Statistical downscaling with `climate4R`: Contribution to the VALUE Intercomparison experiment

Paper notebook - submitted to Environmental Modelling & Software

J. Bedia, J. Baño-Medina, M.N. Legasa, M. Iturbide, R. Manzanas, S. Herrera, D. San Martín, A.S. Cofiño & J. M Gutiérrez

2019-12-06

Abstract

The R package `downscaleR` for statistical downscaling of climate information (SD) covers the most popular approaches (Model Output Statistics –including the so called "bias correction" methods– and Perfect Prognosis) and state-of-the-art techniques. It has been conceived to work primarily with daily data and can be used in the framework of both seasonal forecasting and climate change studies. Its full (Iturbide *et al.* 2019) makes possible the development of end-to-end downscaling applications, from data retrieval to model building, validation and prediction, bringing to climate scientists and practitioners a unique comprehensive framework for SD model development. This notebook reproduces the results presented in the paper, showcasing the main characteristics and functioning of perfect-prog downscaling with `climate4R`, including its integration with the VALUE validation framework a comprehensive evaluation of SD methods. integration within the climate4R framework (Project VALUE, http://www.value-cost.eu) that allows for undertaking

Contents

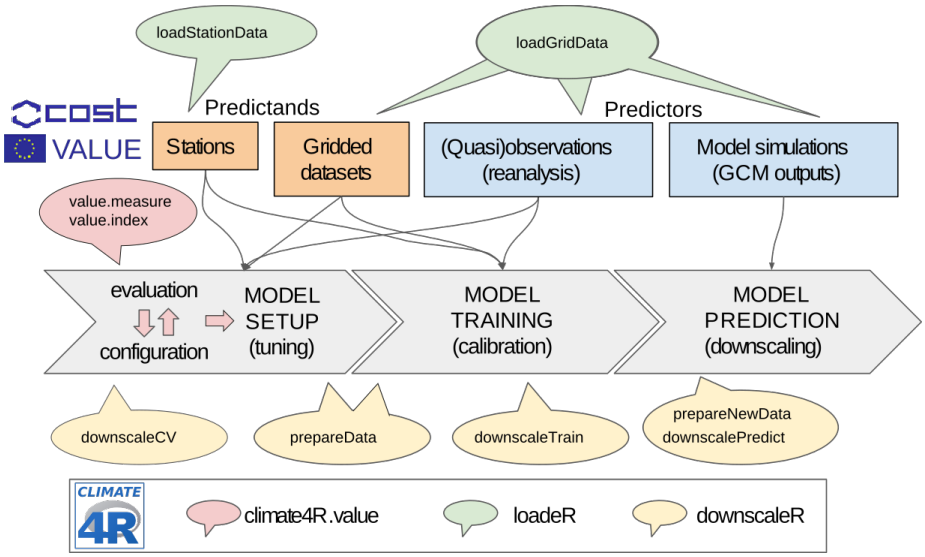


Figure 1: Schematic overview of the R package downscaleR and its framing into the climate4R framework for climate data access and analysis.

1 Introduction

1.1 Package overview

The typical perfect-prog downscaling phases are indicated in the figure above by the grey arrows:

1. In first place, model setup is undertaken. This process is iterative and usually requires testing many different model configurations under a cross-validation set up until an optimal configuration is achieved. The `downscaleCV` function (and `prepareData` under the hood) is used in this stage for a fine-tuning of the model. This part is illustrated through Section 2 and sections 3.3 and 3.4. The suitability of the calibrated model is determined through the use of specific indices and measures reflecting model suitability for different aspects that usually depend on specific research aims (e.g. good reproducibility of extreme events, temporal variability, spatial dependency across different locations etc.). The validation is achieved through the `climate4R.value` package (red-shaded callout), implementing the VALUE validation framework (Sections 2.4 and 3.5).
2. Model training: once an optimal model is achieved, model training is performed using the `downscaleTrain` function (Section 4).
3. Finally, the calibrated model is used to undertake downscaling (i.e. model predictions) using the function `downscalePredict`. The data to be used in the predictions requires appropriate preprocessing (e.g. centering and scaling using the predictor set as reference, projection of PC's onto predictor EOF's, etc.) that is

performed under the hood by function `prepareNewData` prior to model prediction with `downscalePredict`. This is illustrated in Section 4, where future downscaled projections for a CMIP5 GCM are calculated.

1.2 Package installation

To ensure the reproducibility of the paper results as accurately as possible, it is recommended to install the package versions used to compile this notebook. The appropriate package versions are indicated here through their version tags using the `devtools` package function `install_github` (Wickham *et al.* 2018):

```
devtools::install_github(c("SantanderMetGroup/loadeR.java@v1.1.1",
                           "SantanderMetGroup/loadeR@v1.4.14",
                           "SantanderMetGroup/transformeR@v1.5.1",
                           "SantanderMetGroup/downscaleR@v3.1.0",
                           "SantanderMetGroup/visualizeR@v1.4.0",
                           "SantanderMetGroup/VALUE@v2.1.1",
                           "SantanderMetGroup/climate4R.value@v0.0.1"))
```

Alternatively, and updated image of the packages can be installed using the conda recipe for climate4R.

1.3 Cloud computing with the climate4R Hub

Furthermore, there is a docker `climate4R` installation available. The docker file also includes the jupyter framework enabling a direct usage

of `climate4R` via the **climate4R Hub**, a cloud-based computing facility to run `climate4R` notebooks on the cloud using the IFCA/CSIC Cloud Services).

2 Experiment 1 - Testing SD methods for downscaling in the Iberian Peninsula

The `climate4R` packages used in this paper are next loaded:

```
require(loadeR)
require(transformeR)
require(downscaleR)
require(visualizeR) #
require(climate4R.value)
```

Additional packages will be used for convenience. For instance, the package `magrittr` (Bache and Wickham 2014) allows to conveniently concatenate functions via the pipe operator `%>%`

```
require(magrittr)
```

In order to keep Experiment 1 and 2 self-contained, tha data are independently read in both cases, even though the dataset used for this experiment is a subset of the pan-European datasets used for Experiment 2. Therefore, in this section we first delimit the Iberian Peninsula subregion from the PRUDENCE regions used in VALUE (these are indicated in the paper, with further details). The vector layer delimiting these regions is a built-in dataset in package `visualizeR`, and therefore extracting the bounding box for Iberia is straightforward:

```
data("PRUDENCEregions", package = "visualizeR")
```

```
names(PRUDENCEregions)
```

```
## [1] "BI" "IP" "FR" "ME" "SC" "AL" "MD" "EA"
```

```
bb <- PRUDENCEregions["IP"]@bbox
```

```
lonLim <- bb[1,]
```

```
latLim <- bb[2,]
```

The `lonLim` and `latLim` vectors are used in the following to consider the Iberia subregion (names 'IP') as bounding box for data load.

2.1 Loading predictors

```
loginUDG(username = "*****", password = "*****")
```

```
var.list <- c("psl","tas","ta@500","ta@700","ta@850","hus@500","hus@850","z@500")
```

```
grid.list <- lapply(var.list, function(x) {
   loadGridData(dataset = "ECMWF_ERA-Interim-ESD",
                var = x,
                lonLim = lonLim,
                latLim = latLim,
                years = 1979:2008)
   }
)
```

```
x <- makeMultiGrid(grid.list)
```

2.2 Loading predictands

```
value <- file.path(find.package("VALUE"), "example_datasets", "VALUE_ECA_86_v2.zip")
y <- loadStationData(dataset = value,
                     lonLim = lonLim,
                     latLim = latLim,
                     var = "precip",
                     years = 1979:2008) %>% binaryGrid(condition = "GE",
                                                       threshold = 1,
                                                       partial = TRUE)
y_bin <- binaryGrid(y, condition = "GE", threshold = 1)
```

2.3 Testing methods

Building on the previous work by San-Martín *et al.* (2016) regarding predictor selection for precipitation downscaling, a number of predictor configuration alternatives is tested here. These are summarized in the Table below:

Table 1: Summary of predictor configurations tested. Local predictors always correspond to the original predictor fields previously standardized. Independent PCs are calculated separately for each predictor field, while combined PCs are computed upon the previously joined predictor fields. †The standardization in M5 is performed by subtracting to each grid cell the overall field mean, so the spatial structure of the predictor is preserved. Methods marked with an asterisk (*) are included in the VALUE intercomparison. Methods followed by the -L suffix (standing for 'Local') are used only in the pan-European experiment.

SD Method	*ID*	*Predictor configuration description*
GLM	M1	Spatial: n combined PCs explaining 95% of variance
GLM	M1-L	Spatial+local: M1 + first nearest gridbox
GLM	M2	Spatial: n independent PCs explaining 95% of the variance
GLM	M3	Local: first nearest gridbox
GLM	M4	Local: 4 nearest gridboxes
Analogs	M5	Spatial: original standardized† predictor fields
Analogs	M6	Spatial: n combined PCs explaining 95% of variance
Analogs	M6-L	Local: 25 nearest gridboxes
Analogs	M7	Spatial: n independent PCs explaining 95% of the variance

The fold list specifies the years composing each of the 5 subsamples for 5-fold cross-validation, following the VALUE experimental setup:

```
folds <- list(1979:1984, 1985:1990, 1991:1996, 1997:2002, 2003:2008)
```

All the predictor variables previously loaded in Section 2.1 are considered for all methods:

```
(vars <- getVarNames(x))
```

```
## [1] "psl"     "tas"     "ta@500"  "ta@700"  "ta@850"  "hus@500" "hus@850"
## [8] "z@500"
```

2.3.1 Method M1

Spatial predictor parameters. These arguments control how the Principal component analysis is carried-out, and are internally passed to the function `prinComp` of package `transformeR`. In this particular example (method M1), the (non rotated, combined) PCs explaining the 95% of total variance are retained (as in the rest of method, all the predictor variables are included).

```
spatial.pars.M1 <- list(which.combine = vars,
                        v.exp = .95,
                        rot = FALSE)
```

As no other type of predictors (global and/or local) are used in the M1 configuration, the defaults values (`NULL`) assumed by `downscaleCV` are applied. As the internal object containing the PCA information bears all the data inside, the argument `combined.only` serves to discard all the information but the combined PCs of interest. Therefore, with this simple specifications the cross-validation for method M1 is ready to be launched:

```
M1cv.bin <- downscaleCV(x = x, y = y_bin, method = "GLM",
                        family = binomial(link = "logit"),
                        folds = folds,
                        prepareData.args = list(global.vars = NULL,
                                                local.predictors = NULL,
                                                spatial.predictors = spatial.pars.M1
                                                combined.only = TRUE))
```

In the logistic regression model, the `downscaleCV` function returns a multigrid with two output prediction grids, storing two variables named `prob` and `bin`. The first contains the grid probability of rain for every day whereas the second is a binary prediction indicating wheter it rained or not. Thus, in this example the binary output is retained, applying the function `subsetGrid` along the `'var'` (variable) dimension:

```
M1cv.bin <- subsetGrid(M1cv.bin, var = "bin")
```

Note that the log link function can't deal with zeroes in the data for fitting a rain amount model. Here, a minimum threshold of 1 mm precipitation (`condition = "GE"`, i.e., Greater or Equal) is retained for GLM training of precipitation amount, following the VALUE criterion:

```
M1cv.cont <- downscaleCV(x = x, y = y, method = "GLM",
                         family = Gamma(link = "log"),
                         condition = "GE", threshold = 1,
                         folds = folds,
                         prepareData.args = list(global.vars = NULL,
                                                 local.predictors = NULL,
```

```
                                                    spatial.predictors = spatial.pars.M1,
                                                    combined.only = TRUE))
```

The continuous and binary predictions are now multiplied, so the precipitation frequency is adjusted and the final precipitation predictions are obtained:

```
M1cv <- gridArithmetics(M1cv.bin, M1cv.cont, operator = "*")
```

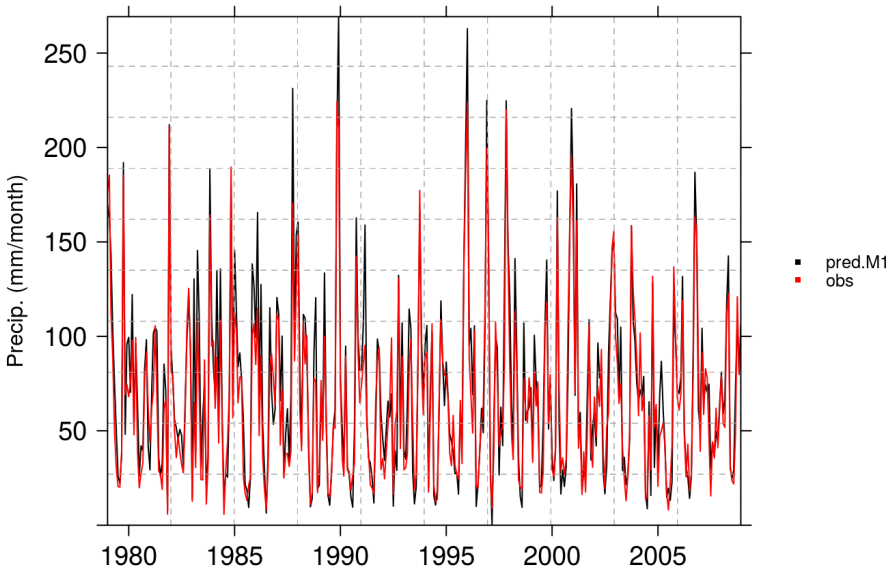
The final results can be handled for further analysis, as it is shown during method validation later in this Section. As an example common operation, here the (monthly accumulated and spatially averaged) predicted and observed time series are displayed using the function `temporalPlot` from package `visualizeR`:

Aggregation:

```
aggr.pars <- list(FUN = "sum", na.rm = TRUE)
## Monthly accumulated (sum) aggregation of predictions and observations:
pred.M1 <- aggregateGrid(M1cv, aggr.m = aggr.pars)
obs <- aggregateGrid(y, aggr.m = aggr.pars)
```

Plotting:

```
## Generates paper Fig. 3
temporalPlot(pred.M1, obs,
             xyplot.custom = list(xlab = "",
                                  ylab = "Precip. (mm/month)",
                                  scales = list(cex = 1.2,
                                                x = list(rot = 0))))
```



2.3.2 Method M2

Unlike M1, here the PCs are independently calculated for each variable, instead of considering one single matrix formed by all joined (combined) variables. To specify this PCA configuration, the spatial predictor parameter list is modified accordingly, by setting the `which.combine` argument to `NULL`. All the predictor variables stored in `x` are considered by default:

```
spatial.pars.M2 <- list(which.combine = NULL,
                        v.exp = .95,
                        rot = FALSE)
```

The rest of arguments passed to `downscaleCV` remain as in M1:

```
M2cv.bin <- downscaleCV(x = x, y = y_bin, method = "GLM",
```

```
                        family = binomial(link = "logit"),
                        folds = folds,
                        prepareData.args = list(global.vars = NULL,
                                                local.predictors = NULL,
                                                spatial.predictors = spatial.pars.M2,
                                                combined.only = FALSE)) %>%
  subsetGrid(var = "bin")
```

Similarly, the continuous GLM model M2 is cross-validated:

```
M2cv.cont <- downscaleCV(x = x, y = y, method = "GLM",
                         family = Gamma(link = "log"),
                         condition = "GE", threshold = 1,
                         folds = folds,
                         prepareData.args = list(global.vars = NULL,
                                                 local.predictors = NULL,
                                                 spatial.predictors = spatial.pars.M2,
                                                 combined.only = FALSE))
```

The final predictions are obtained as in M1:

```
M2cv <- gridArithmetics(M2cv.bin, M2cv.cont, operator = "*")
```

2.3.3 Method M3

Method M3 only uses local predictors. In this case, only the first closest neighbour (n=1) to the predictand location is used. Scaling parameters control how the raw predictor standardization (if any) is done. These parameters are passed to the function `scaleGrid` of package `transformeR`.

The arguments passed to `downscaleCV` are varied accordingly:

```
scaling.pars <- list(type = "standardize",
                     spatial.frame = "gridbox")
local.pars.M3 <- list(n = 1, vars = vars)


# Binary occurence model:
M3cv.bin <- downscaleCV(x = x, y = y_bin, method = "GLM",
                  family = binomial(link = "logit"),
                  folds = folds,
                  scaleGrid.args = scaling.pars,
                  prepareData.args = list(global.vars = NULL,
                                          local.predictors = local.pars.M3,
                                          spatial.predictors = NULL)) %>%
  subsetGrid(var = "bin")
# Continuous precip amount model:
M3cv.cont <- downscaleCV(x = x, y = y, method = "GLM",
                         family = Gamma(link = "log"),
                         condition = "GE", threshold = 1,
                         folds = folds,
                         scaleGrid.args = scaling.pars,
                         prepareData.args = list(global.vars = NULL,
                                                 local.predictors = local.pars.M3,
                                                 spatial.predictors = NULL))


# adjustment of frequencies and amount:
```

```
M3cv <- gridArithmetics(M3cv.bin, M3cv.cont, operator = "*")
```

2.3.4 Method M4

Method M4 is similar to M3, but considering a set of the 4 closest predictor gridboxes, instead of just one. Thus, the local predictor tuning parameters are slightly modified, by setting n = 4:

```
local.pars.M4 <- list(n = 4, vars = vars)


# Binary occurence model:
M4cv.bin <- downscaleCV(x = x, y = y_bin, method = "GLM",
                        family = binomial(link = "logit"),
                        folds = folds,
                        scaleGrid.args = scaling.pars,
                        prepareData.args = list(global.vars = NULL,
                                                local.predictors = local.pars.M4,
                                                spatial.predictors = NULL)) %>%
  subsetGrid(var = "bin")
# Continuous precip amount model:
M4cv.cont <- downscaleCV(x = x, y = y, method = "GLM",
                         family = Gamma(link = "log"),
                         condition = "GE", threshold = 1,
                         folds = folds,
                         scaleGrid.args = scaling.pars,
                         prepareData.args = list(global.vars = NULL,
                                                 local.predictors = local.pars.M4,
```

```
                                                    spatial.predictors = NULL))
# adjustment of frequencies and amount:
M4cv <- gridArithmetics(M4cv.bin, M4cv.cont, operator = "*")
```

2.3.5 Method M5

In method M5 the standardization is performed by centering every grid-box with respect to the overal spatial mean. To account for this particularity, the scaling parameters are modified accordingly, via the argument `spatial.frame`. Furthermore, the raw (standardized) predictors are used instead of PCs, without local predictors (these are indicated by the `global.vars` argument). The tuning parameters to achive this are next indicated:

```
scaling.pars.M5 <- list(type = "standardize", spatial.frame = "field")
```

In this case, the method for model training is set to `"analogs"`:

```
M5cv <- downscaleCV(x = x, y = y,
                    method = "analogs", n.analogs = 1,
                    folds = folds,
                    scaleGrid.args = scaling.pars.M5,
                    prepareData.args = list(global.vars = vars,
                                            local.predictors = NULL,
                                            spatial.predictors = NULL))
```

2.3.6 Method M6

The parameters used for predictor configuration in this method are similar to method M1. Thus, the previously defined argument is reused here:

```
M6cv <- downscaleCV(x = x, y = y,
                    method = "analogs", n.analogs = 1,
                    folds = folds,
                    prepareData.args = list(global.vars = NULL,
                                            local.predictors = NULL,
                                            spatial.predictors = spatial.pars.M1,
                                            combined.only = TRUE))
```

2.3.7 Method M7

In this method, the spatial parameters used for method M2 can be reused.

```
M7cv <- downscaleCV(x = x, y = y, method = "analogs",
                    folds = folds, n.analogs = 1,
                    prepareData.args = list(global.vars = NULL,
                                            local.predictors = NULL,
                                            spatial.predictors = spatial.pars.M2,
                                            combined.only = FALSE))
```

2.4 Validation of the methods

The validation tools available in VALUE have been adapted to the specific data structures of the `climate4R` framework through the wrapping package climate4R.value, enabling a direct application of the comprehensive

VALUE validation framework in downscaling exercises with `downscaleR`. A summary of the subset of VALUE indices used in this study is presented in the following table:

Code	*Description*	*Type*
R01	Relative frequency of wet days (precip $\geq$ 1mm)	index
Mean	Mean	index
SDII	Simple Daily Intensity Index	index
Skewness	Skewness	index
WWProb	Wet-wet transition probability (wet $\geq$ 1mm)	index
DWProb	Dry-wet transition probability (wet $\geq$ 1mm)	index
WetAnnualMaxSpell	Median of the annual wet ($\geq$ 1mm) spell maxima	index
DryAnnualMaxSpell	Median of the annual dry ($<$ 1mm) spell maxima	index
AnnualCycleAmp	Amplitude of the daily annual cycle	index
Var	Quasi-Variance	index
ratio	Ratio predicted/observed	measure[1]
ts.rs	Spearman correlation	measure[2]
ts.RMSE	Root Mean Square Error	measure[2]

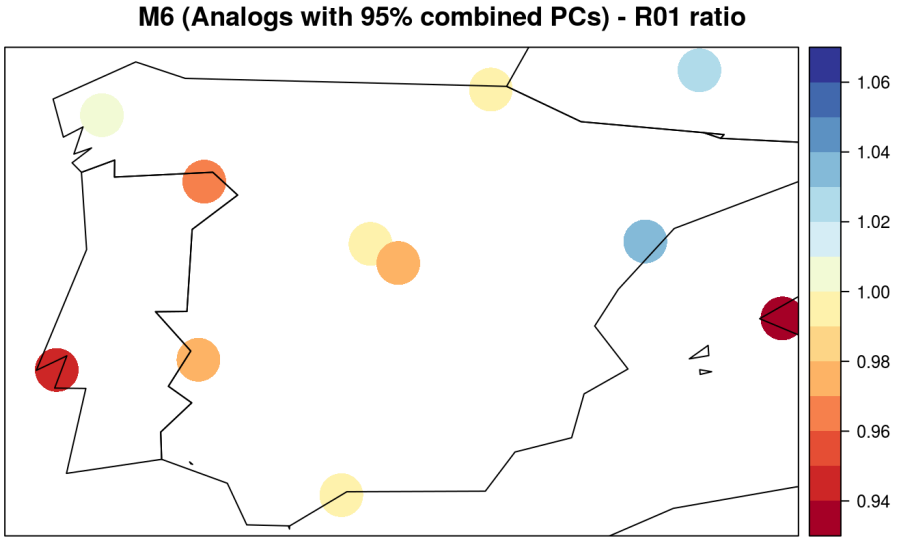
Table: Summary of the subset of VALUE validation indices and measures used in this study. Their codes are consistent with the VALUE reference list. The superindices in the measures indicate the input used to compute them: 1: a single scalar value, corresponding to the predicted and observed indices; 2: The original predicted and observed precipitation time series.

The user can also obtain an overview of the different indices and measures of the VALUE framework via the functions `show.indices()` and `show.measures()` available in the package `VALUE`. All the VALUE indices and measures are calculated by calling to the workhorse functions `valueIndex` and `valueMeasure`. For instance, the ratio is used as a measure to intercompare the predicted/observed frequency of wet days (greater of equal to 1 mm precip, VALUE code R01). To this aim, R01 is computed for both the predicted and observed time series, and then the ratio predicted/observed) is calculated. In the following example, the R01 ratio is calculated for method M1, considering the cross-validated model predictions:

```
index.ratio <- valueMeasure(y, x = M6cv,
                            measure.code = "ratio",
                            index.code = "R01")$Measure
```

A quick spatial plot helps to identify at which locations the frequency of wet days is under/over (red/blue) estimated by method M6:

```
## Generates paper Fig. 4
spatialPlot(index.ratio, backdrop.theme = "countries",
            cex = 4, cuts = seq(0.93, 1.07, 0.01), colorkey = TRUE,
            main = "M6 (Analogs with 95% combined PCs) - R01 ratio",
            ylim = latLim, xlim = lonLim)
```



In order to automate this task, the objects containing the cross validation results for the different methods are listed:

```
(methods <- ls(pattern = "^M.*cv$"))
```

```
## [1] "M1cv" "M2cv" "M3cv" "M4cv" "M5cv" "M6cv" "M7cv"
```

This function iterates over the methods in order to calculate R01 for each of them:

```
lapply(1:length(methods), function(i) {
  valueMeasure(y, x = get(methods[i]),
               measure.code = "ratio", index.code = "R01")$Measure
})
```

In order to simplify the code required, the function `my_validation` is next created, that recursively applies the `valueMeasure` function, as shown

above. The measure considered in all cases will be the *ratio* for the different set of summary indices:

```
my_validation <- function(measure.code = "ratio", index.code) {
  l <- lapply(1:length(methods), function(i) {
    suppressMessages(valueMeasure(y, x = get(methods[i]),
                                  measure.code = measure.code,
                                  index.code = index.code)$Measure)
  })
  names(l) <- methods
  return(l)
}
```

A vector containing the different VALUE indices to be computed is next generated to iterate over:

```
value.indices <- c("R01", "Mean", "SDII",
                   "Skewness","WWProb","DWProb",
                   "WetAnnualMaxSpell","DryAnnualMaxSpell",
                   "AnnualCycleRelAmp")
```

And the full validation results are calculated:

```
val.results <- sapply(value.indices, function(i) my_validation(index.code = i))
names(val.results) <- rep(methods, length(value.indices))
```

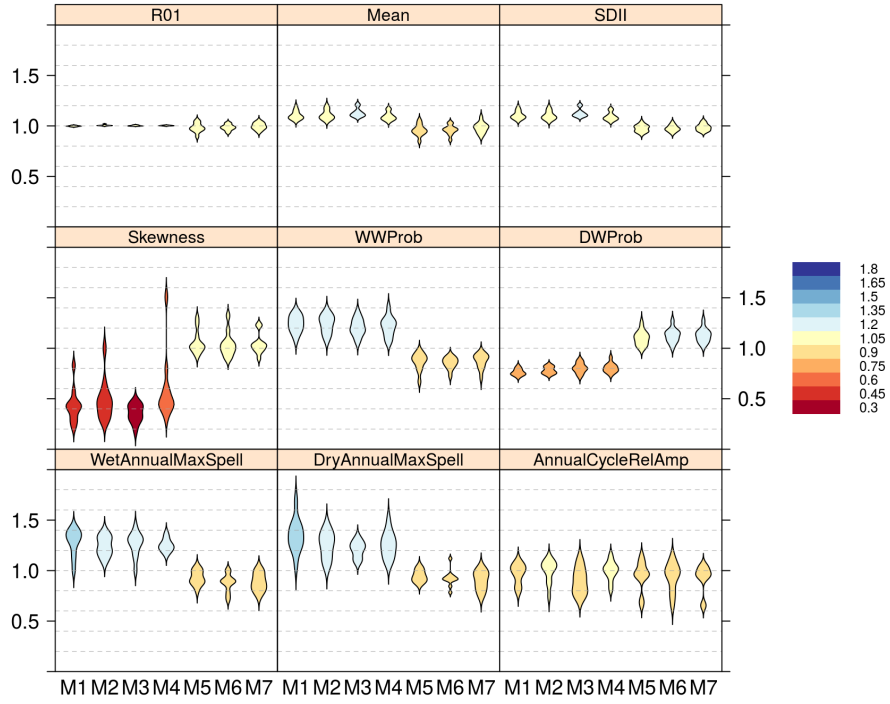
Next, some tuning parameters are indicated in order to prepare the violin plot summarizing the results:

```
# Panel groups arrangement
```

```
val.results[["group.index"]] <- rep(value.indices, each = length(methods))
# Graphical customization options
val.results[["bwplot.custom"]] <- list(layout = c(3, 3),
                                       ylim = c(0, 2),
                                       as.table = TRUE,
                                       scales = list(cex = 1.2,
                                       x = list(labels = paste0("M", 1:7))))
# Colorkey intervals (so colorkey is centered in 1)
val.results[["color.cuts"]] <- seq(0.30,1.85,.15)
```

And the final plot is done using the `visualizeR` function `violinPlot`:

```
do.call("violinPlot", val.results)
```



3 Experiment 2 - Pan-European experiments comparing local and non-local predictor methods

3.1 Data loading

Predictor are loaded considering the European domain determined by the following bounding box:

```
lonLim <- c(-10,32)
latLim <- c(36,72)

vars <- c("psl","tas","ta@500","ta@700","ta@850","hus@500","hus@850","z@500")
```

```
dataset <- "ECMWF_ERA-Interim-ESD"
grid.list <- lapply(vars, function(x) {
    loadGridData(dataset = dataset,
                 var = x,
                 lonLim = lonLim,
                 latLim = latLim,
                 years = 1979:2008)
})
x.eur <- makeMultiGrid(grid.list)
value <- file.path(find.package("VALUE"), "example_datasets", "VALUE_ECA_86_v2.zip")
y <- loadStationData(dataset = value,
                     var = "precip",
                     years = 1979:2008) %>% binaryGrid(condition = "GE",
                                                       threshold = 1,
                                                       partial = TRUE)
y_bin <- binaryGrid(y, condition = "GE", threshold = 1)
```

The following code prepares a map displaying the predictor set reference grid and the predictand locations:

```
coords.x <- get2DmatCoordinates(x.eur)
names(coords.x) <- c("x","y")
grid_clim <- climatology(subsetDimension(x.eur, dimension = "var", indices = 1))
spatialPlot(grid_clim,at = seq(-2, 2, 0.1), set.min = 4, set.max = 8,
            backdrop.theme = "countries",
            main = "Predictand locations and predictor grid",
```

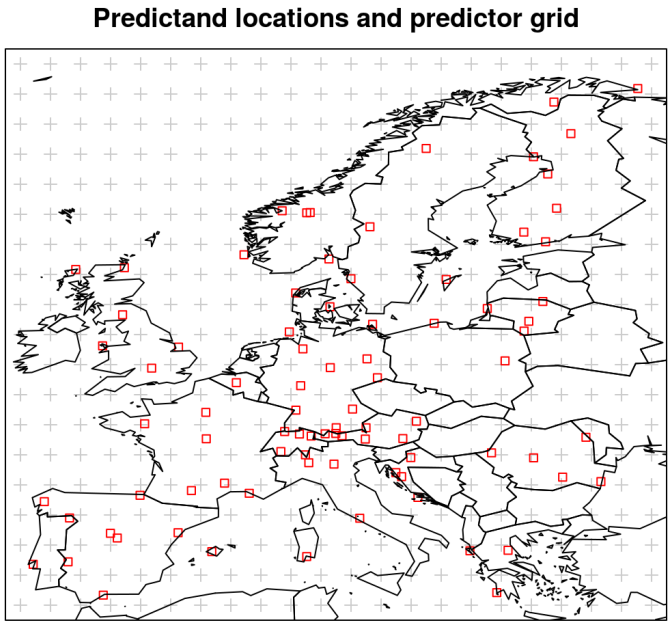


Figure 2: European domain showing the regular grid of the ERA-Interim 2-deg predictors (grey crosses) and the predictand locations corresponding to the ECA-VALUE-86 station dataset (red squares).

```
sp.layout = list(list(sp::SpatialPoints(coords.x),
                      first = FALSE,
                      col = "grey80", pch = 3),
                 list(sp::SpatialPoints(getCoordinates(y)),
                      first = FALSE,
                      col = "red", pch = 22)
),
colorkey = FALSE)
```

Each VALUE station has been assigned a unique PRUDENCE region, corresponding to the subregion for SD model training. A new map can

be prepared displaying the correspondence of each station by colors, as presented in the paper. The colors are similar to those presented in the VALUE Intercomparison synthesis paper by Gutiérrez *et al.* 2019:

```
data("PRUDENCEregions", package = "visualizeR")
areas <- PRUDENCEregions
refcoords <- get2DmatCoordinates(x.eur)
grid_clim <- climatology(subsetDimension(x.eur, dimension = "var", indices = 1))
ind <- sapply(1:length(PRUDENCEregions), FUN = function(z) {
  which(y$Metadata$PRUDENCEregion == names(PRUDENCEregions)[z])
})
# Color palette for the regions
reg.colors <- c("blue", "gold", "green", "cyan", "navyblue",
                "darkgreen", "red", "violet")
# Point layer displaying stations by colors
stations <- lapply(1:length(PRUDENCEregions), function(i) {
  list(sp::SpatialPoints(getCoordinates(y)[ind[[i]],]),
       first = FALSE, col = reg.colors[i], pch = 15)
})
# Vector layer delimiting subregions, by colors
subregions <- lapply(1:length(PRUDENCEregions), function(i) {
  list(PRUDENCEregions[i], col = reg.colors[i], lwd = 1.5)
})
sp.layout <- c(subregions, stations)
# Other graphical parameters passed to spatialPlot:
sp.layout[[length(sp.layout) + 1]] <- list('sp.text',
```

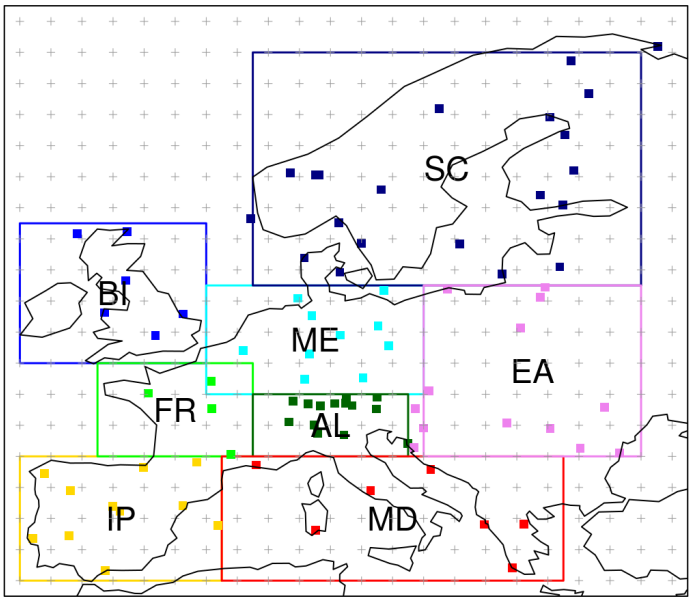


Figure 3: Same as the previous map, but showing the PRUDENCE region correspondence of each VALUE station.

```
                                        sp::coordinates(PRUDENCEregions),
                                        txt = names(PRUDENCEregions),
                                        cex = 1.5)
sp.layout[[length(sp.layout) + 1]] <- list(sp::SpatialPoints(refcoords),
                                        first = FALSE, col = "grey60",
                                        pch = 3, cex = .5, lwd = .5)
```

After defining all the tuning parameters for the map, `spatialPlot` is called, that generates the paper Fig. 2:

```
spatialPlot(grid_clim,at = seq(-2, 2, 0.1),
            backdrop.theme = "coastline",
            sp.layout = sp.layout, colorkey = FALSE)
```

Finally, as in the previous experiment, the folds used in the VALUE Project are defined for cross-validation:

```
folds <- list(1979:1984, 1985:1990, 1991:1996, 1997:2002, 2003:2008)
```

3.2 Spatial Methods M1 and M6 (VALUE methods GLM-DET and ANALOG)

This code is very similar to the code displayed in the previous sections 2.3.1 and 2.3.6, as the same predictor configuration is used:

```
config.M1.M6 <- list(which.combine = vars,
                     v.exp = .95,
                     rot = FALSE)
```

However, now the cross-validation is undertaken iteratively for each PRUDENCE region sepparately. Therefore, a `for` loop is introduced that iteratively subsets predictor and predictand sets across PRUDENCE regions.

```
n <- names(PRUDENCEregions)
n_regions <- length(n)
areas <- PRUDENCEregions
M1cv <- M6cv <- list()
for (i in 1:n_regions) {
  y1reg <- subsetDimension(y, dimension = "loc", indices = ind[[i]])
  x1reg <- subsetGrid(x.eur, lonLim = areas[n[i]]@bbox[1,],
                      latLim = areas[n[i]]@bbox[2,])
  # M6
  M6cv[[i]] <- downscaleCV(x1reg, y1reg,
```

```
                          folds = folds,
                          scaleGrid.args = list(type = "standardize"),
                          method = "analogs", n.analogs = 1,
                          prepareData.args = list(spatial.predictors = config.M1.M6)
  # M1
  y1reg_bin <- binaryGrid(y1reg, condition = "GE", threshold = 1)
  M1cv.bin <- downscaleCV(x1reg, y1reg_bin,
                          folds = folds,
                          scaleGrid.args = list(type = "standardize"),
                          method = "GLM", family = binomial(link = "logit"),
                          prepareData.args = list(spatial.predictors = config.M1.M6))
    subsetGrid(var = "bin")

  M1cv.amo <- downscaleCV(x1reg, y1reg,
                          folds = folds,
                          scaleGrid.args = list(type = "standardize"),
                          method = "GLM", family = Gamma(link = "log"),
                          condition = "GE", threshold = 1,
                          prepareData.args = list(spatial.predictors = config.M1.M6))
  M1cv[[i]] <- gridArithmetics(M1cv.bin, M1cv.amo, operator = "*")
}
```

The resulting model outputs, trained for the different subregions after spatial subsetting, are binded again into the same data structure:

```
M6cv <- bindGrid(M6cv, dimension = "loc") %>% redim(drop = TRUE)
M1cv <- bindGrid(M1cv, dimension = "loc") %>% redim(drop = TRUE)
```

Also, after slicing into different subregions, it is necessary to reorder again the predictions to match the original order of the predictand. The function `matchStations` is used to this aim:

```
M1cv2 <- matchStations(M1cv, y)
M6cv2 <- matchStations(M6cv, y)
```

3.3 Local Methods M1-L and M6-L

The predictor sets are now constructed according to the configurations M1-L and M6-L described in Table 1. Note that the code in this section is much simpler than in the previous predictor configurations, as no iteration over subregions is required in this case.

First, the specific parameter lists are defined:

```
# M1-L parameter
config.M1L <- list(local.predictors = list(n = 1, vars = vars),
                   spatial.predictors = list(v.exp = .95,
                                             which.combine = vars))
# M6-L parameters
config.M6L <- list(local.predictors = list(n = 25, vars = vars))
```

NOTE: *The following calls the* `downcaleCV` *are computationally intensive due to the relative large size of the experiment. These have been run on a 15,6GiB memory computer with a processor i7-6700 CPU@3.40GHz* $\times$ *8. Less than this may be not sufficient to run the cross-validation.*

This is the predictor configuration for GLM method M1-L:

```
std.args <- list(type = "standardize")
M1Lcv.bin <- downscaleCV(x.eur, y_bin,
                         folds = folds,
                         scaleGrid.args = std.args,
                         method = "GLM", family = binomial(link = "logit"),
                         prepareData.args = config.M1L) %>% subsetGrid(var = "bin")

M1Lcv.amo <- downscaleCV(x.eur, y,
                         folds = folds,
                         scaleGrid.args = std.args,
                         method = "GLM", family = Gamma(link = "log"),
                         condition = "GE", threshold = 1,
                         prepareData.args = config.M1L)

M1Lcv <- gridArithmetics(M1Lcv.bin, M1Lcv.amo, operator = "*")
```

And this is the cross validation for the analogs configuration M6-L:

```
M6Lcv <- downscaleCV(x.eur, y,
                     folds = folds,
                     scaleGrid.args = std.args,
                     method = "analogs",
                     n.analogs = 1,
                     prepareData.args = config.M6L)
```

3.4 Validation of the pan-European experiment

As done previously in Section 2.4, the validation is undertaken using the `valueMeasure` wrapper to the VALUE set of validation indices and measures. The root mean square error (RMSE) and correlation between the observed and predicted daily time series of precipitation are considered. In addition, we compare the variance of the observations and the predictions by calculating their respective variance indices, and then using the ratio predicted/observed as measure.

This is the list of objects containing the cross-validation results for the methods of experiment 2:

```
(methods <- ls(pattern = "^M[1|6].*cv$"))
```

```
## [1] "M1cv"  "M1Lcv" "M6cv"  "M6Lcv"
```

Note that the output of the RMSE validation is scaled by a factor of 0.1 in order to make the magnitude comparable to the other measures, so they can be displayed in the same plot.

```
# RMSE (*0.1)
rmse.list <- lapply(methods, function(m) {
    valueMeasure(y = y, x = get(m),
                 measure.code = "ts.RMSE")$Measure %>% gridArithmetics(0.1)
})
names(rmse.list) <- methods
# Correlation (Spearman's)
corr.list <- lapply(methods, function(m) {
```

```
    valueMeasure(y = y, x = get(m), measure.code = "ts.rs")$Measure
})
names(corr.list) <- methods
# Variance ratio
var.list <- lapply(methods, function(m) {
    valueMeasure(y = y, x = get(m), index.code = "Var",
                 measure.code = "ratio")$Measure
})
names(var.list) <- methods
```

Some tuning parameters are indicated in order to obtain an adequate plot display:

```
measure.names <- c("VarianceRatio", "Correlation", "RMSE")
groups <- rep(measure.names, each = length(methods))
arg.list <- c(var.list, corr.list, rmse.list)
arg.list[["group.index"]] <- groups
arg.list[["rev.colors"]] <- TRUE
arg.list[["bwplot.custom"]] <- list(ylim = c(0.1, 1.3))
```

Finally, `violinPlot` from package `visualizeR` is used to produce the paper figure:

```
do.call("violinPlot", arg.list)
```

The validation results indicate that the local predictor counterparts of the original VALUE methods M1 and M6 are competitive (the reach very similar or slightly better performance in all cases). Hence, the M1-L and M6-L method configurations will be used in Sec. **??** to produce the future

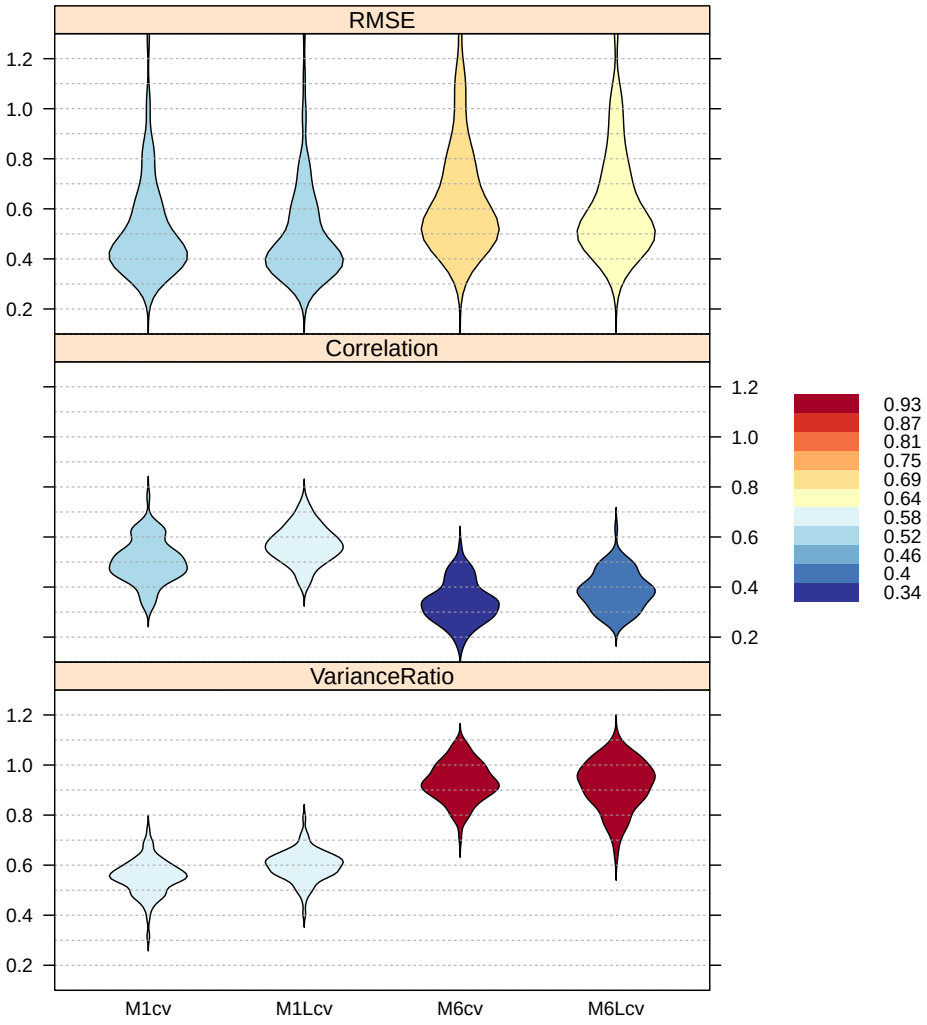


Figure 4: Cross-validation results obtained by the 4 methods tested (M1, M1-L, M7, and M7-L) in the pan-European experiment (n=86 stations), according to three selected validation measures (Spearman correlation (Corr), RMSE and variance ratio). The colour bar indicates the mean value of each measure. A factor of 0.1 has been applied to RMSE for visual comparability of results.

precipitation projections for Europe, provided their more straightforward application as they do not need to be applied independently for each subregion.

4 Delta change downscaled projections

In this section, the calibrated SD models are used to downscale GCM future climate projections from the CMIP5 EC-EARTH model (r12-i1-p1). `downscalePredict` is the workhorse for downscaling once the SD model has been calibrated with `downscaleTrain`.

4.1 Final SD model calibration

After cross-validation, the results suggest that the use of local predictors yields similar results than the original VALUE configurations M1 and M6, but providing a more straightforward implementation that does not require iteration over subregions. For this reason, the final models considering the local GLM (M1-L) and local analog (M6-L) implementations are trained for their application to climate change projections.

Prior to model training, the predictor set is standardized:

```
x_scale <- scaleGrid(x.eur, type = "standardize") #
```

The final configuration of predictors for M1-L (stored in the `config.M1L` list) and M6-L methods (`config.M6L`) is directly passed to the function `prepareData`, whose output contains all the information required to undertake model training via the `downscaleTrain` function:

```
# Predictor configuration
spatialList <- list(v.exp = .95, which.combine = vars)
# spatialList <- NULL
localList.M1L <- list(n = 1, vars = vars)
localList.M6L <- list(n = 25, vars = vars)
# Prepare predictors for M1-L
xy.M1La <- prepareData(x_scale, y_bin,
                       spatial.predictors = spatialList,
                       local.predictors = localList.M1L)
xy.M1Lb <- prepareData(x_scale, y,
                       spatial.predictors = spatialList,
                       local.predictors = localList.M1L)
# Prepare predictors for M6-L
xy.M6L <- prepareData(x_scale, y, local.predictors = localList.M6L)
```

Once the prec citors are adequately configured, the final models are calibrated with `downscaleTrain`:

```
# M1-L - occurrence
model.M1La <- downscaleTrain(xy.M1La, method = "GLM",
                             family = binomial(link = "logit"))


# M1-L - amount
model.M1Lb <- downscaleTrain(xy.M1Lb, method = "GLM",
                             family = Gamma(link = "log"),
                             condition = "GE", threshold = 1)
# M6-L - analogs
```

```
model.M6L <- downscaleTrain(xy.M6L, method = "analogs",n.analogs = 1)
```

4.2 Model prediction

Prior to model prediction, the GCM datasets required are obtained. As previously done with ERA-Interim, the EC-EARTH simulations are obtained from the `climate4R` UDG, considering the same set of variables already used for training the models. Again, these data are recursively loaded and stored in a *multigrid* as shown in Section with the ERA-Interim predictors.

First, the historical scenario is loaded, which has the UDG identifier `CMIP5_EC-EARTH_r12i1p1_historical`:

```
xh <- lapply(vars, function(x) {
  loadGridData(dataset = "CMIP5_EC-EARTH_r12i1p1_historical",
               var = x,
               lonLim = c(-10,32),
               latLim = c(36,72),
               years = 1979:2005) %>% interpGrid(new.coordinates = getGrid(x.eur))
}) %>% makeMultiGrid()
```

We repeat the process considering the UDG identifier for the RCP8.5 dataset:

```
xf <- grid.list <- lapply(vars, function(x) {
  loadGridData(dataset = "CMIP5_EC-EARTH_r12i1p1_rcp85",
               var = x,
               lonLim = c(-10,32),
```

```
                latLim = c(36,72),
                years = 2071:2100) %>% interpGrid(new.coordinates = getGrid(x.eur))
}) %>% makeMultiGrid()
```

The data for prediction (both the historical scenario `xh` and the RCP8.5 `xf`) need to be rescaled, according to the mean and variance of the predictor set used for model calibration:

```
xf <- scaleGrid(xf, base = xh, ref = x.eur,
                type = "center", spatial.frame = "gridbox",
                time.frame = "monthly") %>% scaleGrid(base = x.eur,
                                                      type = "standardize")


xh <- scaleGrid(xh, base = xh, ref = x.eur,
                type = "center", spatial.frame = "gridbox",
                time.frame = "monthly") %>% scaleGrid(base = x.eur,
                                                      type = "standardize")
```

Downscaling predictions are generated with the `downscalePredict` function. As for the model calibration step, data need to be pre-processed. This is done with the function `prepareNewData`, that receives as input the output of `prepareData` in order to undertake all the necessary checks for dimensional consistency and eventual PC related transformations:

```
# Historical dataset
h.M1La <- prepareNewData(xh, xy.M1La)
h.M1Lb <- prepareNewData(xh, xy.M1Lb)
h.M6L <- prepareNewData(xh, xy.M6L)
```

```
# RCP8.5 dataset
f.M1La <- prepareNewData(xf, xy.M1La)
f.M1Lb <- prepareNewData(xf, xy.M1Lb)
f.M6L <- prepareNewData(xf, xy.M6L)
```

Once the data prediction data is prepared, the `downscalePredict` function undertakes prediction. It receives two inputs: i) `newdata` (i.e., the data given for prediction) and ii) `model`, containing the calibrated model:

```
hist.M1La <- downscalePredict(newdata = h.M1La, model = model.M1La)
```

As previously seen, in the logistic regression model the raw prediction is a probability. In order to convert it to binary the function `binaryGrid` of `transformeR` is used. To this aim, the probability threshold for precipitation is given by the observed climatology (argument `ref.obs`) and applied to the probabilistic prediction (argument `ref.pred`):

```
hist.M1La.bin <- binaryGrid(hist.M1La,
                            ref.obs = y_bin,
                            ref.pred = model.M1La$pred)
```

The same process is next used for the remaining models and for RCP8.5 data. For GLM:

```
# GLM
hist.M1Lb <- downscalePredict(h.M1Lb, model.M1Lb)
hist.M1L <- gridArithmetics(hist.M1La.bin, hist.M1Lb, operator = "*")
futu.M1La <- downscalePredict(f.M1La, model.M1La)
futu.M1La.bin <- binaryGrid(futu.M1La,
```

```
                                ref.obs = y_bin,
                                ref.pred = model.M1La$pred)
futu.M1Lb <- downscalePredict(f.M1Lb, model.M1Lb)
futu.M1L <- gridArithmetics(futu.M1La.bin, futu.M1Lb,operator = "*")
```

And for the analogs method:

```
# ANALOG
hist.M6L <- downscalePredict(h.M6L,model.M6L)
futu.M6L <- downscalePredict(f.M6L,model.M6L)
```

4.3 Future delta maps

The downscaled projected anomalies are next calculated as the difference netween the model predictions for RCP8.5 (2071-2100) minus the predictions for the historical scenario (1979-2008). The function `climatology` is used to compute the climatological mean (mm/day) for each period, and then the function `gridArithmetics` is used to calculate the arithmetic difference between both.

We project the change in the frequency of precipitation (i.e., R01) and in the mean wet days (SDII) computed with the function `valueMeasure` of the package `climate4R.value`. This is done for the two configurations described: M1L and M6L.

```
dR01.M1L <- valueMeasure(hist.M1L,futu.M1L,measure.code = "biasRel",
                         index.code = "R01")$Measure
dSDII.M1L <- valueMeasure(hist.M1L,futu.M1L,measure.code = "biasRel",
                          index.code = "SDII")$Measure
```

```
dR01.M6L <- valueMeasure(hist.M6L,futu.M6L,measure.code = "biasRel",
                         index.code = "R01")$Measure
dSDII.M6L <- valueMeasure(hist.M6L,futu.M6L,measure.code = "biasRel",
                          index.code = "SDII")$Measure
```

We can compare the computed deltas for the M1L and M6L configurations with those done in VALUE: M1 and M6. Therefore, we repeat the training and predictions steps for the latter models and loop over the 8 VALUE subregions.

```
hist.M1 <- futu.M1 <- hist.M6 <- futu.M6 <- list()
for (i in 1:n_regions) {
  y1reg <- subsetDimension(y, dimension = "loc", indices = ind[[i]])
  y1reg_bin <- binaryGrid(y1reg,condition = "GE", threshold = 1)
  x1reg <- subsetGrid(x.eur, lonLim = areas[n[i]]@bbox[1,],
                      latLim = areas[n[i]]@bbox[2,])
  x_scale1 <- scaleGrid(x1reg, type = "standardize")
  xy.M1a <- prepareData(x_scale1, y1reg_bin,
                        spatial.predictors = config.M1.M6)
  xy.M1b <- xy.M6 <- prepareData(x_scale1, y1reg,
                        spatial.predictors = config.M1.M6)

  model.M1a <- downscaleTrain(xy.M1a, method = "GLM",
                              family = binomial(link = "logit"))
  model.M1b <- downscaleTrain(xy.M1b, method = "GLM",
                              family = Gamma(link = "log"),
                               condition = "GE", threshold = 1)
```

```
model.M6 <- downscaleTrain(xy.M6, method = "analogs",n.analogs = 1)


xh_scale1 <- subsetGrid(xh, lonLim = areas[n[i]]@bbox[1,],
                        latLim = areas[n[i]]@bbox[2,])
xf_scale1 <- subsetGrid(xf, lonLim = areas[n[i]]@bbox[1,],
                        latLim = areas[n[i]]@bbox[2,])
h.M1a <- prepareNewData(xh_scale1, xy.M1a)
h.M1b <- prepareNewData(xh_scale1, xy.M1b)
h.M6 <- prepareNewData(xh_scale1, xy.M6)
f.M1a <- prepareNewData(xf_scale1, xy.M1a)
f.M1b <- prepareNewData(xf_scale1, xy.M1b)
f.M6 <- prepareNewData(xf_scale1, xy.M6)


hist.M1a <- downscalePredict(newdata = h.M1a, model = model.M1a)
hist.M1a.bin <- binaryGrid(hist.M1a,
                           ref.obs = y1reg_bin,
                           ref.pred = model.M1a$pred)
hist.M1b <- downscalePredict(h.M1b, model.M1b)
hist.M1[[i]] <- gridArithmetics(hist.M1a.bin, hist.M1b, operator = "*")
futu.M1a <- downscalePredict(f.M1a, model.M1a)
futu.M1a.bin <- binaryGrid(futu.M1a,
                           ref.obs = y1reg_bin,
                           ref.pred = model.M1a$pred)
futu.M1b <- downscalePredict(f.M1b, model.M1b)
futu.M1[[i]] <- gridArithmetics(futu.M1a.bin, futu.M1b,operator = "*")
```

```
  hist.M6[[i]] <- downscalePredict(h.M6,model.M6)
  futu.M6[[i]] <- downscalePredict(f.M6,model.M6)
}

hist.M1 <- bindGrid(hist.M1, dimension = "loc") %>% redim(drop = TRUE) %>%
  matchStations(y)
hist.M6 <- bindGrid(hist.M6, dimension = "loc") %>% redim(drop = TRUE) %>%
  matchStations(y)
futu.M1 <- bindGrid(futu.M1, dimension = "loc") %>% redim(drop = TRUE) %>%
  matchStations(y)
futu.M6 <- bindGrid(futu.M6, dimension = "loc") %>% redim(drop = TRUE) %>%
  matchStations(y)

save(hist.M1,hist.M6,futu.M1,futu.M6,
     file = "./Data/deltasVALUE.rda")
```

Finally we compute the deltas in the R01 and SDII indices for the M1 and M6 configurations.

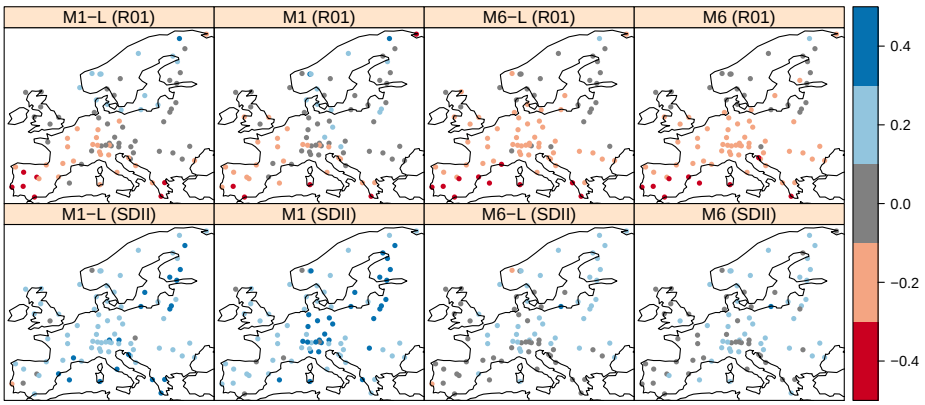
```
dR01.M1 <- valueMeasure(hist.M1,futu.M1,measure.code = "biasRel",
                        index.code = "R01")$Measure
dSDII.M1 <- valueMeasure(hist.M1,futu.M1,measure.code = "biasRel",
                         index.code = "SDII")$Measure
dR01.M6 <- valueMeasure(hist.M6,futu.M6,measure.code = "biasRel",
                        index.code = "R01")$Measure
dSDII.M6 <- valueMeasure(hist.M6,futu.M6,measure.code = "biasRel",
                         index.code = "SDII")$Measure
```

The GLM and Analog projected anomalies for M1L, M6L, M1 and M6 configurations are grouped in a multigrid in order to jointly display them in the map.

```
deltas <- makeMultiGrid(dR01.M1L,dR01.M1,dR01.M6L,dR01.M6,
                        dSDII.M1L,dSDII.M1,dSDII.M6L,dSDII.M6)
```

The figure is produced with `spatialPlot`. Here, we use the color palettes provided by the R package `RColorBrewer`, and several tuning parameters are passed to spatialPlot to produce the final paper figure:

```
cb <- RColorBrewer::brewer.pal(n = 5, "RdBu")
cb[3] <- "#808080"
spatialPlot(deltas, backdrop.theme = "coastline",
            aspect = "iso",
            col.regions = cb,
            cuts = seq(-0.5, 0.5, 0.2),
            set.min = -0.5, set.max = 0.5,
            colorkey = TRUE,
            layout = c(4,2),
            as.table = TRUE,
            cex = 0.6,
            strip = lattice::strip.custom(factor.levels = c("M1-L (R01)","M1 (R01)",
                                                            "M6-L (R01)","M6 (R01)",
                                                            "M1-L (SDII)","M1 (SDII)"
                                                            "M6-L (SDII)","M6 (SDII)"
)
```



We can check the assumption of perfect-prognosis approach (i.e., the predictor/predictand link infered in "perfect" conditions, using reanalysis data, is applicable to downscale GCMs), by comparing the similarity in the probability density function (pdf) of every variable between the ERA-Interim's and EC-Earth predictors in the historical period (1986-2005). We choose a representative gripoint of every VALUE domain and plot the pdf using the densityPlot function of 'visualizeR'.

5 References

- Bache, S.M. and Wickham, H., 2014. `magrittr`: A Forward-Pipe Operator for R. R package version 1.5. https://CRAN.R-project.org/package=magrittr
- Gutiérrez, J.M., Maraun, D., Widmann, M., Huth, R., Hertig, E., Benestad, R., Roessler, O., Wibig, J., Wilcke, R., Kotlarski, S., San Martín, D., Herrera, S., Bedia, J., Casanueva, A., Manzanas, R., Iturbide, M., Vrac, M., Dubrovsky, M., Ribalaygua, J., Pór-

toles, J., Räty, O., Räisänen, J., Hingray, B., Raynaud, D., Casado, M.J., Ramos, P., Zerenner, T., Turco, M., Bosshard, T., Štěpánek, P., Bartholy, J., Pongracz, R., Keller, D.E., Fischer, A.M., Cardoso, R.M., Soares, P.M.M., Czernecki, B., Pagé, C., 2019. An intercomparison of a large ensemble of statistical downscaling methods over Europe: Results from the VALUE perfect predictor cross-validation experiment. International Journal of Climatology. https://doi.org/10.1002/joc.5462

- Iturbide, M., Bedia, J., Herrera, S., Baño-Medina, J., Fernández, J., Frías, M.D., Manzanas, R., San-Martín, D., Cimadevilla, E., Cofiño, A.S., Gutiérrez, J.M., 2019. The R-based climate4R open framework for reproducible climate data access and post-processing. Environmental Modelling & Software 111, 42–54. https://doi.org/10.1016/j.envsoft.2018.09.009
- San-Martín, D., Manzanas, R., Brands, S., Herrera, S., Gutiérrez, J.M., 2016. Reassessing Model Uncertainty for Regional Projections of Precipitation with an Ensemble of Statistical Downscaling Methods. J. Climate 30, 203–223. https://doi.org/10.1175/JCLI-D-16-0366.1
- Wickham, H., Hester, J. and Chang, W., 2018. `devtools`: Tools to Make Developing R Packages Easier. R package version 2.0.1. https://CRAN.R-project.org/package=devtools

6 Session information

```
sessionInfo(package = c("loadeR", "downscaleR", "transformeR",
                        "visualizeR", "VALUE", "climate4R.value")) %>% print()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/libblas/libblas.so.3.6.0
## LAPACK: /usr/lib/lapack/liblapack.so.3.6.0
##
## 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:
## character(0)
##
## other attached packages:
## [1] loadeR_1.4.15         downscaleR_3.1.0      transformeR_1.6.1
```

```
## [4] visualizeR_1.5.1      VALUE_2.0.0           climate4R.value_0.0.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.2              lattice_0.20-38
##  [3] zoo_1.8-6               glmnet_2.0-18
##  [5] digest_0.6.21           foreach_1.4.4
##  [7] SpecsVerification_0.5-2 CircStats_0.2-6
##  [9] signal_0.7-6            evaluate_0.14
## [11] spam_2.2-2              utils_3.6.0
## [13] rlang_0.4.0             data.table_1.12.2
## [15] raster_2.9-23           Matrix_1.2-17
## [17] rmarkdown_1.15.2        stringr_1.4.0
## [19] RcppEigen_0.3.3.5.0     RCurl_1.95-4.12
## [21] munsell_0.5.0           proxy_0.4-23
## [23] compiler_3.6.0          easyVerification_0.4.4
## [25] xfun_0.9                stats_3.6.0
## [27] gridGraphics_0.4-1      htmltools_0.4.0
## [29] tcltk_3.6.0             evd_2.3-3
## [31] gridExtra_2.3           dtw_1.20-1
## [33] codetools_0.2-15        grDevices_3.6.0
## [35] mapplots_1.5.1          deepnet_0.2
## [37] MASS_7.3-51.1           bitops_1.0-6
## [39] grid_3.6.0              gtable_0.3.0
## [41] magrittr_1.5            datasets_3.6.0
## [43] scales_1.0.0            stringi_1.4.3
```

```
## [45] sm_2.2-5.6              pbapply_1.4-1
## [47] kohonen_3.0.8           sp_1.3-1
## [49] latticeExtra_0.6-28     akima_0.6-2
## [51] padr_0.5.0              graphics_3.6.0
## [53] vioplot_0.3.2           boot_1.3-20
## [55] base_3.6.0              loadeR.java_1.1.1
## [57] RColorBrewer_1.1-2      iterators_1.0.10
## [59] tools_3.6.0             maps_3.3.0
## [61] fields_9.8-3            abind_1.4-5
## [63] parallel_3.6.0          yaml_2.2.0
## [65] colorspace_1.4-1        verification_1.42
## [67] dotCall64_1.0-0         rJava_0.9-11
## [69] knitr_1.25              methods_3.6.0
```