Package 'CCMMR'

July 21, 2026

Type Package

Title Minimization of the Convex Clustering Loss Function

Version 0.2.3

Date 2026-07-19

Maintainer Daniel Touw <touw@ese.eur.nl>

Description Implements the convex clustering through majorization-minimization (CCMM) algorithm described in Touw, Groenen, and Terada (2022) <doi:10.48550/arXiv.2211.01877> to perform minimization of the convex clustering loss function.

License GPL (>= 3)

RoxygenNote 7.3.2

URL https://github.com/djwtouw/CCMMR/

BugReports https://github.com/djwtouw/CCMMR/issues/

Imports RANN (>= 2.6.1), Rcpp (>= 1.0.7), methods (>= 4.1.0), graphics (>= 4.1.0)

LinkingTo Rcpp, RcppEigen

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Depends R (>= 4.1), stats (>= 4.1)

Author Daniel Touw [aut, cre] (ORCID: <https://orcid.org/0000-0003-3074-5401>), Patrick Groenen [aut] (ORCID: <https://orcid.org/0000-0001-6683-8971>), Yoshikazu Terada [aut]

Repository CRAN

Date/Publication 2026-07-21 08:40:02 UTC

Contents

as.hclust.cvxclust 2
clusters 3

convex_clustering 4
convex_clusterpath 7
plot.cvxclust 9
sparse_weights 10
two_half_moons 11

Index **12**

`as.hclust.cvxclust` *Conversion of a* `cvxclust` *object into an* `hclust` *object*

Description

Converts the output of convex_clustering or convex_clusterpath into a hclust object. Note that a step in the clusterpath from one value for lambda to the next may cause the number of clusters to decrease by more than one. It is a hard requirement that the clusterpath ends in a single cluster, as standard dendrogram plotting methods fail if this is not the case.

Usage

```
## S3 method for class 'cvxclust'
as.hclust(x, ...)
```

Arguments

`x`	A `cvxclust` object.
`...`	Unused.

Value

A `hclust` object.

See Also

hclust

Examples

```
# Demonstration of converting a clusterpath into a dendrogram, first generate
# data
set.seed(6)
X = matrix(rnorm(14), ncol = 2)
y = rep(1, nrow(X))

# Get sparse distances in dictionary of keys format with k = 3
W = sparse_weights(X, 3, 4.0)

# Sequence for lambda
lambdas = seq(0, 45, 0.02)
```

```
# Compute results
res = convex_clusterpath(X, W, lambdas)

# Generate hclust object
hcl = as.hclust(res)
hcl$height = sqrt(hcl$height)

# Plot clusterpath and dendrogram
oldpar = par(mfrow=c(1, 2))
plot(res, y, label = c(1:7))
plot(hcl, ylab = expression(sqrt(lambda)), xlab = NA, sub = NA, main = NA,
     hang = -1)
par(oldpar)
```

`clusters` *Obtain clustering from a clusterpath*

Description

Get a particular clustering of the data. If there is a clustering for `n_clusters`, it is returned, otherwise the function will stop with a message. To know whether a query is going to be successful beforehand, check the `num_clusters` attribute of the `cvxclust` object, this lists all possible options for the number of clusters.

Usage

```
clusters(obj, n_clusters)
```

Arguments

`obj`	A `cvxclust` object.
`n_clusters`	An integer that specifies the number of clusters that should be returned.

Value

A vector with the cluster labels for each object in the data.

Examples

```
# Load data
data(two_half_moons)
data = as.matrix(two_half_moons)
X = data[, -3]
y = data[, 3]

# Get sparse distances in dictionary of keys format with k = 5 and phi = 8
W = sparse_weights(X, 5, 8.0)
```

```
# Set a sequence for lambda
lambdas = seq(0, 2400, 1)

# Compute results CMM
res = convex_clusterpath(X, W, lambdas)

# Get labels for three clusters
labels = clusters(res, 3)
```

---

convex_clustering *Find a target number of clusters in the data using convex clustering*

Description

convex_clustering attempts to find the number of clusters specified by the user by means of convex clustering. The algorithm looks for each number of clusters between target_low and target_high. If target_low = target_high, the algorithm searches for a single clustering. It is recommended to specify a range around the desired number of clusters, as not each number of clusters between 1 and nrow(X) may be attainable due to numerical inaccuracies.

Usage

```
convex_clustering(
  X,
  W,
  target_low,
  target_high = NULL,
  max_iter_phase_1 = 2000,
  max_iter_phase_2 = 20,
  lambda_init = 0.01,
  factor = 0.025,
  tau = 0.001,
  center = TRUE,
  scale = TRUE,
  eps_conv = 1e-06,
  burnin_iter = 25,
  max_iter_conv = 5000,
  save_clusterpath = FALSE,
  verbose = 0
)
```

Arguments

X	An n x p numeric matrix. This function assumes that each row represents an object with p attributes.
W	A sparseweights object, see sparse_weights.

target_low	Lower bound on the number of clusters that should be searched for. If target_high = NULL, this is the exact number of clusters that is searched for.
target_high	Upper bound on the number of clusters that should be searched for. Default is NULL, in that case, it is set equal to target_low.
max_iter_phase_1	Maximum number of iterations to find an upper and lower bound for the value for lambda for which the desired number of clusters is attained. Default is 2000.
max_iter_phase_2	Maximum number of iterations to to refine the upper and lower bounds for lambda. Default is 20.
lambda_init	The first value for lambda other than 0 to use for convex clustering. Default is 0.01.
factor	The percentage by which to increase lambda in each step. Default is 0.025.
tau	Parameter to compute the threshold to fuse clusters. Default is 0.001.
center	If TRUE, center X so that each column has mean zero. Default is TRUE.
scale	If TRUE, scale the loss function to ensure that the cluster solution is invariant to the scale of X. Default is TRUE. Not recommended to set to FALSE unless comparing to algorithms that minimize the unscaled convex clustering loss function.
eps_conv	Parameter for determining convergence of the minimization. Default is 1e-6.
burnin_iter	Number of updates of the loss function that are done without step doubling. Default is 25.
max_iter_conv	Maximum number of iterations for minimizing the loss function. Default is 5000.
save_clusterpath	If TRUE, store the solution that minimized the loss function for each lambda. Is required for drawing the clusterpath. Default is FALSE. To store the clusterpath coordinates, n x p x $no.lambdas$ values have to be stored, this may require too much memory for large data sets.
verbose	Verbosity of the information printed during clustering. Default is 0, no output.

Value

A cvxclust object containing the following

info	A dataframe containing for each value for lambda: the number of different clusters, and the value of the loss function at the minimum.
merge	The merge table containing the order at which the observations in X are clustered.
height	The value for lambda at which each reduction in the number of clusters occurs.
order	The order of the observations in X in order to draw a dendrogram without conflicting branches.
elapsed_time	The number of seconds that elapsed while running the code. Note that this does not include the time required for input checking and possibly scaling and centering X.
coordinates	The clusterpath coordinates. Only part of the output in case that save_clusterpath=TRUE.

lambdas	The values for lambda for which a clustering was found.
eps_fusions	The threshold for cluster fusions that was used by the algorithm.
phase_1_instances	The number of instances of the loss function that were minimized while finding an upper and lower bound for lambda. The sum phase_1_iterations + phase_2_iterations gives the total number of instances solved.
phase_2_instances	The number of instances of the loss function that were minimized while refining the value for lambda. The sum phase_1_iterations + phase_2_iterations gives the total number of instances solved.
num_clusters	The different numbers of clusters that have been found.
n	The number of observations in X.

See Also

convex_clusterpath, sparse_weights

Examples

```
# Load data
data(two_half_moons)
data = as.matrix(two_half_moons)
X = data[, -3]
y = data[, 3]

# Get sparse weights in dictionary of keys format with k = 5 and phi = 8
W = sparse_weights(X, 5, 8.0)

# Perform convex clustering with a target number of clusters
res1 = convex_clustering(X, W, target_low = 2, target_high = 5)

# Plot the clustering for 2 to 5 clusters
oldpar = par(mfrow=c(2, 2))
plot(X, col = clusters(res1, 2), main = "2 clusters", pch = 19)
plot(X, col = clusters(res1, 3), main = "3 clusters", pch = 19)
plot(X, col = clusters(res1, 4), main = "4 clusters", pch = 19)
plot(X, col = clusters(res1, 5), main = "5 clusters", pch = 19)

# A more generalized approach to plotting the results of a range of clusters
res2 = convex_clustering(X, W, target_low = 2, target_high = 7)

# Plot the clusterings
k = length(res2$num_clusters)
par(mfrow=c(ceiling(k / ceiling(sqrt(k))), ceiling(sqrt(k))))

for (i in 1:k) {
    labels = clusters(res2, res2$num_clusters[i])
    c = length(unique(labels))

    plot(X, col = labels, main = paste(c, "clusters"), pch = 19)
```

```
}
par(oldpar)
```

`convex_clusterpath` *Minimize the convex clustering loss function*

Description

Minimizes the convex clustering loss function for a given set of values for lambda.

Usage

```
convex_clusterpath(
  X,
  W,
  lambdas,
  tau = 0.001,
  center = TRUE,
  scale = TRUE,
  eps_conv = 1e-06,
  burnin_iter = 25,
  max_iter_conv = 5000,
  save_clusterpath = TRUE,
  target_losses = NULL,
  save_losses = FALSE,
  save_convergence_norms = FALSE
)
```

Arguments

`X`	An n x p numeric matrix. This function assumes that each row represents an object with p attributes.
`W`	A `sparseweights` object, see sparse_weights.
`lambdas`	A vector containing the values for the penalty parameter.
`tau`	Parameter to compute the threshold to fuse clusters. Default is 0.001.
`center`	If `TRUE`, center `X` so that each column has mean zero. Default is `TRUE`.
`scale`	If `TRUE`, scale the loss function to ensure that the cluster solution is invariant to the scale of `X`. Default is `TRUE`. Not recommended to set to `FALSE` unless comparing to algorithms that minimize the unscaled convex clustering loss function.
`eps_conv`	Parameter for determining convergence of the minimization. Default is 1e-6.
`burnin_iter`	Number of updates of the loss function that are done without step doubling. Default is 25.
`max_iter_conv`	Maximum number of iterations for minimizing the loss function. Default is 5000.

save_clusterpath
: If TRUE, store the solution that minimized the loss function for each lambda. Is required for drawing the clusterpath. Default is FALSE. To store the clusterpath coordinates, n x p x $no.lambdas$ have to be stored, this may require too much memory for large data sets.

target_losses
: The values of the loss function that are used to determine convergence of the algorithm (tested as: loss - target <= eps_conv * target). If the input is not NULL, it should be a vector with the same length as lambdas. Great care should be exercised to make sure that the target losses correspond to attainable values for the minimization. The inputs (X, W, lambdas) should be the same, but also the same version of the loss function (centered, scaled) should be used. Default is NULL.

save_losses
: If TRUE, return the values of the loss function attained during minimization for each value of lambda. Default is FALSE.

save_convergence_norms
: If TRUE, return the norm of the difference between consecutive iterates during minimization for each value of lambda. Default is FALSE. If timing the algorithm is of importance, do not set this to TRUE, as additional computations are done for bookkeeping that are irrelevant to the optimization.

Value

A cvxclust object containing the following

info
: A dataframe containing for each value for lambda: the number of different clusters, and the value of the loss function at the minimum.

merge
: The merge table containing the order at which the observations in X are clustered.

height
: The value for lambda at which each reduction in the number of clusters occurs.

order
: The order of the observations in X in order to draw a dendrogram without conflicting branches.

elapsed_time
: The number of seconds that elapsed while running the code. Note that this does not include the time required for input checking and possibly scaling and centering X.

coordinates
: The clusterpath coordinates. Only part of the output in case that save_clusterpath=TRUE.

lambdas
: The values for lambda for which a clustering was found.

eps_fusions
: The threshold for cluster fusions that was used by the algorithm.

num_clusters
: The different numbers of clusters that have been found.

n
: The number of observations in X.

losses
: Optional: if save_losses = TRUE, the values of the loss function during minimization.

convergence_norms
: Optional: if save_convergence_norms = TRUE, the norms of the differences between consecutive iterates during minimization.

See Also

convex_clustering, sparse_weights

Examples

```
# Load data
data(two_half_moons)
data = as.matrix(two_half_moons)
X = data[, -3]
y = data[, 3]

# Get sparse weights in dictionary of keys format with k = 5 and phi = 8
W = sparse_weights(X, 5, 8.0)

# Set a sequence for lambda
lambdas = seq(0, 2400, 1)

# Compute clusterpath
res = convex_clusterpath(X, W, lambdas)

# Get cluster labels for two clusters
labels = clusters(res, 2)

# Plot the clusterpath with colors based on the cluster labels
plot(res, col = labels)
```

---

plot.cvxclust *Plot 2D clusterpath*

Description

Plot a clusterpath for two-dimensional data.

Usage

```
## S3 method for class 'cvxclust'
plot(x, col = NULL, labels = NULL, ...)
```

Arguments

x	A cvxclust object.
col	A vector containing cluster membership information. Default is NULL.
labels	A vector containing labels for each object. Default is NULL.
...	Further graphical parameters.

Value

A plot in the console.

Examples

```
# Load data
data(two_half_moons)
data = as.matrix(two_half_moons)
X = data[, -3]
y = data[, 3]

# Get sparse distances in dictionary of keys format with k = 5 and phi = 8
W = sparse_weights(X, 5, 8.0)

# Set a sequence for lambda
lambdas = seq(0, 2400, 1)

# Compute results CMM
res = convex_clusterpath(X, W, lambdas)
plot(res, y + 1)
```

`sparse_weights` *Computation of sparse weight matrix*

Description

Construct a sparse weight matrix in a dictionary-of-keys format. Each nonzero weight is computed as $exp(-phi * ||x_i - x_j||^2)$, where the squared Euclidean distance may be scaled by the average squared Euclidean distance, depending on the argument `scale`. Sparsity is achieved by only setting weights to nonzero values that correspond to two objects that are among each other's k nearest neighbors.

Usage

```
sparse_weights(
  X,
  k,
  phi,
  connected = TRUE,
  scale = TRUE,
  connection_type = "SC"
)
```

Arguments

`X`	An n x p numeric matrix. This function assumes that each row represents an object with p attributes.
`k`	The number of nearest neighbors to be used for non-zero weights.
`phi`	Tuning parameter of the Gaussian weights. Input should be a nonnegative value.

connected — If TRUE, guarantee a connected structure of the weight matrix. This ensures that groups of observations that would not be connected through weights that are based only on the k nearest neighbors are (indirectly) connected anyway. The method is determined by the argument connection_type. Default is TRUE.

scale — If TRUE, scale each squared l2-norm by the mean squared l2-norm to ensure scale invariance of the weights. Default is TRUE.

connection_type — Determines the method to ensure a connected weight matrix if connected is TRUE. Should be one of c("SC", "MST"). SC stands for the method using a symmetric circulant matrix, connecting objects i with objects $i + 1$ (and n with 1). MST stands for minimum spanning tree. The graph that results from the nonzero weights determined by the k nearest neighbors is divided into c subgraphs and a minimum spanning tree algorithm is used to add $c - 1$ nonzero weights to ensure that all objects are indirectly connected. Default is "SC".

Value

A sparseweights object containing the nonzero weights in dictionary-of-keys format.

Examples

```
# Load data
data(two_half_moons)
data = as.matrix(two_half_moons)
X = data[, -3]
y = data[, 3]

# Get sparse distances in dictionary of keys format with k = 5 and phi = 8
W = sparse_weights(X, 5, 8.0)
```

---

two_half_moons *Two interlocking half moons data set*

Description

A dataset containing 150 observations generated according to the two interlocking half moons data generating process. The first two columns contain the x and y-coordinates and the third column contains the cluster ID. Each moon contains 75 observations.

Usage

```
data(two_half_moons)
```

Format

An object of class data.frame with 150 rows and 3 columns.

Index

∗ **datasets**
two_half_moons, 11

as.hclust.cvxclust, 2

clusters, 3
convex_clustering, *2*, 4, *8*
convex_clusterpath, *2*, *6*, 7

hclust, *2*

plot.cvxclust, 9

sparse_weights, *4*, *6–8*, 10

two_half_moons, 11