2017 Climate Services paper

J. Bedia & M. Iturbide

March 27, 2017

This is a worked example that reproduces the steps followed in Bedia et al. 2017 through the use of R-package `fireDanger`, that contains the functionalities for computing the Fire Weather Index (FWI). Packages `loadeR.ECOMS`, `downscaleR` and `easyVerification` have also been used for data loading, data manipulation, bias correction and verification.

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

In the last section of this tutorial (sec. Visualization) we also illustrate the use of different functionalities from the `visualizeR` package for seasonal forecast uncertainty visualization.

Accessing the User Data Gateway (UDG)

The User Data Gateway is the one stop shop for climate data access maintained by the Santander MetGroup. The UDG builds on the THREDDS Access Portal (UDG-TAP) which is the entry point for authentication and data access (information for registration). Authorization is organized in thematic groups, according to the data policies of different projects and international initiatives. There is also public data available that is open for everyone, for instance, the NCEP/NCAR Reanalysis 1 data, that is used in this worked example. After registration, a username and password is provided to sign in.

Loading data using the R interface

`loadeR.ECOMS` is the R-package providing transparent access to different seasonal forecast products available through the UDG (see the complete Table of available datasets and variables).

Prior to accessing the data at UDG, the credentials need to be set. This can be done within R using the function `loginUDG`:

```
loginUDG(username = "myuser", password = "mypassword")
```

Note: loading seasonal forecast data is a time-consuming step. Loading times may vary greatly depending on several factors. This step may take several hours of data download. On the contrary, loading the observations (WFDEI) is very fast, due to the much greater simplicity of the dataset.

Loading seasonal forecast data

Next we are loading the variables needed for computing the FWI with function `loadECOMS`. For the sake of brevity, we will consider a smaller spatio-temporal domain studied in the paper. In this case, we will develop the example using the NCEP's CFS-v2 hindcast instead of ECMWF System4, as the former is a public product, while System4 has restricted access, thus ensuring the maximum reachability of this tutorial. We will load a 27-year period (1983-2009; the available hindcast period for CFS-v2).

A few arguments are used to unequivocally define the seasonal forecast request (type `?load.ECOMS` for details):

```
dataset <- "CFSv2_seasonal"
season <- 5:9
leadMonth <- 0
years <- 1983:2009
latLim <- c(35, 47)
lonLim <- c(-10, 30)
members <- 1:15
```

Note that in order to reproduce the paper results (based on System 4), the argument dataset should be set to `dataset = "System4_seasonal_15"`, leaving the rest of commands and instructions unchanged. Thus, the `loadeR.ECOMS` package has been conceived as "user-transparent", in the sense that the users do not need to worry about the technical complexities regarding member definition (see e.g. how CFS-v2 members have been defined) and/or the different variable naming and units. In addition, the `aggr.d` argument allows to perform time aggregation of sub-daily variables on-the-fly, as required for daily accumulated precipitation. For instantaneous variables, the argument `time` would be set to the corresponding verification time. This ensures the maximum reproducibility of the results, that may be altered to some extent depending on the various aggregation options.

In this example, we will use aggregated data to illustrate this capability. This would be the proxy-based approach for the calculation of FWI used for the generation of the state-of-the-art future FWI projections for Europe (Bedia et al 2014).

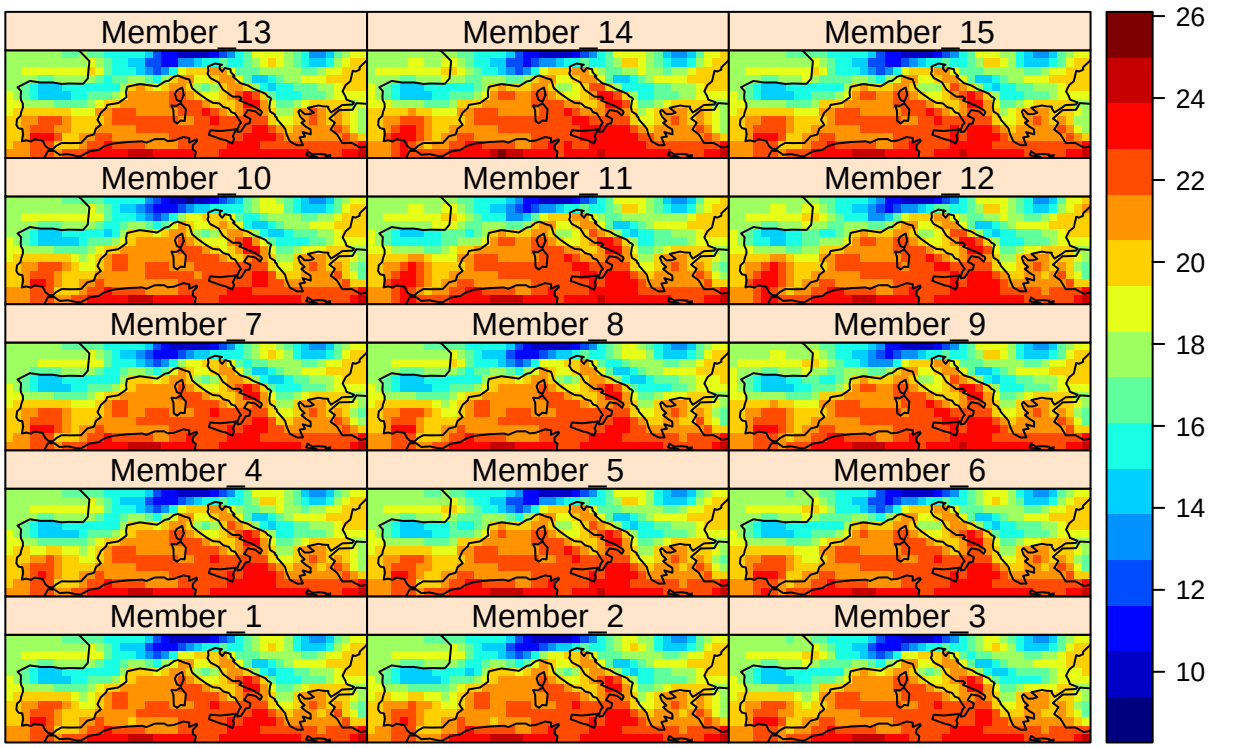
The argument `leadMonth` indicates the initialization time of the predictions. This argument and the argument `season` (in months, from 1 to 12), unequivocally define the initialization time and the corresponding verification times to be chosen. In addition, the argument `years` indicates the whole analysis period. So, in this particular case, we are loading the May initialization (`leadMonth = 0` and `season = 5:9`), since the starting day (1 May) until the end of the target season (30 September). Note that the fire season considered in the paper is June-September (JJAS), but the predictions for May are also considered in order to compute FWI a few weeks before the start of the season as a spin-up period for FWI stabilization (see Sec. 2.1 in the paper).

Finally, the arguments `lonLim` and `latLim` provide the longitudinal and latitudinal boundaries of the spatial domain, while the argument `members` indicates the number of ensemble members to be considered.

```
## Load temperature
Tm <- loadECOMS(dataset, var = "tas", members = members,
                latLim = latLim, lonLim = lonLim,
                season = season, years = years, leadMonth = leadMonth,
                time = "DD", aggr.d = "mean")
## Load relative humidity
H <- loadECOMS(dataset, var = "hurs", members = members,
               latLim = latLim, lonLim = lonLim,
               season = season, years = years, leadMonth = leadMonth,
               time = "DD", aggr.d = "min")
## Load precipitation
r <- loadECOMS(dataset, var = "tp", members = members,
               latLim = latLim,  lonLim = lonLim,
               season = season, years = years, leadMonth = leadMonth,
               time = "DD", aggr.d = "sum")
## Load wind speed
W <- loadECOMS(dataset, var = "wss", members = members,
               latLim = latLim, lonLim = lonLim,
               season = season, years = years, leadMonth = leadMonth,
               time = "DD", aggr.d = "mean")
```

The loaded grids are easily visualized with function `plotClimatology` and `climatology`. For example:

```
plotClimatology(climatology(Tm), backdrop.theme = "coastline")
```



Note that neither relative humidity (`hurs`), nor wind speed (`wss`) are original variables produced by the CFSv2 model (see the table of available variables). These variables are computed on-the-fly by the `loadeR.ECOMS` R interface, greatly facilitating data processing to the user.

Also note that the standard units of `wss` are m/s. Thus, an intermediate step is needed to convert to Km/h, as required for FWI calculation. After this operation, the four input variables are "stacked" in a single `multigrid`, which is an special object containing several variables as an extra-dimension, that ensures their spatio-temporal consistency, either for downscaling, EOF/PCA analyses or, as in this case, to construct a FWI grid with `fireDanger`. We use to this aim the multigrid constructor `makeMultiGrid`:

```
## Convert wss units from m/s to km/h
W$Data <- W$Data*3.6
## Update "units" attribute
attr(W$Variable, "units") <- "km.h-1"
## make multigrid
multigrid_hind <- makeMultiGrid(Tm, H, r, W)
```

Loading Observations

The same operation is repeated to get the multigrid of the observations, in this case we will use the Watch Forcing Dataset based on ERA-Interim, as described in the paper (dataset = `WFDEI`). Note that in this case the arguments `leadMonth` and `members`, particular for predictions, do not apply. Also, because the WFDEI variables currently available are already daily aggregated, there is no need to specify particular time aggregation options, as the default will load the maximum temporal resolution.

As `loadeR.ECOMS` performs dataset homogenization, there is no need to change the variable nomenclature. In the case of relative humidity, we directly request minimum relative humidity (`var = "hursmin"`). Similarly, the variables loaded will have exactly the same units as with the CFSv2 dataset, and thus the conversion from m/s to km/h is needed again:

```
## Define the target dataset
dataset <- "WFDEI"

## Load temperature
Tm.obs <- loadECOMS(dataset = dataset, var = "tas",
                latLim = latLim, lonLim = lonLim,
                season = season, years = years)
## Load relative minimum humidity
H.obs <- loadECOMS(dataset = dataset, var = "hursmin",
               latLim = latLim, lonLim = lonLim,
               season = season, years = years)
## Load precipitation
r.obs <- loadECOMS(dataset = dataset, var = "tp",
               latLim = latLim,  lonLim = lonLim,
               season = season, years = years)
## Load wind speed
W.obs <- loadECOMS(dataset = dataset, var = "wss",
               latLim = latLim, lonLim = lonLim,
               season = season, years = years)
```

Conversion of standard units to km/h for windspeed (and metadata update):

```
W.obs$Data <- W.obs$Data*3.6
attr(W.obs$Variable, "units") <- "km.h-1"
```

And multigrid creation:

```
multigrid_obs <- makeMultiGrid(Tm.obs, H.obs, r.obs, W.obs)
```

The following examples in this tutorial can be reproduced without executing the above lines for data loading, since objects `multigrid_obs` and `multrigrid_hind` are available for download:

```
load(url("http://www.meteo.unican.es/work/fireDanger/wiki/data/multigrid_hind.rda"))
load(url("http://www.meteo.unican.es/work/fireDanger/wiki/data/multigrid_obs.rda"))
```

FWI calculation

The function `fwiGrid` in package `fireDanger` is a wrapper of function `fwi` in the same package, that handles in a convenient way the model data structures ("grids") provided by the `loadeR` and `loadeR.ECOMS` packages. It operates on either ordinary grids (e.g. gridded observations, reanalysis...) or multi-member grids, as it is the case of seasonal forecast datasets. In the latter, FWI is computed member by member independently. The output is a grid containing the FWI, as well as all the metadata (dates, attributes etc.).

There are several arguments controlling the different options and output values. If these are not provided, the default FWI is computed. One optional (yet important) argument is the argument `mask`. It is a land mask that avoids FWI calculation on the sea. In this case, WFDEI is a land dataset, and therefore the argument can be omitted (default to `NULL`). However, in the case of CFSv2, it is highly advisable to apply a land-mask to avoid unnecessary calculations.

Calculating observed (WFDEI) FWI

```
obs <- fwiGrid(multigrid = multigrid_obs)
```

Seasonal Forecast Hindcast

In this example, the CFSv2 model mask is used to avoid sea surface when computing the FWI. The mask is loaded by requesting the variable "lm" (land mask) in function `loadECOMS`. To ensure spatial domain consistence between object `multigrid_hind` and the mask we interpolate `cfs_mask` to the `multigrid_hind` grid with functions `interpGrid` and `getGrid`.

```
cfs_mask <- loadECOMS(dataset, var = "lm", latLim = latLim, lonLim = lonLim)
mask <- interpGrid(cfs_mask, getGrid(multigrid_hind))
```

The object `mask` is passed to the argument of the same name:

```
hindcast <- fwiGrid(multigrid = multigrid_hind, mask = mask)
```

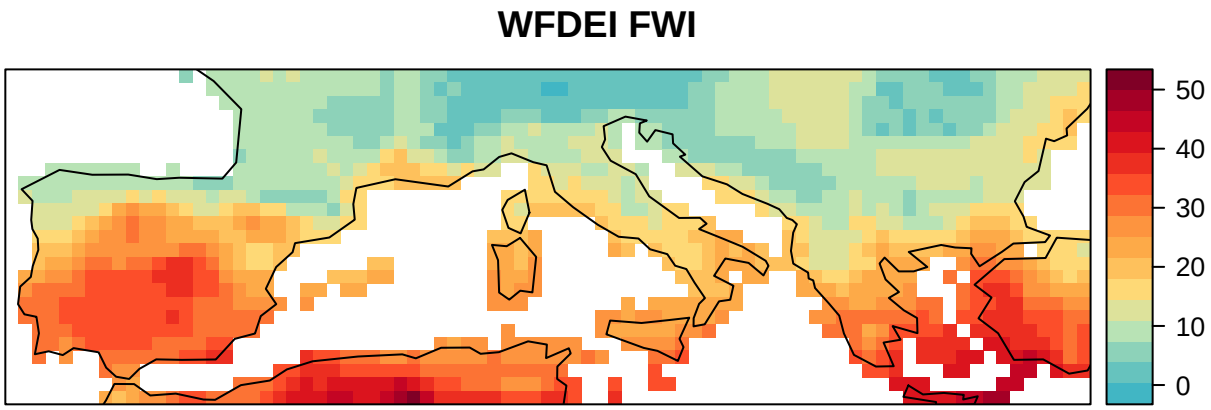
Adjusting the fire season

As indicated in the paper, a spin-up period of 1 month is used to calculate FWI. Now, it is time to remove this first month (May), to retain the data for the fire season (JJAS). This can be done using the `subsetGrid` function from `downscaleR`, allowing flexible grid subsetting along selected dimensions:

```
hindcastJJAS <- subsetGrid(hindcast, season = 6:9)
obsJJAS <- subsetGrid(obs, season = 6:9)
```

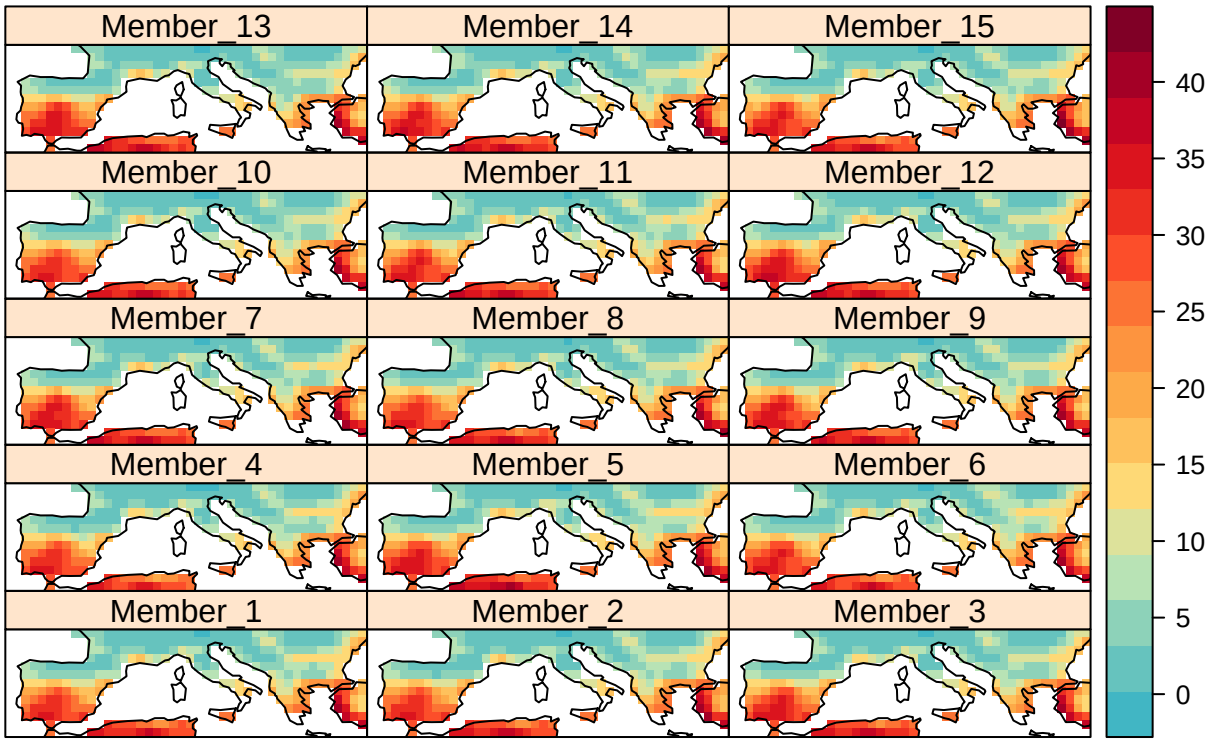
The mean FWI of the considered period (1983-2007) is mapped next, for the observation data,

```
library(RColorBrewer)
fwi.colors <- colorRampPalette(c(rev(brewer.pal(9, "YlGnBu")[3:5]),
                                 brewer.pal(9, "YlOrRd")[3:9]))

plotClimatology(climatology(obsJJAS),
                backdrop.theme = "coastline",
                col.regions = fwi.colors,
                main = "WFDEI FWI")
```



and the forecast data.

```
plotClimatology(climatology(hindcastJJAS),
                backdrop.theme = "coastline",
                col.regions = fwi.colors,
                main = "CFS FWI")
```

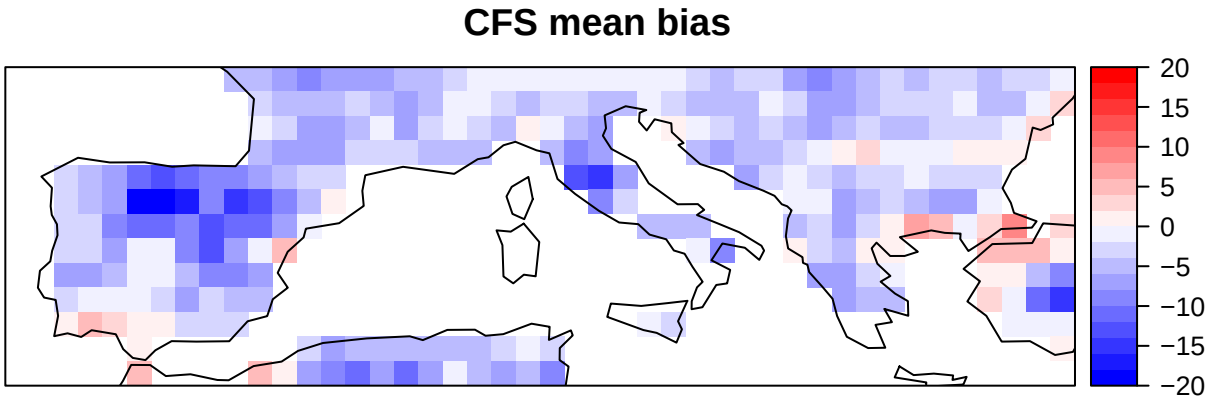


To calculate the mean CFS bias with respect to WFDEI we will perform the multi-member mean of the forecast with function `aggregateGrid`. Then, we will interpolate the observed FWI to the CFS's grid, to operate on the same spatial data, enabling the bias calculation. This is easily done with functions `interpGrid` and `getGrid`. The bias is assigned to a new grid object ("bias").

```
hindcast_aggr <- aggregateGrid(hindcastJJAS, aggr.mem = list(FUN = mean))
bias <- hindcast_aggr
obsJJAS_int <- interpGrid(grid = obsJJAS, new.coordinates = getGrid(hindcast_aggr))
bias$Data <- hindcast_aggr$Data - obsJJAS_int$Data
```

The mean bias can be visualized applying funcions `plotClimatology` and `climatology`. The last will compute the mean of the time-period by default.

```
bias.colors <- colorRampPalette(c("blue", "white", "red"))
plotClimatology(climatology(bias),
                backdrop.theme = "coastline",
                col.regions = bias.colors,
                at = seq(-20,20,2),
                main = "CFS mean bias")
```



Bias Correction

In the `downscaleR` package, the user can find the standard bias correction techniques used in the literature (scaling factors and quantile mapping) as well as other recently published extensions of these techniques (e.g. the multi-variable bias-correction ISI-MIP method, Hempel et al. 2013). The bias correction methods implemented in `downscaleR` are included in the functions `biasCorrection` and `isimip`. For more information, visit the wiki of downscaleR.

Empirical quantile mapping approach:

```
hindcast_bc <- biasCorrection(y = obsJJAS, x = hindcastJJAS,
                              newdata = hindcastJJAS, method = "eqm")
```

Calculating FWI climatologies

Climatologies are easily calculated by means of function `aggregateGrid`. Here, data is annually aggregated by the desired aggregation function.

fwimean

fwimean is the seasonal mean value.

```
fwimean_obs <- aggregateGrid(obsJJAS, aggr.y = list(FUN = mean, na.rm = TRUE))
```

The same procedure is applied to the hindcast:

```
#bias corrected data
fwimean_hind <- aggregateGrid(hindcast_bc, aggr.y = list(FUN = mean, na.rm = TRUE))
#raw data
fwimean_hind_raw <- aggregateGrid(hindcast, aggr.y = list(FUN = mean, na.rm = TRUE))
```

Other common FWI climatologies are FWI90 (percentile 90) and FOT30 (frequency of days above the FWI value 30):

FWI90

```
fwi90_obs <- aggregateGrid(obs, aggr.y = list(FUN = quantile, probs = .9, na.rm = T))
```

FOT30

```
fwi30_obs <- aggregateGrid(obs, aggr.y = list(FUN = function(x) sum(x > 30)/length(x)))
```

In the following we will use only the `fwimean` climatology.

Forecast verification

We use function `veriApply` of package `easyVerification` to compute the different verification measures. In this example we will compute the ROC Skill Score (ROCSS) (verifun = "EnsRocss"). In the manuscript, the bias-corrected and the raw FWI of the forecasts are validated (objects `fwimean_hind` and `fwimean_hind_raw` in this tutorial). For brevity, we will only consider the bias-corrected data in the following, this is, object `fwimean_hind`.

Before calculating the skill, we correct the trend in the data with function `detrendGrid`.

```
fwimean_obs_detrend <- detrendGrid(fwimean_obs)
fwimean_hind_detrend <- detrendGrid(fwimean_hind)
```

ROC Skill Score

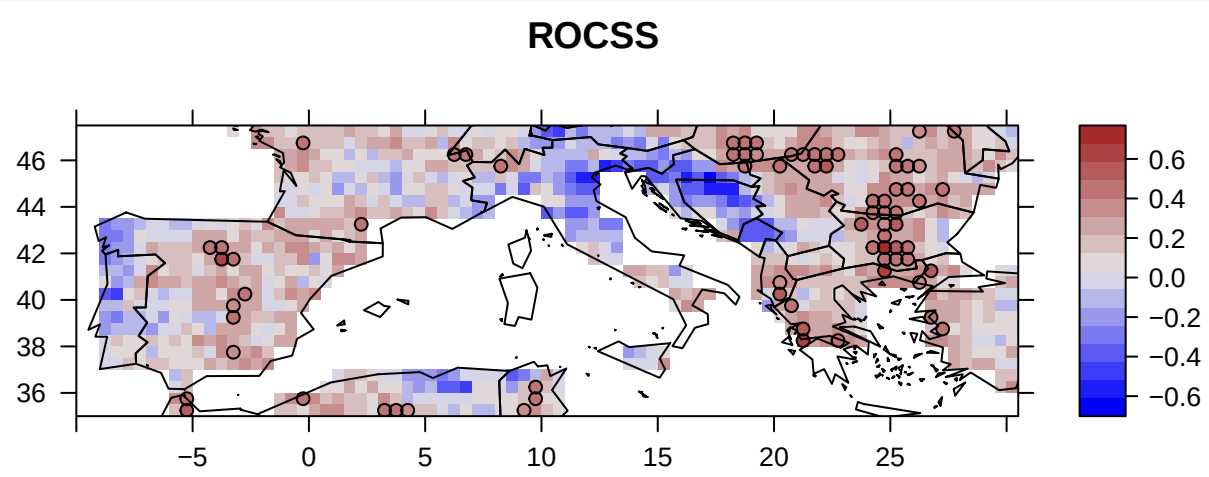
The tercile-based probabilistic approach described in the manuscript (sec. 2.5.1. ROC Skill Score) is applied by means of function `veriApply`. In this case we will extract the ROC Skill score of the upper-tercile, for which the threshold is set in 2/3 (argument `prob`). This way two FWI categories are obained (i.e. above-normal and the rest). If three categorizations are preferred a vector of two thresholds is passed to argument `prob` (`prob = c(1/3, 2/3)`) obtaining the categories described in the manuscript (i.e. normal, below-normal and above-normal).

```
skill <- veriApply("EnsRocss",
                   fcst = fwimean_hind_detrend$Data,
                   obs = fwimean_obs_detrend$Data,
                   prob = 2/3,
                   ensdim = 1,
                   tdim = 2,
                   na.rm = TRUE)

upper.tercile <- easyVeri2grid(easyVeri.mat = skill$cat2,
                               obs.grid = fwimean_obs_detrend,
                               verifun = "EnsRocss")
```

Note that `skill$cat2` corresponds to the second category, this is the upper-tercile (above-normal). The skill map, as those corresponding to figure 3 in the manuscript, is obtained by extracting the significant points and mapping them together with the skill, for which `plotClimatology` function is also used.

```
lons <- rep(fwimean_hind_detrend$xyCoords$x, each = length(fwimean_hind_detrend$xyCoords$y))
lats <- rep(fwimean_hind_detrend$xyCoords$y, length(fwimean_hind_detrend$xyCoords$x))

sig.i <-  skill$cat2 > skill$cat2.sigma*qnorm(0.95)
sig.points <- sp::SpatialPoints(na.omit(cbind(lons[sig.i], lats[sig.i])))
sig <- list("sp.points", sig.points, pch = 21, col = "black", which = 1)
```

```
ms.colors <- colorRampPalette(c("blue", "grey90", "brown"))
plotClimatology(upper.tercile,
                scales = list(draw = TRUE, x=list(alternating=FALSE,tick.number = 12)),
                backdrop.theme = "countries",
                names.attr = c("cat1", "cat2", "cat3"),
                main = "ROCSS",
                col.regions = ms.colors,
                sp.layout = list(sig))
```



Visualization

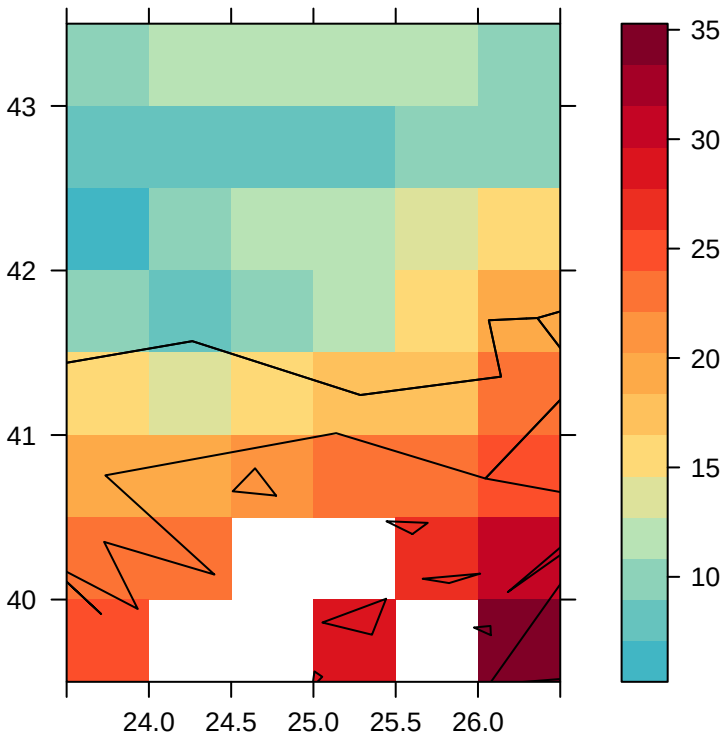
The `visualizeR` package is aimed at the visual communication of the uncertainty in seasonal forecasts, providing tools that incorporate different verification methods. In the following sections, we illustrate three different visualization examples involving functions `tercilePlot`, `reliabilityCategories` and `bublePlot`. Further worked examples on the use and installation of `visualizeR` can be found in the corresponding gitHub repository: https://github.com/SantanderMetGroup/visualizeR

```
library(visualizeR)
```

Tercile validation

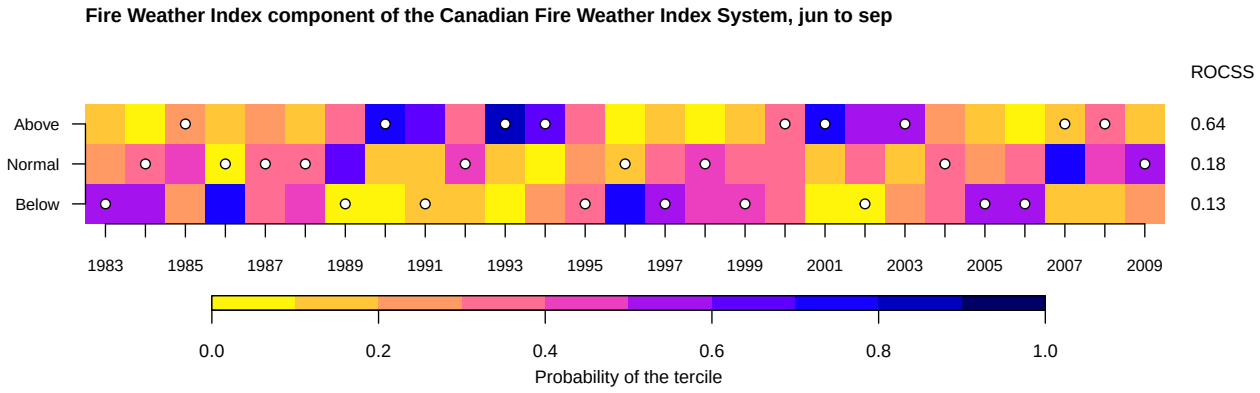
In order to gain a better insight into the areas of high skill (sec. 3.3 in the manuscript), next we will spatially subset the data with function `subsetGrid`. The extracted region encompasses grid boxes over Greece and Bulgaria.

```
fwimean_hind_sub <- subsetGrid(fwimean_hind_detrend,
                               latLim = c(40,43.5), lonLim = c(24,26.5))
fwimean_obs_sub <- subsetGrid(fwimean_obs_detrend,
                              latLim = c(40,43.5), lonLim = c(24,26.5))
plotClimatology(climatology(fwimean_obs_sub),
                backdrop.theme = "countries",
                col.regions = fwi.colors,
                scales = list(draw = TRUE, x=list(alternating=FALSE, tick.number = 6)))
```



The tercile validation plot is obtained by function `tercileValidation`. Terciles are arranged by rows. Further details for graph interpretation are given in Sec. 2.5.2. of the manuscript.

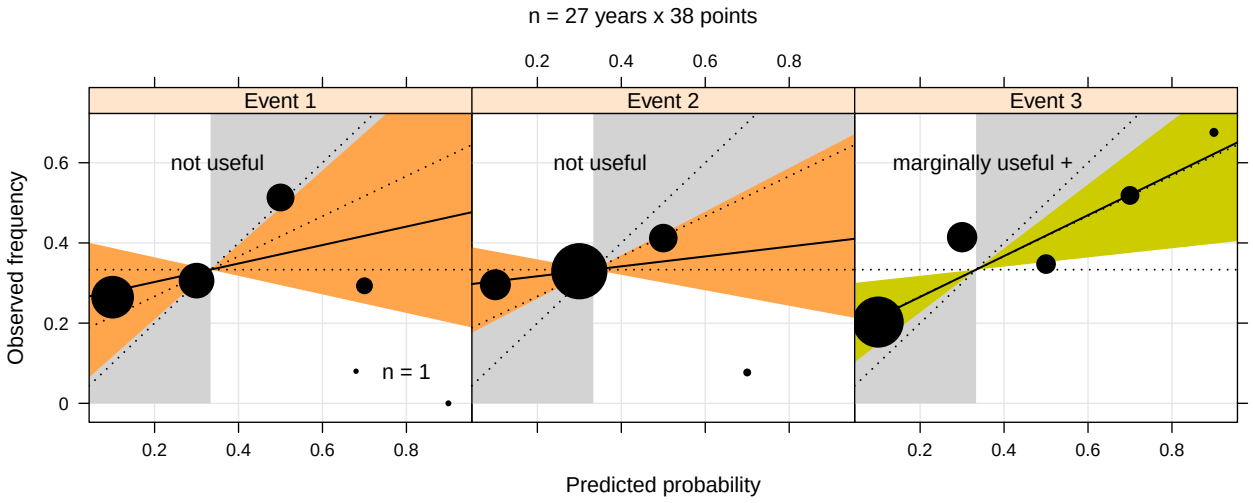
```
tercilePlot(fwimean_hind_sub, obs = fwimean_obs_sub, color.pal = "ypb")
```



Reliability

Function `reliabilityCategories` computes the reliability category of the considered domain. For instance, the following example shows reliability diagrams obtained for the low, normal and high terciles (Event1, Event2 and Event3) in the selected region. Whereas the diagonal —indicated by a dashed gray line— would mark the perfect reliability, points falling in the so-called skill region (in gray) still contribute positively to the forecast skill. The reliability line (solid black line) is the result of fitting the points, which are represented in different size according to the number of forecasts in each bin.

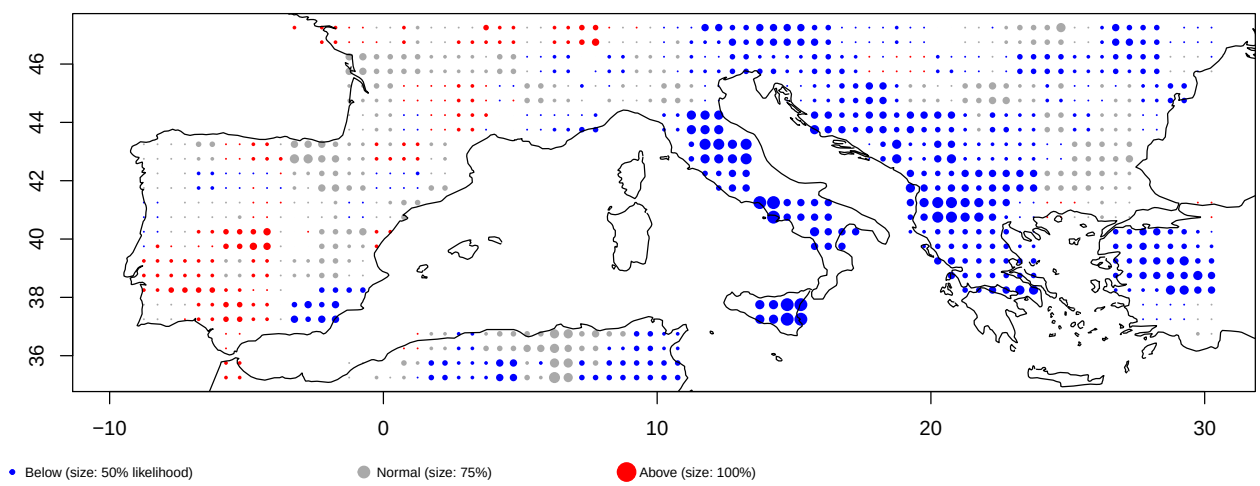
```
reliabilityCategories(obs = fwimean_obs_sub, hindcast = fwimean_hind_sub,
                      n.events = 3, n.bins = 5)
```



Bubble plot

The `bubblePlot` function combines in a geographic map several aspects of the seasonal forecast system with different levels of complexity using three properties of the bubbles: colour, size and transparency. Type `?bubblePlot` for details.

```
bubblePlot(hindcast = fwimean_hind, obs = fwimean_obs,
           bubble.size = 3,
           size.as.probability = TRUE, score = FALSE)
```




```
sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.5 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8          LC_NUMERIC=C
##  [3] LC_TIME=es_ES.UTF-8           LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=es_ES.UTF-8       LC_MESSAGES=en_US.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] visualizeR_0.2-0        sm_2.2-5.4
##  [3] RColorBrewer_1.1-2      easyVerification_0.2.0
##  [5] SpecsVerification_0.4-1 downscaleR_2.0-0
##  [7] transformeR_0.0.7       loadeR.ECOMS_1.2-0
##  [9] loadeR_1.0-7            loadeR.java_1.1-0
## [11] rJava_0.9-8             fireDanger_1.0-0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.10        plyr_1.8.4          bitops_1.0-6
##  [4] tools_3.3.2         vioplot_0.2         boot_1.3-17
##  [7] digest_0.6.8        evd_2.3-2           evaluate_0.10
## [10] lattice_0.20-31     Matrix_1.2-7.1      yaml_2.1.13
## [13] parallel_3.3.2      spam_1.4-0          akima_0.6-2
## [16] stringr_1.0.0       knitr_1.15.1        raster_2.5-8
## [19] mapplots_1.5        fields_8.10         maps_3.1.1
## [22] rprojroot_1.2       grid_3.3.2          dtw_1.18-1
## [25] pbapply_1.3-1       rmarkdown_1.3       sp_1.2-4
```

```
## [28] magrittr_1.5        scales_0.4.1        CircStats_0.2-4
## [31] backports_1.0.5     htmltools_0.3.5     MASS_7.3-44
## [34] abind_1.4-5         colorspace_1.3-2    proxy_0.4-16
## [37] stringi_0.4-1       munsell_0.4.3       RCurl_1.95-4.8
## [40] verification_1.42   RcppEigen_0.3.2.4.0
```
