

# Identifying and qualifying deviant cases in clusters of sequences: The why and the how

R. Piccarreta

January 2024

## Contents

<b>1</b>	<b>Sequences, dissimilarities and clusters</b>	<b>2</b>
<b>2</b>	<b>Measures of deviation of sequences from their own cluster</b>	<b>3</b>
<b>3</b>	<b>Obtain relative deviations and flag noisy cases</b>	<b>4</b>
<b>4</b>	<b>Organize information on flagged deviant cases</b>	<b>5</b>
<b>5</b>	<b>Create flagged plots</b>	<b>6</b>

This document describes the syntax to obtain measures and figures in the paper: Piccarreta, R., Struffolino, E. *Identifying and Qualifying Deviant Cases in Clusters of Sequences: The Why and The How*. Eur J Population 40, 1 (2024). <https://doi.org/10.1007/s10680-023-09682-3>

The data used in the article cannot be distributed. To have access, researchers can apply to the Italian Statistical Office (ISTAT, [www.istat.it](http://www.istat.it)). For illustrating the procedure in R, we use data obtained from the dataset from McVicar and Anyadike-Danes (2002) which is included in the package **TraMineR**. The data describe the school to work transitions from school to work observed on 712 individuals. For the sake of illustration the data will be modified by adding some deviant sequences. The modified dataframe **MVAD.noise** is available in this folder and should be loaded (see the syntax below).

The syntax below describes the steps to follow to produce the results shown in our paper, as well as a description of the functions we created at this aim - all included in the script **Functions\_Deviant\_Github.R** available in this folder, which has to be sourced. The packages **TraMineR**, **WeightedCluster**, and **seriation** should be installed.

```
library(TraMineR)
library(WeightedCluster)
library("seriation")

# the following files should have been saved on hard-disk
# possibly their path should be specified if they are not in the working directory

# Source functions
source("Functions_Deviant_Github.R")

# import the data
load("MVAD.noise.Rdata")
```

## 1 Sequences, dissimilarities and clusters

Information on sequences is stored in columns 15-86:

```
seq.orig<-MVAD.noise[,15:86]

# have a look at the data (first 10 time periods)
seq.orig[1,1:10]
>      Jul.93   Aug.93   Sep.93   Oct.93   Nov.93   Dec.93   Jan.94
> 1 training training employment employment employment employment training
>      Feb.94   Mar.94   Apr.94
> 1 training employment employment
```

To build the dissimilarity matrix, we resort to the functions in package **TraMineR** (please refer to the manual for further details); we build the sequences in the required format and obtain the dissimilarities using OM (of course other options are available).

```
seq.Tr <- seqdef(seq.orig)
# define costs
seq.costs<-seqsubm(seq.Tr, method="TRATE")
# build dissimilarities
seq.dist <- as.matrix(as.dist(seqdist(seq.Tr,method="OM",indel=1,
                                     sm=seq.costs)))
```

To extract clusters we use the PAM algorithm as implemented in the package **WeightedCluster**. We select 6 clusters (based on standard criteria - syntax not shown, please refer to the **WeightedCluster** manual for details).

```
set.seed(200)
n.cl<-6
PAM.stats<- wckMedoids(seq.dist, k=n.cl)
# create a numeric id for clusters
seq.clusters<-factor(PAM.stats$clustering,
                     levels=unique(PAM.stats$clustering),
                     labels=1:n.cl)
```

## 2 Measures of deviation of sequences from their own cluster

The function `obtain.deviations(Partition, Distance, NN, ref.percentiles)` calculates three measures of deviations of sequences from the cluster they have been assigned to, namely a) the average dissimilarity of cases from cases in their cluster (Avedist.own); b) the dissimilarity of cases from their cluster's medoid (Dist.FromMedoid); c) the maximal dissimilarity between one case and its k-th nearest neighbor (DistfromNN.k, for each value of k specified in the argument NN).

The function also calculates the average, the standard deviation and the percentiles (of order specified in `ref.percentiles`) of the three measures above both based on the entire set of data (General) and by clusters (ByCl).

The function has the following arguments:

- **Partition**: a vector or a factor containing cluster membership id;
- **Distance**: a distance matrix (can be a matrix or a 'dist' object);
- **NN**: vector with the number of nearest neighbors to consider when calculating the maximal distance between each case and cases in its neighborhood (more values can be specified);
- **ref.percentiles**: required percentiles of the deviation measures.

For example, below we obtain the deviations based on the distance matrix calculated above:

```
Criteria_References<-obtain.deviations(Partition=seq.clusters,
                                       Distance=seq.dist, NN=c(5,10,15),
                                       ref.percentiles=c(0.9,0.95,0.99))
```

The function returns a **list** with the following elements:

- **Deviations**: a dataframe with the deviations mentioned above calculated for each case (Avedist.own, Dist.FromMedoid, DistfromNN.k, for k in NN);

```
head(Criteria_References$Deviations)
>   Cluster Avedist.own Dist.FromMedoid DistfromNN.5 DistfromNN.10 DistfromNN.15
> 1       1    22.29314      7.878197      7.837596      7.837596      7.837596
> 2       2    26.70515     21.929738      9.818715     21.914124     21.992193
> 3       3    69.13676     51.899759     37.909265     51.426503     51.756172
> 4       4    66.25960     64.574492     21.918798     27.806522     39.102305
> 5       2    22.95026      3.926959      2.000000      3.963479      5.965638
> 6       4    35.20309     21.675192      5.913969      9.832767     17.838698
```

- **Reference.Values**: a dataframe with the averages (Summary='Average'), the standard deviations (Summary='Std.Dev') and the percentiles (Pct100p, with p in `ref.percentiles`) for each column in **Deviations** calculated both based on the entire sample (Type='General') and by cluster (Type='ByCl');

```
head(Criteria_References$Reference.Values)
>   Avedist.own Dist.FromMedoid DistfromNN.5 DistfromNN.10
> Gen_Average    42.39387      29.73071     16.578908     20.93233
> Gen_Std.Dev     16.82385      21.81958     15.364233     17.37982
> Gen_Pct90       66.56500      62.82528     39.707214     47.17095
> Gen_Pct95       75.20156      70.79141     45.931011     51.78395
```

```

> Gen_Pct99      85.51069      85.00461      61.352542      66.46971
> ByCl_1_Average 29.70338      20.90706      8.833423      11.26244
>
>      DistfromNN.15      Type Summary Cluster
> Gen_Average      23.98653 General Average      <NA>
> Gen_Std.Dev      18.45888 General Std.Dev      <NA>
> Gen_Pct90      49.45300 General Pct90      <NA>
> Gen_Pct95      55.94812 General Pct95      <NA>
> Gen_Pct99      70.56609 General Pct99      <NA>
> ByCl_1_Average 13.61954 ByCl Average      1

```

- **createdby**: a character indicating the name of the function, (`obtain.deviations`).

### 3 Obtain relative deviations and flag noisy cases

The function `obtain.rel_deviations(info.deviations, criterion, ref.location, ref.scale, ref.noise, Distance)` calculates the relative deviation of sequences from their clusters or identify noisy data based on the nearest neighbors criterion.

The function has the following arguments:

- **info.deviations**: output of function `obtain.deviations`.
- **criterion**: one of the measures of deviation. It can take values `Avedist.own`, `Dist.FromMedoid`, `DistfromNN.k`, for `k` in `NN`;
- **ref.location**: a single-character *named* vector indicating whether the considered criterion should be centered based on a reference value. The vector's name can be 'General' or 'ByCl', and indicates whether the whole-sample or the by-clusters center should be used to center. For example: `ref.location(General='Average')` centers the specified criterion using the general mean of the criterion itself (more examples below). The vector value can be the 'Average' or a percentile, 'Pct100p' where `p` is one of the values specified in the argument `ref.percentiles` of the function `obtain.deviations`. If `ref.location` is not specified, the selected criterion is not scaled.
- **ref.scale**: a single-character *named* vector indicating whether the considered criterion should be scaled based on a reference value. The vector's name can be 'General' or 'ByCl', and indicates whether the whole-sample or the by-clusters scaling factor should be considered. For example, `ref.scale(General='Average')` calculates the ratio between the specified criterion and the general average. The vector value can be the 'Average', 'Std.Dev' or a percentile, 'Pct100p', where `p` is one of the values specified in the argument `ref.percentiles` of the function `obtain.deviations`. If `ref.scale` is not specified, the selected criterion is not scaled.
- **ref.noise**: a single-character vector, *alternative* to `ref.location` and/or `ref.scale`. It specifies the order of the percentile used to flag cases as core, border, or noise. It can be only a percentile, 'Pct100p', where `p` is one of the values specified in the argument `ref.percentiles` of the function `obtain.deviations`, and only the 'General' percentile is automatically used (if the name of the vector is 'ByCl', the request is ignored). If noisy cases have to be flagged, a distance matrix should be indicated.
- **Distance**: distance matrix - same used in `obtain.deviations` - which *must* be specified together with `ref.noise`.

The function returns a vector with the selected criterion, centered and/or scaled criterion as specified via `ref.location` and/or `ref.scale` **\*OR\*** with the indication of the type of case (core, border, or noise) obtained by specifying `ref.noise`.

Some examples are reported below.

**Scale deviations from cluster and from medoid using the general mean (approach followed in the original article):**

```

ave.gen_mean<-obtain.rel_deviations(Criteria_References,criterion="Avedist.own",
                                     ref.scale=c(General="Average"))

```

```
med.gen_mean<-obtain.rel_deviations(Criteria_References,criterion="Dist.FromMedoid",
                                   ref.scale=c(General="Average"))
```

Define noisy data based on distances from NN (used in the original article):

```
FlagNoise<-obtain.rel_deviations(Criteria_References,criterion="DistfromNN.5",
                                 ref.noise="Pct95",
                                 Distance=seq.dist)
```

Below are possible alternatives (not used in the original article) to scale/center average from other cases in own cluster of the distance from medoid.

Scale deviations from cluster and from medoid using the by-cluster means:

```
ave.bycl_mean<-obtain.rel_deviations(Criteria_References,criterion="Avedist.own",
                                     ref.scale=c(ByCl="Average"))

med.bycl_mean<-obtain.rel_deviations(Criteria_References,criterion="Dist.FromMedoid",
                                     ref.scale=c(ByCl="Average"))
```

Scale and center deviations from cluster and from medoid using general means and standard deviations:

```
ave.gen_std<-obtain.rel_deviations(Criteria_References,criterion="Avedist.own",
                                   ref.location =c(General="Average"),
                                   ref.scale=c(General="Std.Dev"))

med.gen_std<-obtain.rel_deviations(Criteria_References,criterion="Dist.FromMedoid",
                                   ref.location =c(General="Average"),
                                   ref.scale=c(General="Std.Dev"))

# using by.clusters means and standard dev:

ave.bycl_std<-obtain.rel_deviations(Criteria_References,criterion="Avedist.own",
                                    ref.location =c(ByCl="Average"),
                                    ref.scale=c(ByCl="Std.Dev"))

med.bycl_std<-obtain.rel_deviations(Criteria_References,criterion="Dist.FromMedoid",
                                    ref.location =c(ByCl="Average"),
                                    ref.scale=c(ByCl="Std.Dev"))
```

## 4 Organize information on flagged deviant cases

In the original article we define as deviant:

- cases with a deviation from other cases in own cluster  $> 1.5$  times the General Average deviation;
- cases with a distance from their cluster's medoid  $> 1.5$  times the General Average distance from medoid;
- cases flagged as noise based on the 95-th General Percentile (using five nearest neighbors,  $NN=5$ )

In order to identify cases to display in the sub-plots for deviant cases, it is convenient to store information on cases flagged as deviant in a dataframe:

```
Flag.Regular.Deviant<-data.frame(Deviant.Ave=ave.gen_mean>1.5,
                                 Deviant.Medoid=med.gen_mean>1.5,
                                 Noise=FlagNoise=="Noise")

# for each case count according to how many criteria it is flagged as deviant
Flag.Regular.Deviant$HowManyDeviant<-apply(Flag.Regular.Deviant[,1:3],1,sum)
```

To build the flagged plots, we refer to a list with one element for each plot whose elements indicate whether a case is regular or deviant. For example, in the original article we build for each cluster one plot for cases flagged as regular whatever the criterion, and one sub-plot for each type of deviation.

```
Use.deviant<-list(Regular=Flag.Regular.Deviant$HowManyDeviant==0,
  "a."=Flag.Regular.Deviant$Deviant.Ave==T,
  "b."=Flag.Regular.Deviant$Deviant.Medoid==T,
  "c."=Flag.Regular.Deviant$Noise==T)
```

As an alternative one might also create a plot for regular cases, one sub-plot for cases flagged as deviant by one criterion only, and one additional sub-plot for sequences flagged as deviant based on more criteria (not used in the original article):

```
IdDeviant.List.Sep<-list(Regular=Flag.Regular.Deviant$HowManyDeviant==0,
  "a."=Flag.Regular.Deviant$Deviant.Ave==T &
    Flag.Regular.Deviant$HowManyDeviant==1,
  "b."=Flag.Regular.Deviant$Deviant.Medoid==T &
    Flag.Regular.Deviant$HowManyDeviant==1,
  "c."=Flag.Regular.Deviant$Noise==T &
    Flag.Regular.Deviant$HowManyDeviant==1,
  "d."=Flag.Regular.Deviant$HowManyDeviant>1)
```

## 5 Create flagged plots

To build flagged plots, we prefer to avoid defining functions and rather guiding you through the steps followed to build the plots. This allows having a better control on the final appearance of the plots, that depends also on the number of clusters. Please note that we do not rely on the plots obtained using *TraMineR*. As a preliminary step, it is necessary to recode the sequences using numerical values rather than labels.

```
# recode sequences using a numeric format
seq.num<-seq.orig
for(k in 1:ncol(seq.num)){
  seq.num[,k]<-as.numeric(factor(seq.num[,k],
    levels=c("joblessness","school","FE",
             "training","HE","employment")))
```

We also create a vector with colors and labels for the states, to be used in the legend.

```
Lab.Col_states<- c(JL="red", S="yellow", FE="orange", T="cyan",
  HE="green4",E="darkblue")
```

Below is the information on the number of clusters – needed to set the parameters for the plots - and on the labels to display in the plots' titles, as well as some choices concerning the number of clusters (displayed vertically) in each plot and the relative space left for the main plot (displaying regular cases) for the sub-plots dedicated to deviant sequences, and for the legend.

```
number.cl<-length(levels(seq.clusters))
labels.cl<-levels(seq.clusters) # labels of clusters
labcl="Cluster "

# save current par options and set par options
par.or <- par(no.readonly = TRUE)

# these parameters define the space between and around plots
marplot=c(0.3, 2,0.3,0.5)
omaplot=c(0.1 ,1.2 ,1.6 ,0.1 )
```

```

# Display of legend / nr of columns in the legend
ncol.legend=length(Lab.Col_states)

# Display of plots: how many clusters in each plot?
n.cl.perplot<-4

# relative heights of plots for regular and deviant cases
height.reg<-9
height.deviant<-3
# height of the legend
heightt.legend<-2

```

Below is the syntax to obtain the plots, depending on the choices and on the objects defined above. I typically do not use `window()` but simply expand the plots pane in RStudio in order to properly visualize the plots. The syntax depends on the list `Use.deviant` defined before, on the dissimilarity matrix, `seq.dist`, on the sequences in a numeric coding, `seq.num`, and on the factor identifying cluster membership, `seq.clusters`.

```

## Total nr of plots:
n.todisplay<-trunc(number.cl/n.cl.perplot)
if(number.cl>n.todisplay){n.todisplay<-n.todisplay+1}
tot.plot<-n.todisplay*n.cl.perplot

# blocks of plots
blocks<-as.numeric(cut(1:tot.plot,breaks=n.todisplay))

## layout matrix
nplots<-length(Use.deviant) # list with id of cases in subplots

Matrix.To.Disp<-matrix(1:(nplots*n.cl.perplot),ncol=n.cl.perplot,nrow=nplots)
add<-rep((max(Matrix.To.Disp)+1),ncol(Matrix.To.Disp))
Matrix.To.Disp<-rbind(Matrix.To.Disp,add)

# 1 plot for each block (w)
for(w in 1:n.todisplay){
  layout(Matrix.To.Disp, heights = c(height.reg,
                                     rep(height.deviant,length(Use.deviant)-1),
                                     heightt.legend))

  par(mar=marplot,oma=omaplot)

  # 1 column for each cluster (j)
  for(j in (1:number.cl)[blocks==w]){
    id.first<-(1:number.cl)[blocks==w][1]

    # clusters exist
    if(!is.na(labels.cl[j])){
      sel.cl<-seq.clusters==labels.cl[j]

      # check where to place horiz axis
      place.hor<-NULL
      for(ss in 1:length(Use.deviant)){
        place.hor<-c(place.hor,sum((sel.cl==TRUE) & (Use.deviant[[ss]]==TRUE)))
      }

      # last plot in the column (cluster)
      place.hor<-max((1:length(place.hor))[place.hor>1])
    }
  }
}

```

```

# Regular cases
sel.cl.reg<-(sel.cl==TRUE) & (Use.deviant$Regular==TRUE)

# order sequences using TSP
# might take some time when the nr of cases is high
set.seed(100)
useTSP<-seriate(as.dist(seq.dist[sel.cl.reg==T,sel.cl.reg==T]),
               method="TSP")
useOrder<-as.matrix((get_order(useTSP)))

States.j.reg<-seq.num[sel.cl.reg==T,]
States.j.reg<-t(as.matrix(States.j.reg[useOrder,]))
par( yaxt="s")
image( x=1:nrow(States.j.reg), z=States.j.reg, y=1:ncol(States.j.reg),
       zlim=c(1,length(Lab.Col_states)),xlim=c(1,nrow(States.j.reg)),
       axes=F,xlab="",ylab="",
       cex.lab=1.2,font.lab=2,col=(Lab.Col_states),main="",cex.main=1.5)

box()

mypretty<-pretty(1:sum(sel.cl.reg),5)
mypretty<-mypretty[mypretty>0]
axis(side = 2, tck = -.02, at=mypretty,labels = NA)
axis(side = 2, lwd = 0, line = -.6, las = 1,
     at=mypretty,labels=mypretty,cex.axis=1)

# if last plot in the column (cluster) - no deviant seq. -
# draw horizontal axis

if(place.hor==1){
  axis(side = 1, tck = -.02, labels = NA)
  axis(side = 1, lwd = 0, line = -.8, las = 1,
       labels=T,cex.axis=1)}
mtext(side = 3,paste0(labcl,labels.cl[j]), line = 0.2,cex=0.8,font=2)

# Deviant sequences
for(cc in 2:length(Use.deviant)){
  sel.cl.cc<-(sel.cl==TRUE) & (Use.deviant[[cc]]==TRUE)

  # if there are deviant sequences plot:
  if(sum(sel.cl.cc)>1){
    set.seed(100)
    useTSP<-seriate(as.dist(seq.dist[sel.cl.cc==T,sel.cl.cc==T]),
                   method="TSP")
    useOrder<-as.matrix((get_order(useTSP)))

    States.j.cc<-seq.num[sel.cl.cc==T,]
    States.j.cc<-t(as.matrix(States.j.cc[useOrder,]))

    image( x=1:nrow(States.j.cc), z=States.j.cc, y=1:ncol(States.j.cc),
          zlim=c(1,length(Lab.Col_states)),xlim=c(1,nrow(States.j.cc)),
          axes=F,xlab="",ylab="",cex.lab=1.2,font.lab=2,
          col=(Lab.Col_states),main="",cex.main=1.5)
  }
}

```



```

box()

mypretty<-pretty(1:sum(sel.cl.cc),3)
mypretty<-mypretty[mypretty>0]
mypretty<-unique(round(mypretty,0))
axis(side = 2, tck = -.02, at=mypretty,labels = NA)
axis(side = 2, lwd = 0, line = -.6, las = 1,
      at=mypretty,labels=mypretty,cex.axis=1)

# if first column: add deviants label (a., b., or c.)
if(j==id.first){
  mtext(side = 2,names(Use.deviant)[cc], line = 1.7,
        cex=0.8,font=2,las=1)}

# if last plot in the column (cluster) draw horizontal axis
if(cc==place.hor ){
  axis(side = 1, tck = -.02, labels = NA)
  axis(side = 1, lwd = 0, line = -.8, las = 1,
        labels=T,cex.axis=1)
}
}

if(sum(sel.cl.cc)<=1){
  plot(1:4,1:4,type="n",axes=F)
  if(j==id.first){
    mtext(side = 2,names(Use.deviant)[cc], line = 1.7,
          cex=0.8,font=2,las=1)}
}

} # closes plots deviant
} # closes exist label

# if there is no cluster plot empty plots
if(is.na(labels.cl[j])){
  for(cc in 1:length(Use.deviant)){plot(1:4,1:4,type="n",axes=F) }
} # closes label does not exist

} # closes clusters in the plot

# Add legend
par(mar=c(0,0.1,0.1,0) )
plot(1:7, 1:7,type= "n", axes = FALSE, ann = FALSE)
legend(x=1,y=5.5,names(Lab.Col_states),pch = 22 ,
       pt.bg=Lab.Col_states,ncol=ncol.legend,cex=1.1,
       pt.cex=1.5,x.intersp=1,y.intersp=1 , bg="white" )

par(par.or)

} # closes blocks

```

