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:

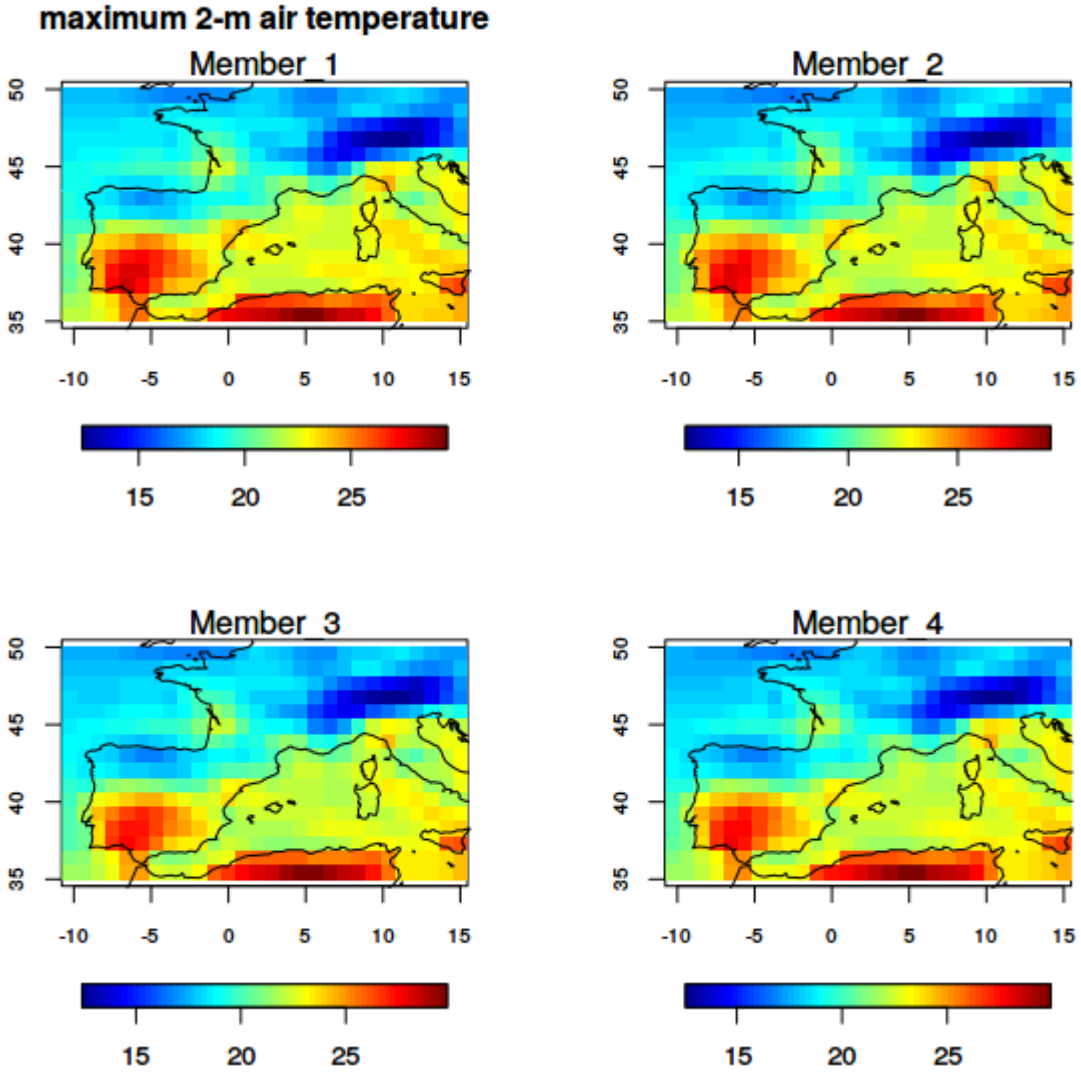
```
library(loadeR.ECOMS)
library(downscaleR)
```

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

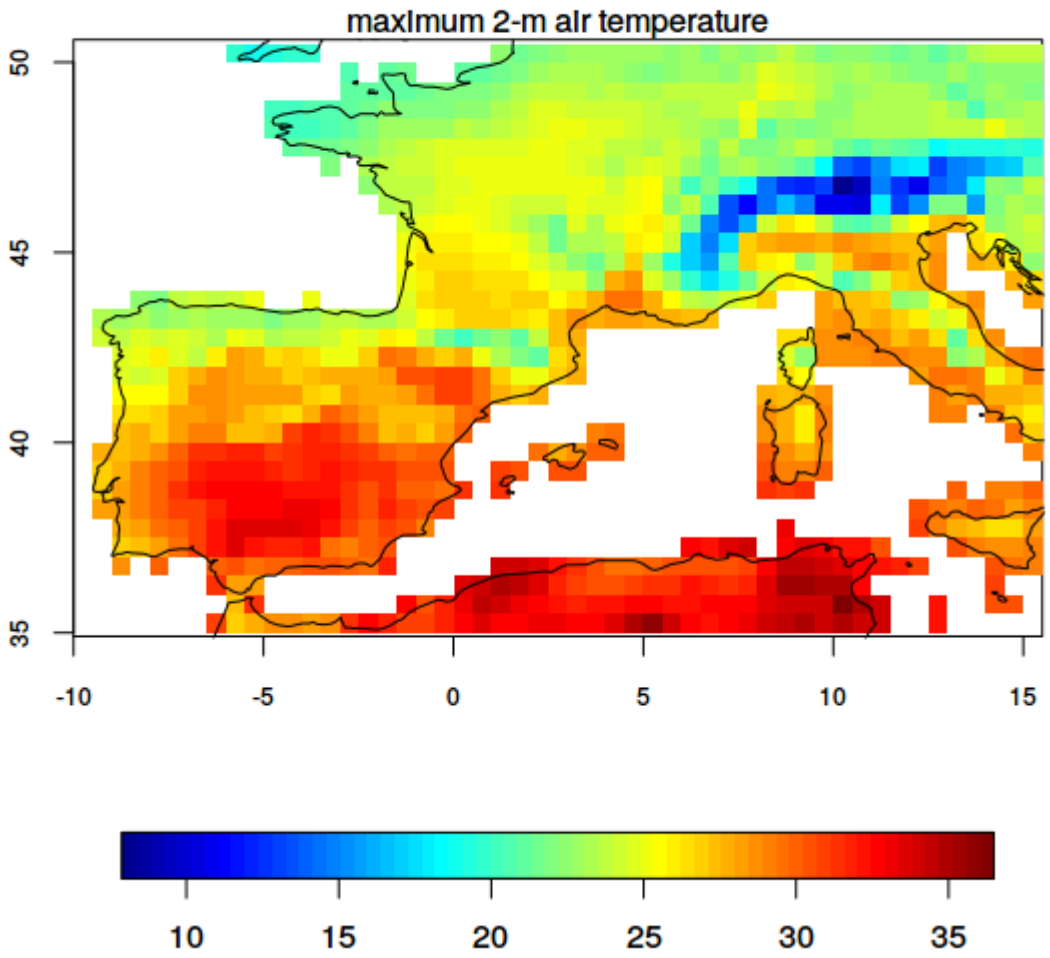
```
# Attempting the installation of SpecsVerification
if (!require("SpecsVerification")) {
     install.packages("SpecsVerification")
}
# Attempting the installation of easyVerification
if (!require("easyVerification")) {
     if (!require(devtools)) {
          install.packages("devtools")
     }
     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

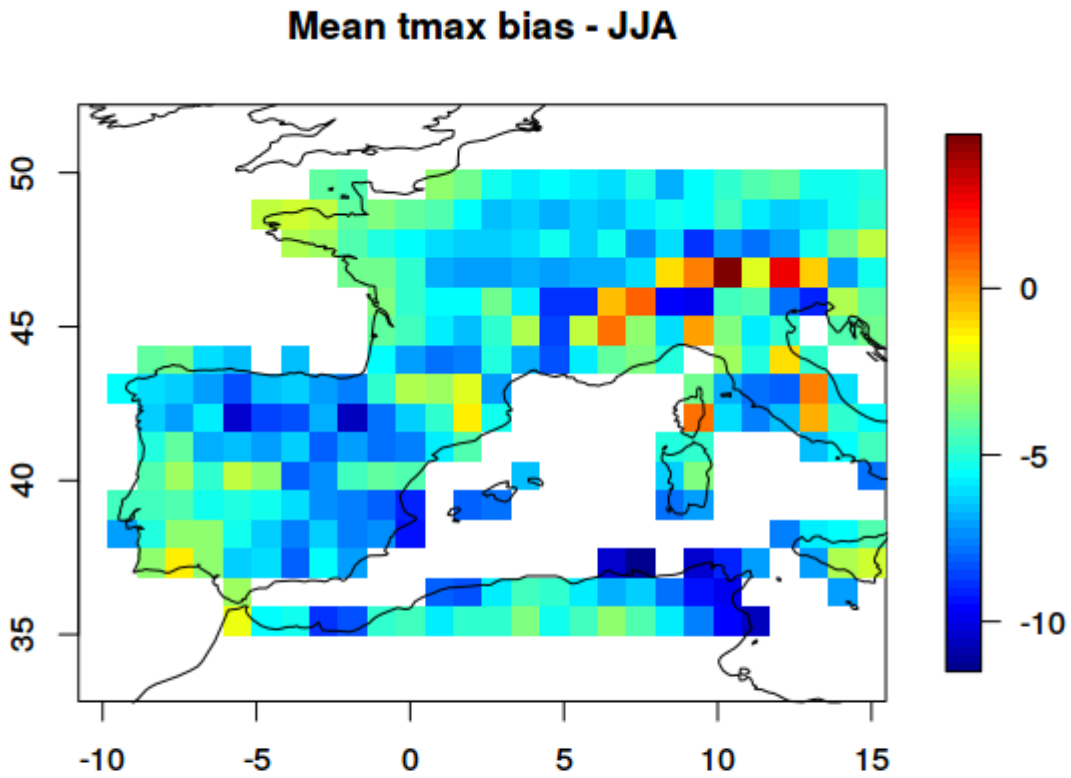
```
library(easyVerification)
```

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()
```

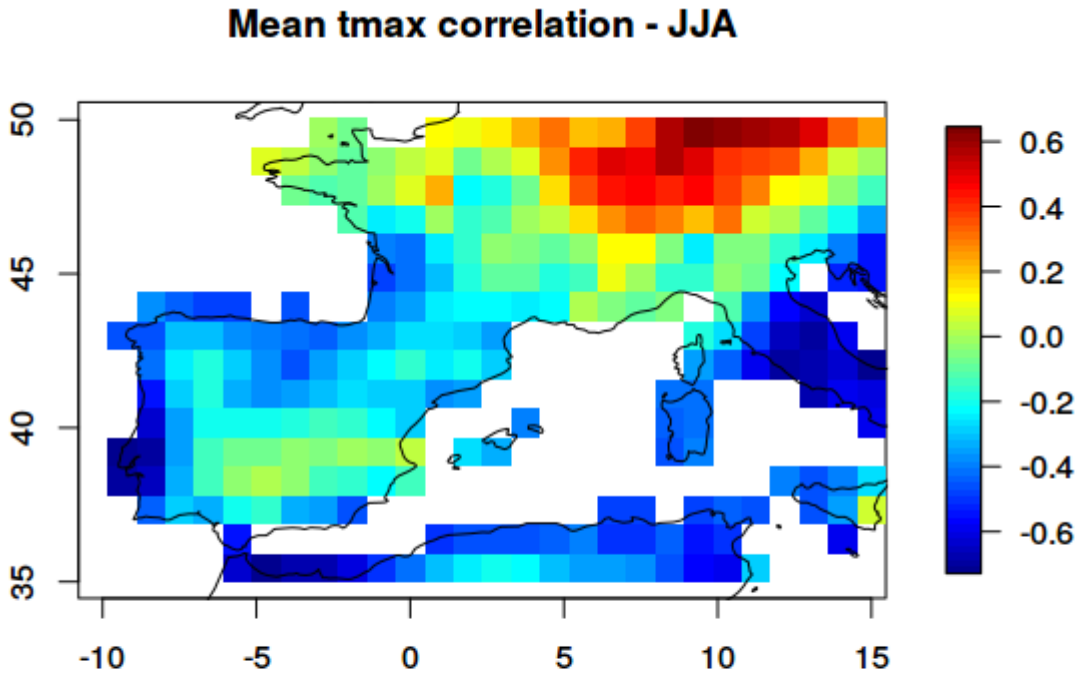


The results reveal a significant cold bias of the CFSv2 model predictions.

Correlation

We follow a similar approach to compute the ensemble mean forecast correlation against the verifying observations:

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

fields::image.plot(tx.forecast$xyCoords$x,
                   tx.forecast$xyCoords$y,
                   t(corr),
                   asp = 1, xlab = "", ylab = "",
                   main = "Mean tmax correlation - JJA")

downscaleR:::draw.world.lines()
```



We find that the ensemble mean summer forecasts for 1991-2000 correlate well with the verifying observations over the north-western sector of the analysis area, but the forecasts do not skilfully represent year-to-year variability over the Iberian Peninsula and the Mediterranean area.

Ranked probability skill score (RPSS)

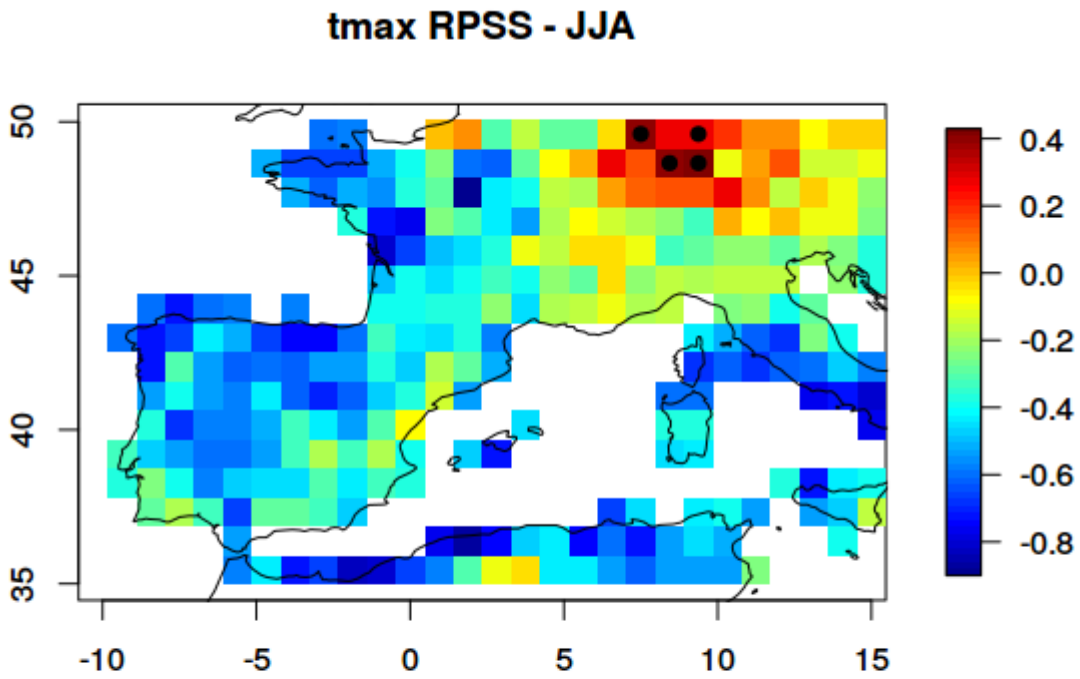
We next illustrate the ranked probability skill score (RPSS). Here we use the RPSS for tercile forecasts, that is probability forecasts for the three categories colder than average, average, and warmer than average. In order to convert observations and forecast in probabilities for the three categories, we have to add an additional argument `prob` to the `veriApply` function with the quantile boundaries for the categories chosen. In this case, to indicate that validation is performed on the terciles, we use the value `prob=c(1/3,2/3)`, as indicated next:

```
rpss <- veriApply("EnsRpss",
                  fcst = mn.tx.forecast$Data,
                  obs = mn.tx.obsintp$Data,
                  prob = c(1/3,2/3),
                  ensdim = 1, tdim = 2)
```

In this case, the output is a list consisting of two components: The first one is the RPSS lon-lat matrix, as in the previous examples. The second one, provides the standard error, useful to calculate the significance of the score at each particular grid point (at the 95% c.i. in this example):

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

# Compute significant points and collocate spatially:
sig.i <- rpss$rpss > rpss$rpss.sigma*qnorm(0.95)
lons <- rep(mn.tx.obsintp$xyCoords$x, each = length(mn.tx.obsintp$xyCoords$y))
```

```
lats <- rep(mn.tx.obsintp$xyCoords$y, length(mn.tx.obsintp$xyCoords$x))
points(lons[sig.i], lats[sig.i], pch = 19)
```



In this case, we only get significant RPSS values in four points in the northern part of the domain.

Acknowledgements

These examples have been prepared by Jonas Bhend (**Meteo Swiss**), in collaboration with the **Santander Met Group**.

Package versions and session info

```
print(sessionInfo(), locale = FALSE)

## R version 3.3.0 (2016-05-03)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.4 LTS

## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base

## other attached packages:
## [1] downscaleR_1.0-1        easyVerification_0.2.0  loadeR.ECOMS_1.0-0      loadeR_1.0-0            loadeR.java_1.1-0
## [6] rJava_0.9-8             SpecsVerification_0.4-1

## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.4       devtools_1.10.0   maps_3.1.0        MASS_7.3-44       evd_2.3-2         munsell_0.4.3     colors
##  [8] lattice_0.20-33   pbapply_1.1-3     plyr_1.8.3        fields_8.4-1      tools_3.3.0       CircStats_0.2-4   parall
```

```
## [15] grid_3.3.0        spam_1.3-0        dtw_1.18-1        digest_0.6.9      abind_1.4-3       akima_0.5-12      bitops
## [22] RCurl_1.95-4.8    memoise_1.0.0     sp_1.2-3          proxy_0.4-15      scales_0.4.0      boot_1.3-17       verifi
```