length(y)
```

```
## [1] 200
```

```
table(y)
```

```
## y
##  0  1  2
## 67 66 67
```

As shown, `X` is a two-way matrix. The dimensions of `X` match those specified in input argument `arraydim`. Likewise, `y` is a factor with length equal to the number of levels of the first mode of `X`. As desired, we can see that `y` contains three classes. We see that the output classes in `y` are balanced.

We confirm that the columns of the first mode's weights are linearly associated with `y` (note: while `Bmat` is

associated with the second mode in Parafac and Parafac2 models, it is associated with the *first* mode in PCA for the `simcpfa` function):

```
# examine correlations between columns of first mode weights 'Bmat' and
# simulated response vector 'y'
cor(data$Bmat, (as.numeric(data$y) - 1))
```

```
##           [,1]
## [1,] 0.5342150
## [2,] 0.5780482
## [3,] 0.5196194
```

As shown, the classification mode weight matrix `Bmat` contains columns that have a positive correlation with the response vector `y`. The target correlations (i.e., 0.5) were nearly achieved. The positive correlations indicate that building a classifier between `X` and `y` through the first mode of a three-component PCA model could prove useful. Again, for emphasis, while `Bmat` is associated with the second mode in Parafac and Parafac2 models, it is associated with the *first* mode in PCA for this function.

As before, we initialize values for tuning parameter α from PLR implemented through package **glmnet**. We specify the model of interest as PCA through `model`. We specify other arguments in a fashion similar to the previous example (i.e., Example 2). We also specify that we wish to apply a Varimax rotation to the loadings matrix for the PCA solution by setting `pcarot = "varimax"`. In R:

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

# initialize tuning parameters and store within a list called 'parameters'
alpha <- seq(0, 1, length.out = 5)
parameters <- list(alpha = alpha)

# initialize inputs
method <- c("PLR")
model <- "pca"
nfolds <- 3
nfac <- c(2, 3)
family <- "multinomial"
nrep <- 3
ratio <- 0.9

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

We examine classification performance in the output object:

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

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

As shown, classification accuracy is highest for the three-component PLR classifier. In this case, the data-generating model with three components worked best for classification purposes (i.e., compared to the two-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 0.5000000 0.08044747    NA   NA    NA       NA   NA    NA        NA
## 2    3 0.4166667 0.06318615    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 tuning values that worked best. We next can use the function `plotcpfa` to fit the best PCA model and to plot the results. The R code is provided but not evaluated:

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

# plot heat maps of component weights for optimal model
results <- plotcpfa(output)
```

The simulated matrix contains mode A as the loadings and mode B as the scores. As in the previous example, the 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 could reveal relationships between model components and the levels of mode A (i.e., for PCA, the loadings).

Concluding Thoughts

Package **cpfa** implements a k-fold cross-validation procedure, connecting Parafac and Parafac2 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); **doParallel** (Microsoft Corporation and Weston, 2022); and **doRNG** (Gaujoux, 2025). The package also supports principal component analysis applied to a matrix. The examples above highlight 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., and Yuan, J. (2025). xgboost: Extreme gradient boosting. R Package Version 1.7.9.1.

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

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.

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.

Hornik, K. (2005). A CLUE for CLUster ensembles. Journal of Statistical Software, 14, 1-25.

Hornik, K. (2026). clue: Cluster ensembles. R Package Version 0.3-68.

Huang, P. (2025). Residual permutation tests for feature importance in machine learning. British Journal of Mathematical and Statistical Psychology.

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

O'Gorman, T. (2005). The performance of randomization tests that use permutations of independent variables. Communications in Statistics - Simulation and Computation, 34(4), 895-908.

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.