Forecast verification: a worked example

In this example we obtain data from the NCEP's CFSv2 seasonal forecasting system (`dataset = "CFSv2_seasonal"`). In particular, in this example we will retrieve maximum daily surface temperature (`var = "tasmax"`) for boreal summer (JJA, `season = 6:8`) for a rectangular domain centered on the Iberian Peninsula and France (`lonLim = c(-10,15)` and `latLim = c(35,50)`), for the period 1981-2000 (`years = 1981:2000`), and considering the first 9 ensemble members (`members = 1:9`) and a lead-month 2 forecast 2 (`leadMonth = 2`).

We will illustrate the verification of these predictions data against the observational gridded datasets WATCH Forcing Dataset-ERA-Interim (WFDEI, `dataset = "WFDEI"`), also available via the ECOMS-UDG. To this aim, we will use the tools developed within the projects SPECS and EUPORIAS. In particular, we will use the verification routines available in the R package `SpecsVerification` (available on CRAN). However, instead of using them directly, we will use the user-friendly interface implemented in package `easyVerification`, via the wrapper function `veriApply` (To find out more on the functionality in the `easyVerification` package, please read the vignette with the R instruction `vignette("easyVerification")`)

Package loading/install

We first load (and install if necessary) the required libraries. `loadeR.ECOMS` and `downscaleR` are loaded for data loading and manipulation respectively:

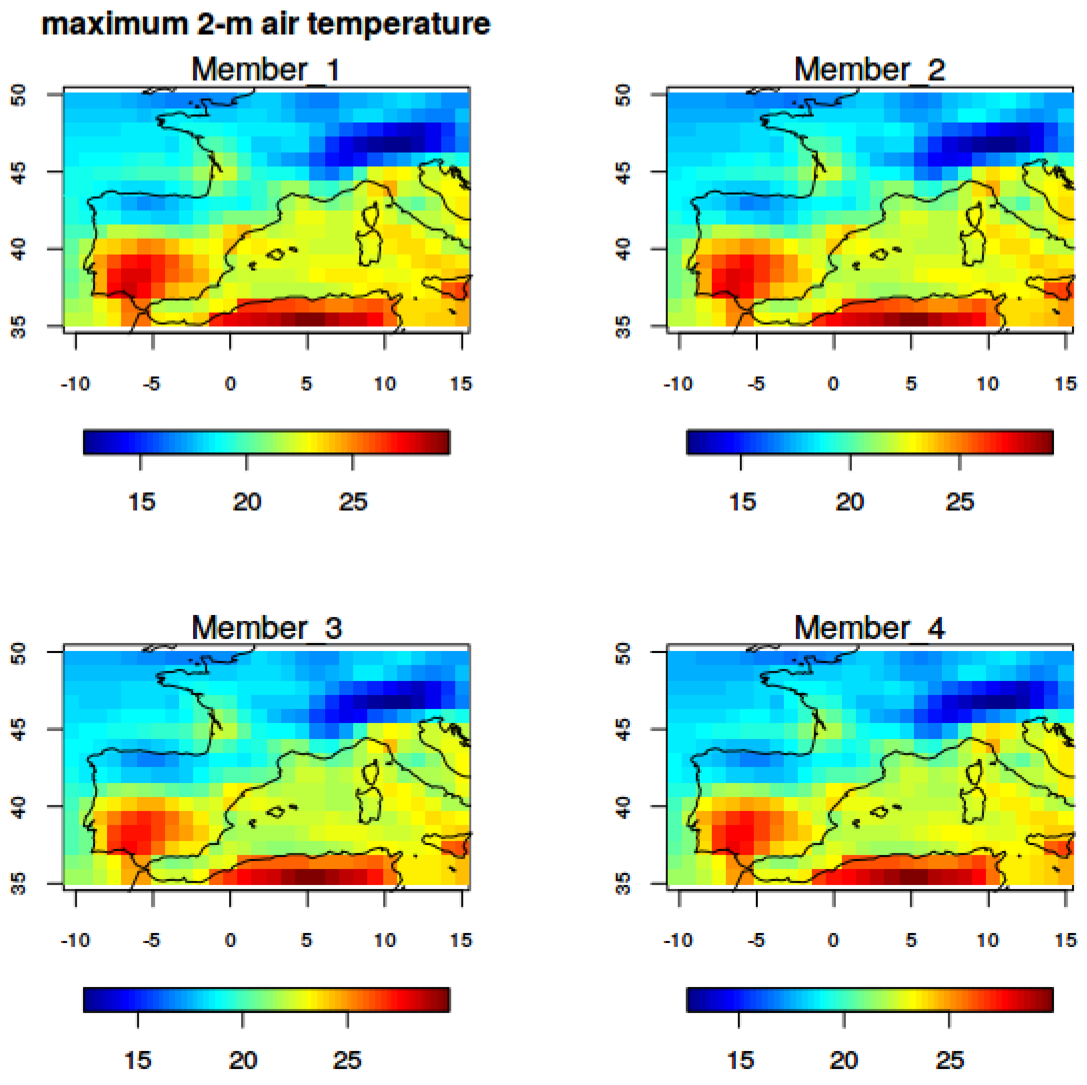
```
library(loadeR.ECOMS)
library(downscaleR)
```

The two packages related with verification are next installed if not already present:

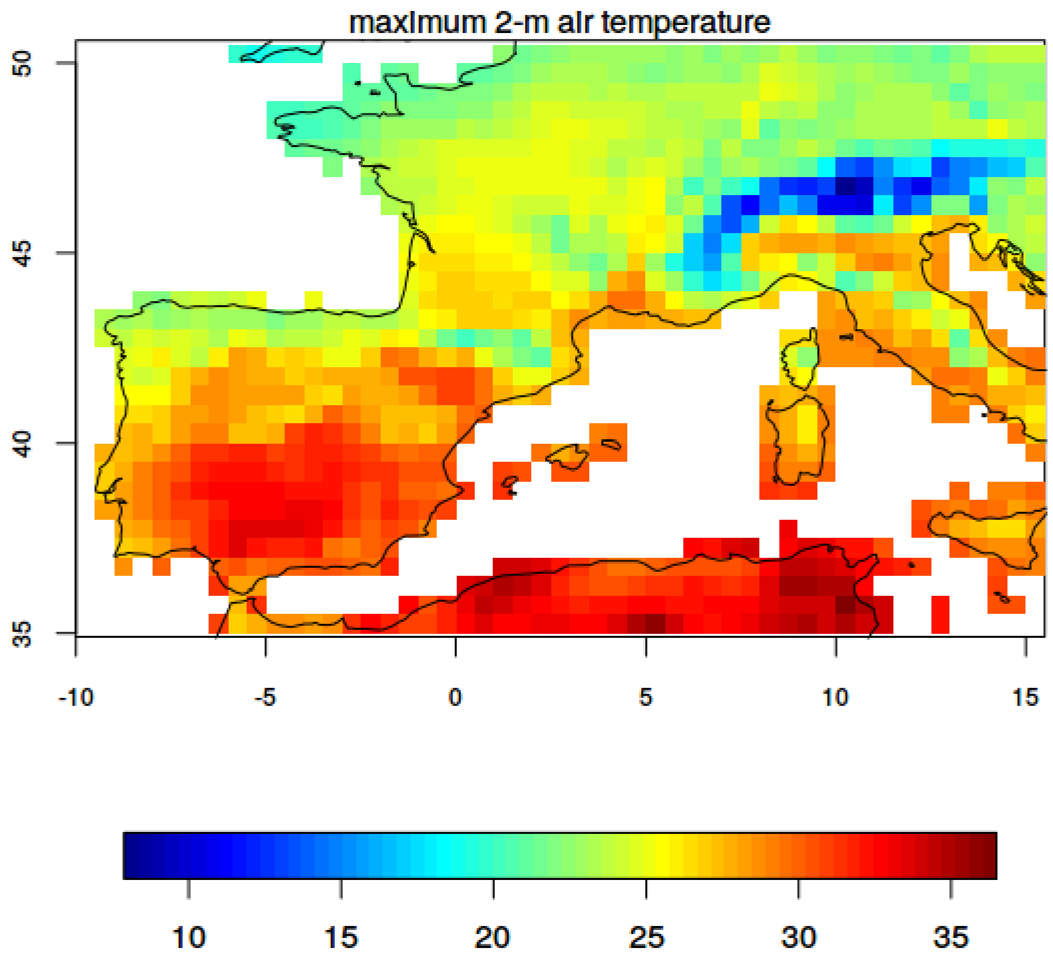
```
# Attempting the installation of SpecsVerification
if (!require("SpecsVerification")) {
     install.packages("SpecsVerification")
}
# Attempting the installation of easyVerification
if (!require("easyVerification")) {
     if (!require(devtools)) {
           install.packages("devtools")
     }
     devtools::install_github("MeteoSwiss/easyVerification", build_vignettes=TRUE)
}
```

Data loading from the ECOMS-UDG

```
tx.forecast <- loadECOMS(dataset = "CFSv2_seasonal",
                        var = "tasmax",
                        members = 1:4,
                        lonLim = c(-10 ,15),
                        latLim = c(35, 50),
                        season = 6:8,
                        years = 1991:2000,
                        leadMonth = 2)

## [2016-05-12 12:56:18] Defining homogeneization parameters for variable "tasmax"
## [2016-05-12 12:56:18] Opening dataset...
## [2016-05-12 12:56:48] The dataset was successfuly opened
## [2016-05-12 12:56:48] Defining geo-location parameters
## [2016-05-12 12:56:49] Defining initialization time parameters
## [2016-05-12 12:56:51] Retrieving data subset ...
## [2016-05-12 13:07:02] Done

plotMeanGrid(tx.forecast, multi.member = TRUE)
```



```
tx.obs <- loadECOMS(dataset = "WFDEI",
                   var = "tasmax",
                   lonLim = c(-10 ,15),
                   latLim = c(35, 50),
                   season = 6:8,
                   years = 1991:2000)
## [2016-05-12 14:03:40] Defining homogeneization parameters for variable "tasmax"
## [2016-05-12 14:03:40] Opening dataset...
## [2016-05-12 14:03:42] The dataset was successfuly opened
## [2016-05-12 14:03:42] Defining geo-location parameters
## [2016-05-12 14:03:42] Defining time selection parameters
## [2016-05-12 14:03:42] Retrieving data subset ...
## [2016-05-12 14:03:52] Done

plotMeanGrid(tx.obs)
```



Data preprocessing: interpolation and aggregation using downscaleR

Data interpolation

To be able to validate the forecasts, we first have to interpolate either the forecasts or the observations to have the data on a common grid. We interpolate the observations to the grid of the forecasts using the nearest neighbour algorithm:

```
tx.obsintp <- interpGrid(tx.obs, new.coordinates = getGrid(tx.forecast), method = "nearest")
## [2016-05-12 14:12:10] Calculating nearest neighbors...
## [2016-05-12 14:12:11] Performing nearest interpolation... may take a while
## [2016-05-12 14:12:11] Interpolating member 1 out of 4
## [2016-05-12 14:12:11] Interpolating member 2 out of 4
## [2016-05-12 14:12:11] Interpolating member 3 out of 4
## [2016-05-12 14:12:11] Interpolating member 4 out of 4
## [2016-05-12 14:12:13] Done
```

Temporal aggregation

Now that the predictions are in the same grid than the observations, we can proceed with the verification. However, before we go on, we compute seasonal averages of the forecasts and observations for validation of seasonal average daily maximum temperature. This is easily undertaken using downscaleR's function `aggregateGrid` (in Linux and MacOS it is possible to speed-up the aggregation by using the parallelization option):

```
mn.tx.forecast <- aggregateGrid(tx.forecast, aggr.y = list(FUN = "mean"), parallel = TRUE)
## Parallel computing enabled
## Number of workers: 3
## [2016-05-12 14:30:17] Performing annual aggregation in parallel...
## [2016-05-12 14:30:18] Done.

mn.tx.obsintp <- aggregateGrid(tx.obsintp, aggr.y = list(FUN = "mean"), parallel = TRUE)
##  Parallel computing enabled
##  Number of workers: 3
##  [2016-05-12 14:30:28] Performing annual aggregation in parallel...
##  [2016-05-12 14:30:30] Done.
```

Now we are ready to compute validation scores on the 3-monthly mean daily maximum temperature forecasts. We next compute several typical verification measures:

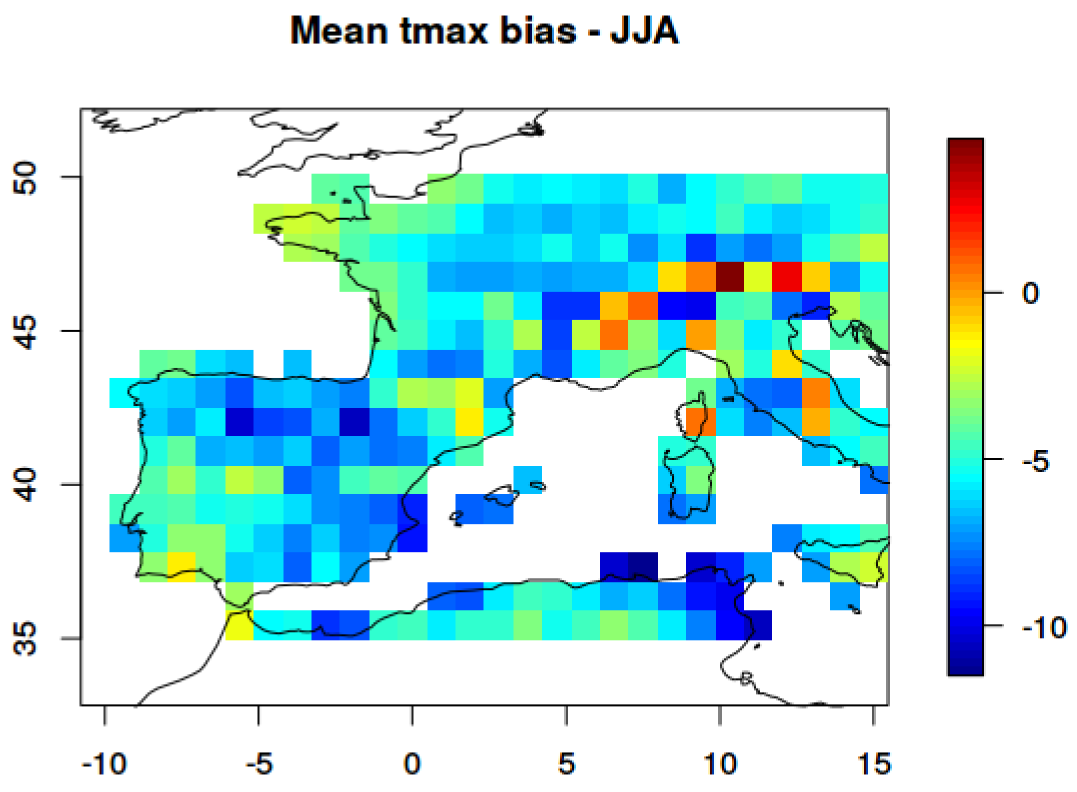
Verification measures using SpecsVerification? and easyVerification

Mean bias

```
bias <- veriApply("EnsMe",
                 fcst = mn.tx.forecast$Data,
                 obs = mn.tx.obsintp$Data,
                 ensdim = 1,
                 tdim = 2)
```

The object `bias` is a matrix with longitudes in columns and latitudes in rows. To do the plotting, we can use the coordinates of the reference grid of the CFS forecast. For convenience, we will use the function `image.plot` from package `fields` (this should be already installed because it is a dependency of `downscaleR`). In order to add a coastline map, we do a trick and call an internal function of `downscaleR` using the triple `:::` notation:

```
fields::image.plot(tx.forecast$xyCoords$x,
                  tx.forecast$xyCoords$y,
                  t(bias),
                  asp = 1, xlab = "", ylab = "", main = "Mean tmax bias - JJA")
downscaleR:::draw.world.lines()
```


Mean tmax bias - JJA
50
45
40
35
-10
-5
0
5
10
15
0
-5
-10