# User Guide for CLARA.seq

Louis Tochon Matthias Studer

06/27/2021

## Contents

# Introduction

CLARA.seq is a R-package to cluster big datasets of sequences. It uses the TraMineR (http://cran.r-project.org/web/packages/TraMineR/) package to calculate distances between each sequence but this package is made for 32-bits computers, which implies that the quantity of objects is limited to 46'300 objects (because we need to compute a matrix of size $n * n$). CLARA.seq provides three different functions : CLARA, CLARANS and ROBUST FUZZY C-MEDOIDS.
The package has an index called Davies-Bouldin index to check the validity of the clustering found after the use of a clustering method.

# Installation

This package is still in a test phase. Therefore it is only available on GitHub : Link to repository.

To download the package and get the functions in your personal environment, here are the commands :

```
install.packages("devtools")
library(devtools)
install_github("Ltochon/CLARA.seq")
library(CLARA.seq)
```

# Use cases

## Creation of sequences

To begin, the creation of sequences is needed. The use of the *TraMineR* package allows to get example datas :

```
#creating sequences
library(TraMineR)
data(mvad)
mvad.labels <- c("employment", "further education", "higher education","joblessness",
+   "school", "training")
mvad.scode <- c("EM", "FE", "HE", "JL", "SC", "TR")
mvad.seq <- seqdef(mvad, 17:86, states = mvad.scode,abels = mvad.labels, xtstep = 6)
```

## CLARA

CLARA has been implemented with a parameter that can determine which aspect of the clustering is the most important. Indeed the parameter *find_best_method* allows users to choose between :

- **Distance** : finding the best clustering with the mean distance method which aims to minimize the distance between each object and his medoid
- **DB** : finding the best clustering with the Davies-Bouldin index method which aims to minimize the index of the global clustering

### Partitioning sequences with the mean distance method

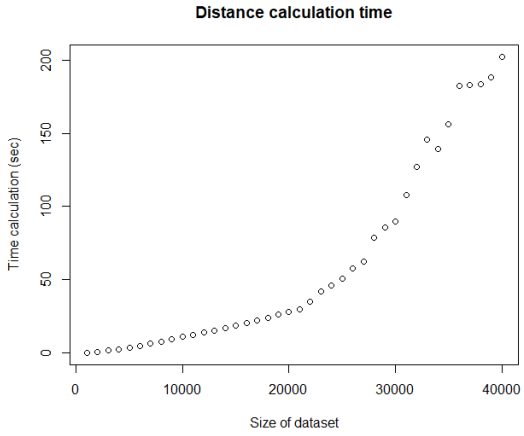
With the sequences previously created, a clustering can be applied on them. The process will be parallelized to find the best clustering faster. The number of cores used is initially set to use all possible cores minus one to avoid any overloading.

The function call is as follow for CLARA with the mean distance :

```
my_cluster <- clara_clust(mvad.seq,nb_cluster = 4, nb_sample = 20, size_sample = 20, with.diss = TRUE)
```

The most important parameters for this algorithm are the number of clusters (*nb_cluster*), the number of subsets to use (*nb_sample*) and the size of each of these subsets (*size_sample*).

**Warnings** the calculation time will increase exponentially according to the size of each subset, as shown on this figure :




The calculation time has been analyzed on the supercomputer of the University of Geneva, *Baobab*, using an Intel® Xeon® E5-2660 processor.

When all iterations have been done, a console output will be written with a summary of the mean distance value of each iteration :

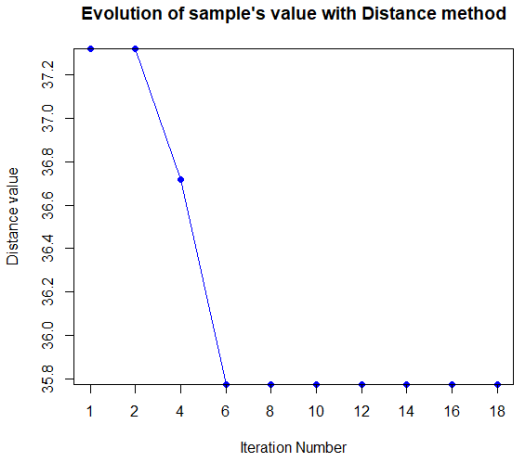
```
CLARA ALGORITHM Improved


Table of Sample Values with Distance Method

N°1  (Med: 609, 186, 167, 309) : 37.32022471911  |  N°2  (Med: 684, 143, 40, 200)  : 40.20224719101
N°3  (Med: 639, 54, 479, 63)   : 36.82303378652  |  N°4  (Med: 259, 212, 288, 177) : 36.71629213415
N°5  (Med: 599, 691, 262, 27)  : 35.77528088764  |  N°6  (Med: 208, 509, 251, 219) : 41.72471911236
N°7  (Med: 134, 686, 21, 465)  : 40.33988404494  |  N°8  (Med: 323, 350, 2, 125)   : 40.08707516854
N°9  (Med: 48, 640, 131, 56)   : 42.2528988764   |  N°10 (Med: 525, 473, 125, 390) : 39.15443820225
N°11 (Med: 314, 464, 50, 366)  : 38.69213483146  |  N°12 (Med: 269, 628, 273, 200) : 39.68314606742
N°13 (Med: 592, 435, 560, 470) : 37.19629213483  |  N°14 (Med: 280, 506, 355, 246) : 39.38483106742
N°15 (Med: 299, 125, 604, 618) : 37.1404494382   |  N°16 (Med: 579, 186, 320, 514) : 47.43252696629
N°17 (Med: 282, 321, 354, 679) : 39.23370786517  |  N°18 (Med: 88, 247, 166, 376)  : 40.45561797753
N°19 (Med: 126, 376, 307, 19)  : 44.57078651685  |  N°20 (Med: 252, 582, 25, 549)  : 38.02247191011

[>] Minimum Index Value for Sample N°5

Calculation time : 14.4400000000001 sec.
```

The function will create an object of class *clara_seq* with the following values :

```
$ seq #the original sequences dataset
$ id.med #the identifier of the medoids
$ clusters #the resulting clustering
$ diss #the matrix of distance of size k*n
$ evol.diss #the value of the best value found for each iteration
$ find_best_method #method used to find the best clustering
```

The plot parameter determines if the evolution of the mean distance value must be shown at the end.

The graph is the representation of the value evolution for each iteration (depending on the method used) :




In case the user wants to get this plot again, the default function *plot()* can render it again, thanks to the function *plot.clara-seq()* which detects if the object passed as parameter is from the *clara-seq*'s class.

```
plot(my_cluster)
```

### Partitioning sequences with the Davies-Bouldin index method

The call of this function is similar to the one previously seen. The only difference is that the *find_best_method* must be specified as *DB*. In this example, we are using the same sequences created earlier

```
my_cluster <- clara_clust(mvad.seq,nb_cluster = 4, nb_sample = 20,
+   size_sample = 20, with.diss = TRUE, find_best_method = "DB")
```

This function will create the same type of console output as with the mean distance method :

```
CLARA ALGORITHM Improved


Table of Sample Values with DB Method

N°1  (Med: 182, 71, 98, 118)  : 1.342827221118   |  N°2  (Med: 25, 126, 200, 153) : 1.299852116316
N°3  (Med: 77, 174, 189, 179) : 1.40897064430    |  N°4  (Med: 3, 96, 36, 69)     : 1.248435482041
N°5  (Med: 68, 171, 111, 170) : 1.0031981757334  |  N°6  (Med: 142, 127, 119, 66) : 0.9345574879117
N°7  (Med: 171, 116, 118, 33) : 0.834863495085   |  N°8  (Med: 127, 90, 101, 31)  : 1.479710535507
N°9  (Med: 74, 189, 92, 32)   : 1.168584125339   |  N°10 (Med: 26, 146, 89, 118)  : 1.50465609725
N°11 (Med: 22, 157, 65, 142)  : 1.445766706488   |  N°12 (Med: 119, 33, 149, 28)  : 1.264641148222
N°13 (Med: 126, 91, 179, 74)  : 0.9903531541278  |  N°14 (Med: 146, 66, 94, 111)  : 1.31802845133
N°15 (Med: 1, 22, 111, 119)   : 1.970945630127   |  N°16 (Med: 155, 139, 65, 201) : 1.655903988049
N°17 (Med: 63, 20, 93, 182)   : 1.546096302749   |  N°18 (Med: 137, 119, 33, 92)  : 1.388637698447
N°19 (Med: 179, 115, 95, 3)   : 1.508608204084   |  N°20 (Med: 173, 66, 95, 142)  : 1.186216041016

[>] Minimum Index Value for Sample N°7

Calculation time : 11.9299999999998 sec.
```

The same object of the same class as with the previous method will be created, and the same type of plot can be rendered at any time with the same function *plot()*.

## CLARANS

CLARANS function has been implemented but is less efficient than CLARA. Indeed, CLARANS cannot be parallelized and the calculation time is bigger than CLARA's. After many tests, it has been shown that for the same amount of time, we get better results for CLARA.

### Partitioning sequences

The CLARANS algorithm can be called with the following function :

```
my_cluster2 <- clarans_clust(mvad.seq,nb_cluster = 4, maxneighbours = 20, numlocal = 4,
+   plot = TRUE)
```

The main parameters are the number of clusters (*nb_cluster*), the number of neighbours to explore (*maxneighbours*) and the number of tries to do (*numlocal*) i.e. the number initialization of the initial medoid's set.

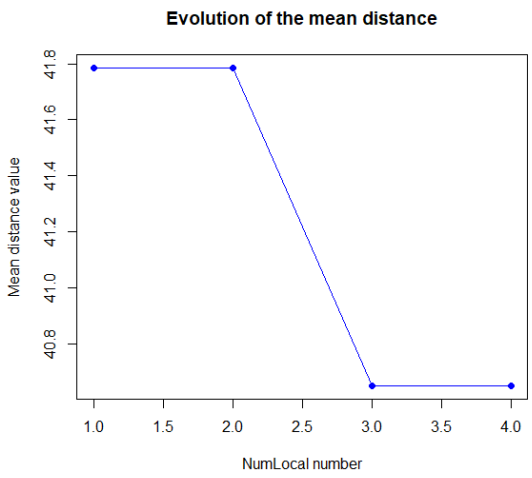
A console output will be written with a summary for each *numlocal* value :

```
CLARANS ALGORITHM

Table of Iteration Distance

N°1 (Med: 163, 176, 232, 282) : 41.78651685393   N°2 (Med: 570, 306, 662, 435) : 51.21348314606
N°3 (Med: 211, 269, 479, 299) : 40.64887640449   N°4 (Med: 7, 469, 89, 562)    : 43.56741573033

[>] Minimum Distance for iteration N°3

Calculation time : 16.6700000000001 sec.
```

The function will create an object of class *clarans_seq* with the following values :

```
$ seq #the original sequences dataset
$ id.med #the identifier of the medoids
$ clusters #the resulting clustering
$ diss #the matrix of distance of size k*n
$ evol.diss #the value of the best value found for each numlocal value
```

The plot parameter determines if the quality plot must be render. If true, the following graph is drawn :




Again, this quality graph can be created with a *clarans_seq* object with the function :

```
plot(my_cluster2)
```

## ROBUST FUZZY C-MEDOIDS

This function comes from the fuzzy algorithm family and provide a fuzzy clustering based on samples from the big dataset. This variant applies a fuzzy clustering on each subset and takes the most popular objects as medoids (determined with the membership matrix). For this algorithm, the objective function to minimize is :

$$\sum_{x_j \in S} u_{i,j}^m d(med, x_j)$$

where $x\_j$ is an object in the dataset without outliers $S$, $i$ is the cluster for which a medoid is being sought, and $med$ is the medoid tested.

**Warning** this function has not been fully tested.

### Partitioning sequences

This function can be used as follow :

```
my_cluster3 <- fuzzy_clust(mvad.seq,nb_sample = 14, size_sample = 50, plot = TRUE,
+   threshold = 7, max_iter = 10, p=5)
```

The main parameters are the number of clusters (*nb_cluster*), the number of subsets (*nb_sample*) and the size of each one of them (*size_sample*) as in the *clara_clust()* function, the threshold to exclude outliers (*threshold*), the maximal number of iterations to find the best set of medoids (*max_iter*) and the number of candidates to test to find a better medoid ($p$).

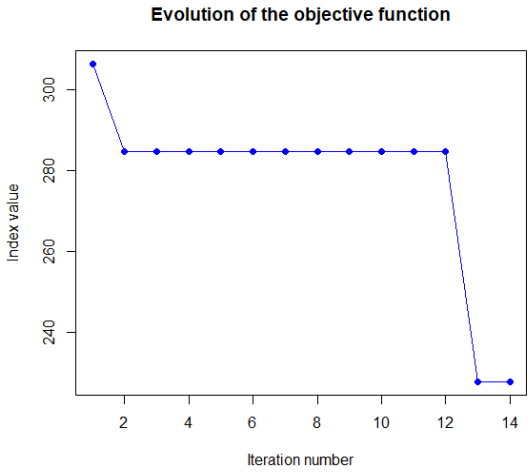
A console output will be written with a summary of each iterations index value (the value of the function to minimize):

```
ROBUST FUZZY C-MEDOIDS ALGORITHM

Table of objective function values

N°1  (Med: 166, 171, 1, 672)   : 306.3782076078  |  N°2  (Med: 478, 54, 446, 489)  : 284.7549317274
N°3  (Med: 345, 430, 140, 355) : 316.5071933888  |  N°4  (Med: 228, 184, 685, 504) : 351.6212640614
N°5  (Med: 48, 505, 455, 588)  : 367.0773016113  |  N°6  (Med: 481, 425, 338, 638) : 286.4788921731
N°7  (Med: 412, 120, 77, 639)  : 298.6053701533  |  N°8  (Med: 502, 158, 190, 51)  : 389.8523768593
N°9  (Med: 192, 646, 666, 363) : 387.4922466893  |  N°10 (Med: 253, 467, 214, 223) : 319.6099482256
N°11 (Med: 173, 554, 585, 495) : 334.3587832990  |  N°12 (Med: 453, 2, 144, 109)   : 331.5818583005
N°13 (Med: 497, 543, 602, 605) : 227.8510767599  |  N°14 (Med: 425, 350, 599, 396) : 239.3543957804

[>] Minimum Index for Sample N°13

Calculation time : 14.27 sec.
```

The function will create an object of class *clarafuzzy_seq* with the following values :

```
$ seq #the original sequences dataset
$ id.med #the identifier of the medoids
$ clusters #the resulting clustering
$ diss #distance matrix
$ harm_value #harm value to adjust the threshold
$ nb_iter #number of iterations needed to find the medoids
$ membership #membership matrix
$ q #value of the index for each iteration
$ evol.diss #the value of the best value found for each iteration
```

The plot parameter determines if the quality plot must be rendered. If true, the following graph is drawn :




Again, this quality graph can be created with a *clarafuzzy_seq* object with the function :

```
plot(my_cluster3)
```

## Quality analysis

A quality analysis can be execute with the help of the Davies-Bouldin function or the *TraMineR* package.
For the following examples, the object *my_cluster* coming from the previous CLARA clustering is used but the quality analysis is working with every object from each function.

### Davies-Bouldin Index

Davies-Bouldin provides a quality analysis index working with big datasets, because it doesn't require to compute the complete distance matrix. Indeed, this index aims to calculate the following equation :

$$\frac{1}{K}\sum_{k\in K}\max_{k'\neq k}\frac{diam(k)+diam(k')}{dist(k,k')}$$

where K is the set of cluster, k and k' are two clusters in K, diam(k) is the diameter of the cluster k and dist(k,k') is the distance between the medoids of k and k'.

This index calculates the ratio between the intra-cluster and the inter-cluster dispersion of a clustering.

The calculation of this index on an object coming from one of the three algorithms of this package can easily be done :

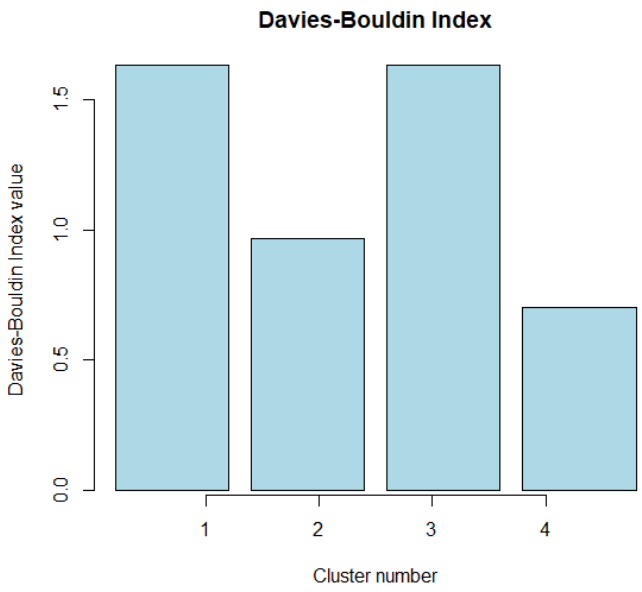
```
db <- davies_bouldin(my_cluster)
```

The call of this function will produce a console output as follows :

```
DAVIES-BOULDIN INDEX for clara_seq Clustering

Value of DB Index for a 4-clusters : 1.23197050017674
Calcul time : 0.0100000000002183 sec.
```

This index is very fast to compute, even with a big dataset.

A plot can also be rendered, illustrating the value of the index for each cluster. The resulting global index is the mean of these different values :




Finally, the function *davies_bouldin* has as output the value of the global index.

### Visual representation of the clustering

With the package *TraMineR* it is possible to draw a graph with the sequences distribution. To do this, the function *seqdplot* will be use as follows :

```
seqdplot(mvad.seq, my_cluster$clusters)
```

A graph as the following can be obtained, showing the distribution and the frequency of the sequences :