Adaptive Calibration

Paper notebook - submitted to ...

O.Mirones, J.Bedia, S.Herrera & M.Iturbide

2023-02-03

Abstract

This example notebook ilustrates the adaptive calibration methodology developed in (name of paper), providing full reproducibility. Thus, we present the methodology for the Kolopelu station, located in the French Overseas Collectivity of Wallis and Futuna in the South Pacific region. To this aim, we use a dataset [https://zenodo.org/record/7014397/files/South_Pacific_precipitation.zip] generated in our previous work Mirones *et al.*(2022) which collects the rain gauge (PACRAIN) and satellite precipitation (TRMM) and the weather types (WTs) associated with each date.

Contents

Used packages. Installing climate4R

To ensure full reproducibility of the results, it is strongly recommended to install the packages versions used to avoid errors compiling. All operations hereinafter are performed with the core packages of climate4R, excepting package installation. The appropriate package versions are indicated here through their version tags using the `devtools` package function `install_github` (Wickham *et al.* 2020).

```
library(devtools)
install_github(c("SantanderMetGroup/loadeR.java@1.1.1",
                 "SantanderMetGroup/climate4R.UDG",
                 "SantanderMetGroup/loadeR@1.7.0",
                 "SantanderMetGroup/transformeR@time_res",
                 "SantanderMetGroup/visualizeR",
                 "SantanderMetGroup/downscaleR@oscar_devel",
                 "SantanderMetGroup/climate4R.value"))
```

```
library(loadeR)
library(transformeR)
library(downscaleR)
library(climate4R.value)
```

Adaptive calibration in Kolopelu station

Loading, collocating and harmonizing data

First, we load the required dataset resulting from the study of Mirones *et al.*(2022) located at Zenodo open repository [https://zenodo.org/record/7014397/files/South_Pacific_precipitation.zip].

```
temp <- tempfile(fileext = ".zip")
download.file("https://zenodo.org/record/7014397/files/South_Pacific_precipitation.zi
```

The dataset downloaded contains a set of stations located in the South Pacific region, obtained from the **PACRAIN** (Pacific Rainfall Database) [http://pacrain.ou.edu/]. Similarly, the **TRMM** (Tropical Rainfall Measuring Mission) satellite precipitation series [https://trmm.gsfc.nasa.gov/] and the **ERA5** reanalysis precipitation series [https://www.ecmwf.int/en/forecasts/datasets/reanalysis-datasets/era5] have been extracted for the Kolopelu (Wallis and Futuna), Alofi (Niue), Rarotonga (Cook Islands), Raoul Island (New Zealand), Port Vila (Vanuatu), Aoloau and Nuu'uli (American Samoa) stations. For the stations corresponding to IDs *NZ75400*, *NZ82400*, *NZ84317*, *NZ99701*, *SP00646*, *US14000* and *US14690*, the following daily variables are included:

- TRMM raw satellite precipitation
- ERA5 raw reanalysis precipitation
- PACRAIN rain gauge precipitation
- Weather type associated
- TRMM calibrated (scaling)
- TRMM calibrated (empirical quantile mapping)
- TRMM calibrated (scaling conditioned)
- TRMM calibrated (empirical quantile mapping conditioned)

where the last four are a consequence of the application of 4 different calibration methods.

The required variables are *rain gauge precipitation* (`pr`), *satellite precipitation* (`pp_trmm`) and the *WTs* (`wt`). Thus, we set the corresponding variable names in the argument `var` when calling to `loadStationData` from `loadeR`:

```
obs <- loadStationData(dataset = temp, var = "pr")
trmm <- loadStationData(dataset = temp, var = "pp_trmm")
wt <- loadStationData(dataset = temp, var = "wt")
```

For the variables extracted, we can select the station to study through the argument `station.id` (the station IDs appear in `$Metadata`) using `subsetGrid` function from `transformeR` package. We reproduce the methodology for Kolopelu station, since the code to deploy the adaptive methodology is analogous for all the stations (the unique change is the choice of `station.id` as argument in `subsetGrid`).

```
obs <- subsetGrid(obs, station.id = obs$Metadata$station_id[1])
trmm <- subsetGrid(trmm, station.id = obs$Metadata$station_id)
wt <- subsetGrid(wt, station.id = obs$Metadata$station_id)
```

Finally, we intersect the rain gauge and TRMM data, since their time coverage may differ. To solve it, we use `intersectGrid` also from `transformeR` package. The argument `which.return` specifies which grid is to be returned encompassing the overlapping time period. The observation series is returned since the target period to be calibrated is the TRMM period.

```
obs <- intersectGrid(obs, trmm, which.return = 1)
```

Adaptive calibration computation

Here, we perform the adaptive calibration for the Kolopelu TRMM series. The adaptive calibration methodology developed in the article is encapsulated in a function named as `adaptativeCalibration`. For a proper evaluation and inter-comparison of different calibration methods (scaling, eQM, pQM, gpQM-95 and gpQM-75) a single score calculation methodology has been followed as stated in Kotlarski *et al.* (2019). The calibration methods are applied separately for each WT, being the final calibrated series the result of joining the best calibration among the methods into a single time series encompassing the entire period. To decide which method is best for each WT we use the score based in Kotlarski *et al.* (2019). The score computing framework follows these steps:

1. Calculate a set of climate indices from VALUE [http://www.value-cost.eu/validationportal/app/#!indices] using the `climate4R.value` package, which serve as validation.
2. Compute the difference in absolute value with the climate indices of the observed series.
3. Normalize among all the methods the difference in absolute value for each index. The resulting score of the method is the average of the normalized values of the climate indices.

3.1. In the score computation it is possible to add different weights (where the sum of the weights is 1) to the normalized climate indices. In this way it is possible to obtain a score more focused on certain indices, i.e. on extreme precipitation (*R20p* or *P98WetAmount*).

The cross-validation scheme and the calibration methods are implemented in the calibration and empirical statistical downscaling tools available in the R package `downscaleR` (Bedia *et al.*, 2020) of the `climate4R` framework.

```
adaptativeCalibration <- function(obs, sim, clustering, methods,
                                  index, custom_function = NULL,
                                  weights = NULL,
                                  scaling.type = "multiplicative",
                                  window = NULL, theta = .95){

  if (sum(weights) > 1) {

    stop("The sum of the weigths must be lower than 1")

  }

  if ((length(weights) >0) && (length(weights) != (length(index)+length(custom_functi

    stop("Size of weights and index+custom functions are different")
  }

  if(length(which(methods == "gpqm")) != length(theta)){
```

```r
  stop(paste0("Number of gpqm methods and theta size are different. Must be the sam
              length(which(methods == "gpqm")), " != ", length(theta)))

}

final.list <- list()
scores.list <- list()
cal.method <- c()

#loop on the WTs

for (k in unique(clustering)) {

  gpqm.count <- 1

  aux.obs <- subsetDimension(obs, dimension = "time",
                             indices = which(clustering == k))
  aux.sim <- subsetDimension(sim, dimension = "time",
                             indices = which(clustering == k))

  n.folds <- trunc(length(aux.obs$Data)/275)

  if (n.folds <= 1) {
    n.folds <- 2
```

```r
}

index.obs <- c()
index.sim <- c()

for (i in c(1:length(index))) {

  index.obs <- c(index.obs,
                 valueIndex(aux.obs, index.code = index[i])$Index$Data)
  index.sim <- c(index.sim,
                 valueIndex(aux.sim, index.code = index[i])$Index$Data)

}

aux.obs <- setTimeResolution(aux.obs,
                             time_resolution = "DD")
aux.sim <- setTimeResolution(aux.sim,
                             time_resolution = "DD")

aux.obs <- setGridDates.asPOSIXlt(aux.obs,
                                  tz = "UTC")
aux.sim <- setGridDates.asPOSIXlt(aux.sim,
                                  tz = "UTC")

cal.list <- list()
```

```
index.list <- list()

#computation of the calibration methods for the WT-subseted data

for (j in c(1:length(methods))) {

  if (methods[j] == "scaling") {

    cal <- biasCorrection(y = aux.obs, x = aux.sim,
                          precipitation = TRUE,
                          method = "scaling",
                          scaling.type = scaling.type,
                          window = window,
                          cross.val = "kfold",
                          folds = n.folds)

  }else if (methods[j] == "eqm") {

    cal <- biasCorrection(y = aux.obs, x = aux.sim,
                          precipitation = TRUE,
                          method = "eqm",
                          cross.val = "kfold",
                          folds = n.folds)

  }else if (methods[j] == "pqm") {
```

```
    cal <- biasCorrection(y = aux.obs, x = aux.sim,
                          precipitation = TRUE,
                          method = "pqm",
                          cross.val = "kfold",
                          folds = n.folds)

  }else if (methods[j] == "gpqm") {



    if(length(which(methods == "gpqm")) > 1){

      cal <- biasCorrection(y = aux.obs, x = aux.sim,
                            precipitation = TRUE,
                            method = "gpqm",
                            theta = theta[gpqm.count],
                            cross.val = "kfold",
                            folds = n.folds)

      gpqm.count <- gpqm.count + 1

    }else{

      cal <- biasCorrection(y = aux.obs, x = aux.sim,
                            precipitation = TRUE,
```

```
                                method = "gpqm",
                                theta = theta,
                                cross.val = "kfold",
                                folds = n.folds)

  }

}




cal <- subsetDimension(cal, dimension = "time",
                       indices = which(!is.na(cal$Data)))


cal$Dates$start <- as.POSIXct(cal$Dates$start,tz = "GMT")
cal$Dates$end <- as.POSIXct(cal$Dates$end,tz = "GMT")

#computation of climate indices from VALUE


index.cal <- c()
for (i in c(1:length(index))) {

  index.cal <- c(index.cal, valueIndex(grid = cal,
                                       index.code = index[i])$Index$Data)
```

```
  }

  aux.obs$Dates$start <- as.POSIXct(aux.obs$Dates$start, tz = "CEST")
  aux.obs$Dates$end <- as.POSIXct(aux.obs$Dates$end, tz = "CEST")

  #computation of custom functions as additional indices
  if (length(custom_function) > 0) {
    for (i in c(1:length(custom_function))) {
      index.obs <- c(index.obs,
                     custom_function[[i]](aux.obs))
      index.cal <- c(index.cal,
                     custom_function[[i]](cal))

    }

  }

  cal.list[[j]] <- cal
  index.list[[j]] <- index.cal
}

names(cal.list) <- methods
names(index.list) <- methods

#function which normalize the values for a certain index
```

```r
normalization <- function(measure){
  measure.norm <- c()
  for (i in c(1:length(measure))) {
    measure.norm <- c(measure.norm,
                      1-((measure[i]-min(measure))/(max(measure)-min(measure))))
  }
  return(measure.norm)
}

#computation of the absolute bias of the indices of each calibration method wit

measures <- list()

for (i in c(1:length(index.cal))) {
  aux <- c()
  for (j in c(1:length(methods))) {
    aux <- c(aux, abs(index.list[[j]][i]-index.obs[i]))
  }
  measures[[length(measures)+1]] <- aux
}

aux <- NULL

#values normalization for the different indices
```

```r
norm.vector <- list()
for (i in c(1:length(measures))) {
  norm.vector[[length(norm.vector)+1]] <- normalization(measures[[i]])
}

#calculation of the scores for the different methods

scores <- c()
for (j in c(1:(length(methods)))) {
  score <- c()
  for (i in c(1:length(measures))) {

    score <- c(score,norm.vector[[i]][j])

  }
  if (length(weights > 0)) {

    score <- weighted.mean(score, w = weights)

  }else{

    score <- mean(score)

  }
```

```
    scores <- c(scores, score)
  }

  names(scores) <- methods

  if (length(which(methods == "gpqm")) > 1) {

    idx <- which(names(scores) == "gpqm")
    for (i in c(1:length(which(methods == "gpqm")))) {

      names(scores)[idx[i]] <- paste0("gpqm","-",theta[i])

    }
  }



  scores.list[[length(scores.list)+1]] <- scores

  final.list[[length(final.list)+1]] <- cal.list[[order(scores,
                                                   decreasing = T)[1]]]

  cal.method <- c(cal.method,names(scores)[order(scores,
                                                decreasing = T)][1])
}
```

```
  #merging of the best calibrations per WT

  output <- bindGrid(final.list[[1]],
                     final.list[[2]],
                     dimension = "time")

  #if the number of WTs is greater than 2 to merge the remainder is applied in a lo

  if (length(final.list) > 2) {

    for (i in c(3:length(final.list))) {
      output <- bindGrid(output,
                         final.list[[i]],
                         dimension = "time")
    }
  }

  attr(output$Data, "dimensions") <- "time"
  attr(output$Data, "weather_types") <- unique(clustering)
  attr(output$Data, 'adaptative_calibration') <- cal.method
  attr(output$Data, "RF_scores") <- scores.list

  return(output)
}
```

Our validation of the results is based on VALUE climatic indices. However, in the adaptativeCalibration function we can add other indices elaborated by ourselves, using the custom_function argument, such as with the MaxReturnValue index.

```
MaxReturnValue <- function(data, indices = NULL){
  require(evd)
  data$Dates$start <- as.POSIXct(data$Dates$start,
                                tz = "GMT")
  data$Dates$end <- as.POSIXct(data$Dates$end,
                              tz = "GMT")
  if(!is.null(indices)){
    subsetDimension(grid = data, dimension = "time",
                    indices = indices)
  }
  nyears <- 20
  data.maxYear <- aggregateGrid(redim(data),
                               aggr.y = list(FUN = "max",na.rm = TRUE))

  data.maxrv <- climatology(data,
                           clim.fun = list(FUN = "mean", na.rm = T))

  auxData <- data.maxYear$Data
  auxData[which(is.infinite(auxData))] <- NA

  if (any(!is.na(auxData))){
```

```
    auxGEV <- fgev(auxData)
    if ((auxGEV$estimate[3] - auxGEV$std.err[3] < 0) & (0  < auxGEV$estimate[3] + aux
      auxGEV <- fgev(auxData, shape = 0)
      auxRV <- qgev(1-1/nyears,
                    loc = auxGEV$estimate[1],
                    scale = auxGEV$estimate[2],
                    shape = 0)
    }else{
      auxRV <- qgev(1-1/nyears,
                    loc = auxGEV$estimate[1],
                    scale = auxGEV$estimate[2],
                    shape = auxGEV$estimate[3])
    }
    data.maxrv <- as.numeric(auxRV)
    names(data.maxrv) <- paste("RV",nyears,"_max", sep = "")
  }
  return(data.maxrv)

}
```

Then, we perform the adaptive calibration using the **scaling**, **eQM**, **pQM** and **gpQM** (with 0.95 and 0.75 as thresholds) techniques. Also these aforementioned methods are computed independently as well to evaluate if the adaptive calibration results improve the standard calibration methods.

```r
cal <- adaptativeCalibration(obs = obs, sim = trmm,
                             clustering = wt$Data,
                             methods = c("scaling", "eqm", "pqm", "gpqm", "gpqm"),
                             index = c("Skewness","Mean","SDII","R10","R10p","R20","R
                             weights = NULL,
                             custom_function = list(MaxReturnValue),
                             scaling.type = "multiplicative",
                             window = NULL,
                             theta = c(.95, .75))
```

Standard techniques computation

Once the series calibrated with the adaptive methodology is computed, we need to obtain the series of standard calibrations to calculate the scores of the series and determine if the adaptive methodology obtains better results. Depending on the calibration method used, the `biasCorrection` function, from `downscaleR` package, displays arguments specific to the technique in question:

- The argument `multiplicative` indicates the type of the scaling method. Options available are "additive" (preferable for unbounded variables, i.e. temperature) or "multiplicative" (bounded variables, i.e. precipitation). This argument is ignored if "scaling" is not selected as the bias correction method.
- Through `fitdistr.args` we can choose further arguments passed to function `fitdistr` (densfun, start, ...). Only used when applying the "pqm" method.

- The argument `theta` indicates the upper threshold (and lower for the left tail of the distributions, if needed) above which values are fitted to a Generalized Pareto Distribution (GPD). Values below this threshold are fitted to a gamma distribution. By default, 'theta' is the 95th percentile (and 5th percentile for the left tail). Exclusive for "gpqm" method.

Thus, we compute the standard calibration for scaling, eQM, pQM and gpQM. With the gpQM method two series are computed named as gpQM-95 and gpQM-75. The first fits to a GPD above the 95th percentile and the second the 75th percentile.

```
obs <- setGridDates.asPOSIXlt(obs,
                              tz = "UTC")
trmm <- setGridDates.asPOSIXlt(trmm,
                               tz = "UTC")
```

```
scaling <- biasCorrection(y = obs, x = trmm,
                          precipitation = TRUE,
                          method = "scaling",
                          scaling.type = "multiplicative",
                          window = NULL,
                          cross.val = "loo")
```

```
eqm <- biasCorrection(y = obs, x = trmm,
                      precipitation = TRUE,
                      method = "eqm",
                      cross.val = "loo")
```

```r
pqm <- biasCorrection(y = obs, x = trmm,
                      precipitation = TRUE,
                      method = "pqm",
                      cross.val = "loo")
```

```r
gpqm95 <- biasCorrection(y = obs, x = trmm,
                         precipitation = TRUE,
                         method = "gpqm",
                         theta = .95,
                         cross.val = "loo")
```

```r
gpqm75 <- biasCorrection(y = obs, x = trmm,
                         precipitation = TRUE,
                         method = "gpqm",
                         theta = .75,
                         cross.val = "loo")
```

Ranking Framework (RF) Score. Choosing the best calibrated series

From the `adaptativeCalibration` function we extract the part where the **RF Score** is computed to evaluated which method is the best for a certain WT. The idea is use this “new” function to assess if the adaptive methodology results improve those obtained from the standard calibration methods.

```r
scoreComputation <- function(obs, series, index, custom_function = NULL, methods){

index.obs <- c()

for (i in c(1:length(index))) {

  index.obs <- c(index.obs,
                 valueIndex(obs, index.code = index[i])$Index$Data)



}
aux.obs <- setTimeResolution(obs,
                             time_resolution = "DD")
aux.obs <- setGridDates.asPOSIXlt(aux.obs,
                                  tz = "UTC")



index.list <- list()
for (j in c(1:length(methods))) {
  index.cal <- c()
  for (i in c(1:length(index))) {

    index.cal <- c(index.cal,
                   valueIndex(grid = series[[j]], index.code = index[i])$Index$Data)
```

```
  }

  aux.obs$Dates$start <- as.POSIXct(aux.obs$Dates$start,
                                    tz = "CEST")
  aux.obs$Dates$end <- as.POSIXct(aux.obs$Dates$end,
                                  tz = "CEST")

  if (length(custom_function) > 0) {
    for (i in c(1:length(custom_function))) {
      index.obs <- c(index.obs,
                     custom_function[[i]](aux.obs))
      index.cal <- c(index.cal,
                     custom_function[[i]](series[[j]]))

    }

  }

  index.list[[j]] <- index.cal
}

names(index.list) <- methods
```

```
normalization <- function(measure){
  measure.norm <- c()
  for (i in c(1:length(measure))) {
    measure.norm <- c(measure.norm,
                      1-((measure[i]-min(measure))/(max(measure)-min(measure))))
  }
  return(measure.norm)
}

measures <- list()

for (i in c(1:length(index.cal))) {
  aux <- c()
  for (j in c(1:length(methods))) {
    aux <- c(aux, abs(index.list[[j]][i]-index.obs[i]))
  }
  measures[[length(measures)+1]] <- aux
}

aux <- NULL

norm.vector <- list()
for (i in c(1:length(measures))) {
  norm.vector[[length(norm.vector)+1]] <- normalization(measures[[i]])
```

```
}



scores <- c()
for (j in c(1:(length(methods)))) {
  score <- c()
  for (i in c(1:length(measures))) {

    score <- c(score,norm.vector[[i]][j])

  }

  score <- mean(score)


  scores <- c(scores, score)
}


names(scores) <- methods


return(scores)
}
```

Finally, the RF score is computed using `scoreComputation`. The argument `obs` corresponds to the rain gauge series, `series` indicates the list of calibrations from which the score is to be computed, in `index` we set the validation indices from VALUE and also the `MaxReturnValue`, which is included as part of validation indices, set in `custom_function` argument.

```
score <- scoreComputation(obs = obs,
                          series = list(scaling, eqm, pqm, gpqm95, gpqm75, cal),
                          index = c("Skewness","Mean","SDII","R10","R10p","R20","R20p
                          custom_function = list(MaxReturnValue),
                          methods = c("scaling", "eqm", "pqm", "gpqm95", "gpqm75", "a


score
```

```
##   scaling       eqm       pqm    gpqm95    gpqm75  adaptive
## 0.3715607 0.5870230 0.5323443 0.5169643 0.5630853 0.7326085
```

The results show that with the adaptive methodology we obtain a significantly better score than with the other techniques, which reinforces the approach of this type of calibration.

References

- Bedia, J., Baño-Medina, J., Legasa, M. N., Iturbide, M., Manzanas, R., Herrera, S., Casanueva, A., San-Martín, D., Cofiño, A. S., and Gutiérrez, J. M.: Statistical downscaling with the downscaleR package (v3.1.0): contribution to the VALUE intercomparison experiment, Geoscientific Model Development, 13, 1711–1735, https://doi.org/10.5194/gmd-13-1711-2020, 2020.

- Maraun, D., Widmann, M., Gutiérrez, J. M., Kotlarski, S., Chandler, R. E., Hertig, E., Wibig, J., Huth, R., and Wilcke, R. A.: VALUE: A framework to validate downscaling approaches for climate change studies, Earth's Future, 3, 1–14, https://doi.org/https://doi.org/10.1002/2014EF000259, 2015
- Kotlarski, S., Szabó, P., Herrera, S., Räty, O., Keuler, K., Soares, P. M., Cardoso, R. M., Bosshard, T., Pagé, C., Boberg, F., Gutiérrez, J. M., Isotta, F. A., Jaczewski, A., Kreienkamp, F., Liniger, M. A., Lussana, C., and Pianko-Kluczynska, K.: Observational uncertainty and regional climate model evaluation: A pan-European perspective, International Journal of Climatology, 39, 3730–3749, https://doi.org/https://doi.org/10.1002/joc.5249, 2019.
- Mirones, O., Bedia, J., Fernández-Granja, J. A., Herrera, S., Van Vloten, S. O., Pozo, A., Cagigal, L., and Méndez, F. J.: Weather-type-conditioned calibration of Tropical Rainfall Measuring Mission precipitation over the South Pacific Convergence Zone, International Journal of Climatology, pp. 1–18, https://doi.org/https://doi.org/10.1002/joc.7905, 2022.
- Wickham, H., Hester, J. and Chang, W., 2020. devtools: Tools to Make Developing R Packages Easier. R package version 2.3.0. https://CRAN.R-project.org/package=devtools

Session info

```r
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.4 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=es_ES.UTF-8          LC_NUMERIC=C
##  [3] LC_TIME=es_ES.UTF-8           LC_COLLATE=es_ES.UTF-8
##  [5] LC_MONETARY=es_ES.UTF-8       LC_MESSAGES=es_ES.UTF-8
##  [7] LC_PAPER=es_ES.UTF-8          LC_NAME=es_ES.UTF-8
##  [9] LC_ADDRESS=es_ES.UTF-8        LC_TELEPHONE=es_ES.UTF-8
## [11] LC_MEASUREMENT=es_ES.UTF-8    LC_IDENTIFICATION=es_ES.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] evd_2.3-3             climate4R.value_0.0.2 VALUE_2.2.2
##  [4] downscaleR_3.3.3      transformeR_2.1.4     loadeR_1.7.0
```

```
##  [7] climate4R.UDG_0.2.0   loadeR.java_1.1.1     rJava_0.9-11
## [10] devtools_2.3.2        usethis_1.6.3
##
## loaded via a namespace (and not attached):
##  [1] CircStats_0.2-6     bitops_1.0-7        fs_1.5.0
##  [4] rprojroot_2.0.3     sticky_0.5.6.1      tools_3.6.3
##  [7] utf8_1.2.3          R6_2.5.1            colorspace_2.1-0
## [10] withr_2.5.0         sp_1.6-0            tidyselect_1.2.0
## [13] gridExtra_2.3       prettyunits_1.1.1   processx_3.7.0
## [16] compiler_3.6.3      glmnet_4.1-3        cli_3.6.0
## [19] desc_1.2.0          scales_1.2.1        proxy_0.4-26
## [22] dtw_1.22-3          callr_3.5.1         pbapply_1.5-0
## [25] stringr_1.4.0       digest_0.6.30       rmarkdown_2.5
## [28] pkgconfig_2.0.3     htmltools_0.5.0     akima_0.6-3.4
## [31] sessioninfo_1.1.1   maps_3.4.1          rlang_1.0.6
## [34] rstudioapi_0.14     shape_1.4.6         generics_0.1.3
## [37] jsonlite_1.8.3      dplyr_1.0.9         RCurl_1.98-1.5
## [40] magrittr_2.0.3      verification_1.42   dotCall64_1.0-2
## [43] Matrix_1.5-1        Rcpp_1.0.10         munsell_0.5.0
## [46] fansi_1.0.4         abind_1.4-5         reticulate_1.26
## [49] viridis_0.6.2       lifecycle_1.0.3     stringi_1.5.3
## [52] yaml_2.3.6          MASS_7.3-53         pkgbuild_1.1.0
## [55] grid_3.6.3          parallel_3.6.3      crayon_1.5.1
## [58] lattice_0.20-41     splines_3.6.3       knitr_1.39
## [61] ps_1.7.1            pillar_1.8.1        boot_1.3-25
```

```
## [64] codetools_0.2-16    pkgload_1.1.0       glue_1.6.2
## [67] evaluate_0.15       kohonen_3.0.11      remotes_2.2.0
## [70] png_0.1-7           vctrs_0.5.2         spam_2.9-1
## [73] foreach_1.5.1       testthat_2.3.2      gtable_0.3.1
## [76] assertthat_0.2.1    ggplot2_3.4.0       xfun_0.30
## [79] RcppEigen_0.3.3.9.3 survival_3.2-7      viridisLite_0.4.1
## [82] signal_0.7-7        tibble_3.1.8        iterators_1.0.13
## [85] memoise_1.1.0       fields_14.1         deepnet_0.2
## [88] ellipsis_0.3.2
```