Bakeoff Pipeline

Anna Norberg 2017

Bakeoff is a pipeline for fitting various single and joint species distribution models to community data. The pipeline includes importing and formatting the data, fitting the models, and producing interpolative and extrapolative predictions based on validation data.

All the scripts sourced, the data used and the folder structure are available in Github: AnnaNorb/bakeoff. For the pipeline to run smoothly, the user needs to download or otherwise construct an indentical folder structure and have the data in its appropriate location.

For the method HMSC, all models are fitted and predictions are made in Matlab. This R pipeline assumes that these steps have already been done. For this, see the document 'bakeoffHMSCvignette'.

1 Preparations

First clean the workspace and define the path for the location of the 'settings.r' script.

```
rm(list = ls(all=TRUE))
gc()

SETT<-".../bakeoff/pipeline/scripts/settings.r"
source(SETT)
setwd(WD)
```

Next the user should check whether she has the required packages installed already and install the ones missing. With the command below, all the required packages are downloaded, along with their dependencies. Note that for the package 'mvpart' one should tun only the lines either suited for a computer with OS X or Windows.

```
install.packages(c("pROC","mvabund","MultivariateRandomForest","randomForest",
                    "e1071","gbm","dismo","yaImpute","earth","devtools","glmnet",
                    "boral","gjam","spaMM","nlme","MASS","spaMM","vegan"))
library(devtools)
install_github("davharris/mistnet2")

# OS X
    install.packages("doMC")
    install.packages(paste(WD,"MODELS/mvpart_pkg/mvpart_1.6-2.tar",sep=''), repos = NULL, type="source"
# Windows
    install.packages("doParallel")
    install.packages(paste(WD,"MODELS/mvpart_pkg/mvpart_1.6-2.zip",sep=''), repos = NULL, type="source"
```

Tthe 'mvpart' package, it is available from CRAN archive (https://cran.r-project.org/src/contrib/Archive/mvpart/), and it can be downloaded from there, but it is included in the pipeline, and can hence be installed using the lines below.

Species archetypes models are provided as a source code along with the pipeline ("…/pipeline/MODELS/sam/source"). The code is sourced within the model fitting prcedure.

If you are using OS X, install also the package 'doMC', whereas if you are using Windows, install 'doParallel' insted.

```
# OS X
    install.packages("doMC", dependencies=TRUE)) #OSX
```

```
# Windows
    install.packages("doParallel", dependencies=TRUE)) #Win
```

Next, within the script 'settings.r', fill in the path 'WD', which is the path to the location of the pipeline.

```
WD <- ".../pipeline"
```

Within the same script, the user should also define setting for parallel computing. With the first line, the user defines the number of cores to use. With the second line the user registers the multicore parallel backend.

```
# OS X
library('doMC')
crs<-4 #change this number according to the computer used
registerDoMC(cores=crs)

# Windows
library('doParallel')
crs<-4 #change this number according to the computer used
registerDoParallel(cl=makeCluster(crs))
```

2 Model fitting and predictions

Next, the user defines the size of the data. The provided data sets have been divided into training and validation beforehand (see the main text of Norberg et al. for details), and the all the sets are of the same size. Small data set means that the models are fitted to the first 300 sampling units of the sets, whereas large set means the full set of 600 sampling units. The custom data set size allows the user to define both the sampling units and the species selected. The default option is to use the smaller subset, as the other options have been commented out.

```
sz<-1   # small
#sz<-2  # large
#sz<-3  # custom
    #customData<-list(1:100,1:10)   # [[1]] #sampling units [[2]] #species
```

Then we can fit the models and make predictions, one data set at a time. Within the loop, also a matrix for the computation times is initiated and within the model fitting scripts, the computation times are saved and they are saved in the results directory.

```
for (d in 1:length(Sets)) { # loop over data sets

    set_no <- Sets[d]
    source(readdata)

    comTimes<-rep( list(list()), nmodels )  # initiate computation times
    names(comTimes)<-mod_names

    # FIT MODELS
    ##############

    source(paste(MD,"fit.glm.r",sep=""))        # ssGLM 1
    source(paste(MD,"it.glmmPQL.r",sep=""))     # ssGLM 2
    source(paste(MD,"fit.spaMM.r",sep=""))      # ssGLM 3
    source(paste(MD,"fit.manyglm.r",sep=""))    # MVABUND
    source(paste(MD,"fit.gam.r",sep=""))        # GAM
    source(paste(MD,"fit.mrt.r",sep=""))        # MRTs
    source(paste(MD,"fit.rf.r",sep=""))         # RFs
    source(paste(MD,"fit.brt.r",sep=""))        # BRT
    source(paste(MD,"fit.svm.r",sep=""))        # SVM
    source(paste(MD,"fit.gnn.r",sep=""))        # GNNs
    source(paste(MD,"fit.mars.r",sep=""))       # MARS
    source(paste(MD,"fit.gjam.r",sep=""))       # GJAMS
    source(paste(MD,"fit.mistnet.r",sep=""))    # mistnet
    source(paste(MD,"fit.mvrf.r",sep=""))       # MVRF
    source(paste(MD,"fit.sam.r",sep=""))        # SAMs
    source(paste(MD,"fit.spbayes.r",sep=""))    # SPBAYES

    # SAVE COMPUTATION TIMES
    #########################

    save(comTimes, file=paste(WD,"comTimes_",set_no,dataN[sz],'.RData',sep="")) # save computation times

    # GET PREDICTIONS
    ###################

    # ssHMSC
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.hmsc.ss.r",sep=""))

    # ss GLM 1
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.glm.r",sep=""))

    # ss GLM 2
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.glmmPQL.r",sep=""))

    # ss GLM 3
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.spaMM.r",sep=""))

    # TRAITGLM
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.traitglm.r",sep=""))

    # GAM
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.gam.r",sep=""))

    # BRT
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.brt.r",sep=""))

    # SVM
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.svm.r",sep=""))

    # RFs
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
    source(paste(PD,"predict.rf.r",sep=""))
```

```
# GNNs
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.gnn.r",sep=""))

# MRTs
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.mrt.r",sep=""))

# MARS
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.mar.r",sep=""))

# MVRF
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.mvrf.r",sep=""))

# GJAMS
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.gjam.r",sep=""))

# BORAL
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.boral.r",sep=""))

# SAMs
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.sam.r",sep=""))

# mistnet
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.mistnet.r",sep=""))

# SPBAYES
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.spbayes.r",sep=""))

# HMSC
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.hmsc.all.r",sep=""))

}
```

3 Measures for evaluating the predictive performance of the models