Package 'skmeans'

October 11, 2026

Version 0.2-21

Title Spherical k-Means Clustering

Description Algorithms to compute spherical k-means partitions. Features several methods, including a genetic and a fixed-point algorithm and an interface to the CLUTO vcluster program.

Imports slam (>= 0.1-31), clue (>= 0.3-39), cluster, stats, utils

Enhances Matrix

License GPL-2

NeedsCompilation no

Author Kurt Hornik [aut, cre] (ORCID: <https://orcid.org/0000-0003-4198-9911>), Ingo Feinerer [aut] (ORCID: <https://orcid.org/0000-0001-7656-8338>), Martin Kober [aut]

Maintainer Kurt Hornik <Kurt.Hornik@R-project.org>

Repository CRAN

Date/Publication 2026-10-11 08:27:12 UTC

Contents

skmeans 1
skmeans_xdist 6

Index 7

`skmeans` *Compute Spherical k-Means Partitions*

Description

Partition given vectors x_b by minimizing the spherical k-means criterion $\sum_{b,j} w_b u_{bj}^m d(x_b, p_j)$ over memberships and prototypes, where the w_b are case weights, u_{bj} is the membership of x_b to class j, p_j is the *prototype* of class j (thus minimizing $\sum_b w_b u_{bj}^m d(x_b, p)$ over p), and d is the cosine dissimilarity $d(x, p) = 1 - \cos(x, p)$.

Usage

```
skmeans(x, k, method = NULL, m = 1, weights = 1, control = list())
```

Arguments

x	A numeric data matrix, with rows corresponding to the objects to be partitioned (such that row b contains x_b). Can be a dense matrix, a simple triplet matrix (package **slam**), or a dgTMatrix (package **Matrix**). Zero rows are not allowed.
k	an integer giving the number of classes to be used in the partition.
method	a character string specifying one of the built-in methods for computing spherical k-means partitions, or a function to be taken as a user-defined method, or NULL (default value). If a character string, its lower-cased version is matched against the lower-cased names of the available built-in methods using pmatch. See **Details** for available built-in methods and defaults.
m	a number not less than 1 controlling the softness of the partition (as the "fuzzification parameter" of the fuzzy c-means algorithm). The default value of 1 corresponds to hard partitions; values greater than one give partitions of increasing softness obtained from a generalized soft spherical k-means problem.
weights	a numeric vector of non-negative case weights. Recycled to the number of objects given by x if necessary.
control	a list of control parameters. See **Details**.

Details

The "standard" spherical k-means problem where all case weights are one and $m = 1$ is equivalent to maximizing the criterion $\sum_j \sum_{b \in C_j} \cos(x_b, p_j)$, where C_j is the j-th class of the partition. This is the formulation used in Dhillon and Modha (2001) and related references, and when optimized over the prototypes yields the criterion function $\mathcal{I}_2$ in the CLUTO documentation.

Obtaining optimal spherical k-means partitions obviously is a computationally hard problem, and several methods are available which attempt to obtain optimal partitions. The built-in methods are as follows.

"genetic" a genetic algorithm patterned after the genetic k-means algorithm of Krishna and Narasimha Murty (1999).

"pclust" a Lloyd-Forgy style fixed-point algorithm which iterates between determining optimal memberships for fixed prototypes, and computing optimal prototypes for fixed memberships. For hard partitions, this can optionally attempt further local improvements via Kernighan-Lin chains of first variation single object moves as suggested by Dhillon, Guan, and Kogan (2002).

"CLUTO" an interface to the vcluster partitional clustering program from CLUTO, the CLUstering TOolkit by George Karypis (Karypis 2002).

"gmeans" an interface to the gmeans partitional clustering program by Yuqiang Guan.

Method "pclust" is the only method available for soft spherical k-means problems. Method "genetic" can handle case weights. By default, the genetic algorithm is used for obtaining hard partitions, and the fixed-point algorithm otherwise.

Common control parameters for methods "genetic" and "pclust" are as follows.

`start` a specification of the starting values to be employed. Can either be a character vector with elements `"p"` (randomly pick objects as prototypes), `"i"` (randomly pick ids for the objects), `"S"` (take p minimizing $\sum_b w_b d(x_b, p)$ as the first prototype, and successively pick objects farthest away from the already picked prototypes), or `"s"` (like `"S"`, but with the first prototype a randomly picked object). Can also be a list of `skmeans` objects (obtained by previous runs), a list of prototype matrices, or a list of class ids. For the genetic algorithm, the given starting values are used as the initial population; the fixed-point algorithm is applied individually to each starting value, and the best solution found is returned. Defaults to randomly picking objects as prototypes.

`reltol` The minimum relative improvement per iteration. If improvement is less, the algorithm will stop under the assumption that no further significant improvement can be made. Defaults to `sqrt(.Machine$double.eps)`.

`verbose` a logical indicating whether to provide some output on minimization progress. Defaults to `getOption("verbose")`.

Additional control parameters for method `"genetic"` are as follows.

`maxiter` an integer giving the maximum number of iterations for the genetic algorithm. Defaults to 12.

`popsize` an integer giving the population size for the genetic algorithm. Default: 6. Only used if `start` is not given.

`mutations` a number between 0 and 1 giving the probability of mutation per iteration. Defaults to 0.1.

Additional control parameters for method `"pclust"` are as follows.

`maxiter` an integer giving the maximal number of fixed-point iterations to be performed. Default: 100.

`nruns` an integer giving the number of fixed-point runs to be performed. Default: 1. Only used if `start` is not given.

`maxchains` an integer giving the maximal length of the Kernighan-Lin chains. Default: 0 (no first variation improvements are attempted).

Control parameters for method `"CLUTO"` are as follows.

`vcluster` the path to the CLUTO `vcluster` executable.

`colmodel` a specification of the CLUTO column model. See the CLUTO documentation for more details.

`verbose` as for the genetic algorithm.

`control` a character string specifying arguments passed over to the `vcluster` executable.

Control parameters for method `"gmeans"` are as follows.

`gmeans` the path to the `gmeans` executable.

`verbose` as for the genetic algorithm.

`control` a character string specifying arguments passed over to the `gmeans` executable.

Method "CLUTO" requires that the CLUTO vcluster executable is available. CLUTO binaries for the Linux, SunOS, Mac OS X, and MS Windows platforms used to be downloadable from 'https://www-users.cse.umn.edu/~karypis/cluto/'. If the executable cannot be found in the system path via Sys.which("vcluster") (i.e., named differently or not made available in the system path), its (full) path must be specified in control option vcluster.

Method "gmeans" requires that the gmeans executable is available. Sources for compilation with ANSI C++ compliant compilers are available from https://github.com/feinerer/gmeans-ansi-compliant; original sources can be obtained from https://www.cs.utexas.edu/~dml/Software/gmeans.html. If the executable cannot be found in the system path via Sys.which("gmeans") (i.e., named differently or not made available in the system path), its (full) path must be specified in control option gmeans.

User-defined methods must have formals x, k and control, and optionally may have formals weights or m if providing support for case weights or soft spherical k-means partitions, respectively.

Value

An object inheriting from classes skmeans and pclust (see the information on pclust objects in package **clue** for further details) representing the obtained spherical k-means partition, which is a list with components including the following:

prototypes	a dense matrix with k rows giving the prototypes.
membership	cluster membership as a matrix with k columns (only provided if $m > 1$).
cluster	the class ids of the closest hard partition (the partition itself if $m = 1$).
value	the value of the criterion.

Objects representing spherical k-means partitions have special methods for print, cl_validity (providing the "dissimilarity accounted for") from package **clue**, and silhouette from package **cluster** (the latter two take advantage of the special structure of the cosine distance to avoid computing full object-by-object distance matrices, and hence also perform well for large data sets).

Package **clue** provides additional methods for objects inheriting from class pclust, see the examples.

Author(s)

Kurt Hornik <Kurt.Hornik@R-project.org>,
Ingo Feinerer <feinerer@logic.at>,
Martin Kober.

References

Dhillon IS, Guan Y, Kogan J (2002). "Iterative Clustering of High Dimensional Text Data Augmented by Local Search." In *2002 IEEE International Conference on Data Mining*, 131–138. doi:10.1109/ICDM.2002.1183895. https://www.cs.utexas.edu/~inderjit/public_papers/iterative_icdm02.pdf.

Dhillon IS, Modha DS (2001). "Concept Decompositions for Large Sparse Text Data Using Clustering." *Machine Learning*, **42**(1–2), 143–175. doi:10.1023/a:1007612920971.

Karypis G (2002). "CLUTO: A Clustering Toolkit." Technical Report 02-017, Department of Computer Science, University of Minnesota. https://hdl.handle.net/11299/215521.

Krishna K, Narasimha Murty M (1999). "Genetic K-means algorithm." *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, **29**(3), 433–439. doi:10.1109/3477.764879.

Examples

```
set.seed(1234)

## Use CLUTO dataset 're0' and the reader for CLUTO sparse matrix
## format in package 'slam'.  (In text clustering applications, x will
## often be a DocumentTermMatrix object obtained from package 'tm'.)
x <- slam::read_stm_CLUTO(system.file("cluto", "re0.mat",
                                      package = "skmeans"))
## Which is not really small:
dim(x)

## Hard partition into 5 clusters.
hparty <- skmeans(x, 5, control = list(verbose = TRUE))
## Criterion value obtained:
hparty$value
## Compare with "true" classifications:
class_ids <- attr(x, "rclass")
table(class_ids, hparty$cluster)
## (Note that there are actually 10 "true" classes.)

## Plot the silhouette information for the obtained partition.
require("cluster")
plot(silhouette(hparty))
## Clearly, cluster 3 is "best", and cluster 5 needs splitting.

## Soft partition into 5 clusters.
sparty <- skmeans(x, 5, m = 1.1,
                  control = list(nruns = 5, verbose = TRUE))
## Criterion value obtained:
sparty$value
## (This should be a lower bound for the criterion value of the hard
## partition.)

## Compare the soft and hard partitions:
table(hparty$cluster, sparty$cluster)
## Or equivalently using the high-level accessors from package 'clue':
require("clue")
table(cl_class_ids(hparty), cl_class_ids(sparty))
## Which can also be used for computing agreement/dissimilarity measures
## between the obtained partitions.
cl_agreement(hparty, sparty, "Rand")

## How fuzzy is the obtained soft partition?
cl_fuzziness(sparty)
## And in fact, looking at the membership margins we see that the
## "sureness" of classification is rather high:
```

```
summary(cl_margin(sparty))
```

---

skmeans_xdist *Cosine Cross-Distances*

Description

Compute cosine cross-distances between the rows of matrices.

Usage

```
skmeans_xdist(x, y = NULL)
```

Arguments

x	A numeric data matrix. Can be a dense matrix, simple triplet matrix (package **slam**), or a dgTMatrix (package **Matrix**).
y	NULL (default), or as for x. The default is equivalent to taking y as x (but more efficient).

Value

A dense matrix d with entry $d_{ij} = 1 - \cos(x_i, y_j)$ the cosine distance between the i-th row x_i of x and the j-th row y_j of y.

Index

∗ **cluster**
skmeans, 1

cl_validity, *4*

dgTMatrix, *2*, *6*

pclust objects, *4*
pmatch, *2*
print, *4*

silhouette, *4*
simple triplet matrix, *2*, *6*
skmeans, 1
skmeans_xdist, 6
Sys.which, *4*