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, producing 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 required folder structure is created as a part of the pipeline to the location indicated by the user. Be sure, that the location of the pipeline (where it is downloaded), is also a location where additional results folders can be created and results saved in the end.

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.

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

Then indicate your operating system: "osx" for macOS, or "win" for Windows.

```
OS<-"osx"
#OS<-"win"
```

Define the path for the location of the 'settings.r' script, which defines the paths to all files of the pipeline and runs a series of setting.

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

Next we will create the results folders unless they already exist, in which case the following lines can be ignored. If the folder structure exists and the following lines are ran, some warnings are produced, which can be ignored.

```
dir.create(PD2)
dir.create(RD)
for (set in Sets) {
    dir.create(paste(RD,set,sep=""))
    }
dir.create(RDfinal)
dir.create(paste(RDfinal,"300/",sep=""))
dir.create(paste(RDfinal,"600/",sep=""))
dir.create(paste(RDfinal,"custom/",sep=""))
```

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.

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

```
if (OS=="osx") {
    install.packages("doMC")
    install.packages(paste(WD,"MODELS/mvpart_pkg/mvpart_1.6-2.tar",sep=''), repos = NULL, type="source")
    }
if (OS=="win") {
    install.packages("doParallel")
    install.packages(paste(WD,"MODELS/mvpart_pkg/mvpart_1.6-2.zip",sep=''), repos = NULL, type="source")
    }
```

The 'mvpart' package is available from the 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.

Then select the number of cores you wish to use in parallel computing and set the clusters.

```
crs<-4
if (OS=="osx") {
    require(doParallel)
    registerDoParallel(cl=makeCluster(crs))
}
if (OS=="win") {
    require(doMC)
    registerDoMC(cores=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, for the selected data size. Within the model fitting scripts, also computation times are calculated and they are saved in the results directory.

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

    set_no <- Sets[d]
    source(readdata)

    # FIT MODELS
    ##############
    source(paste(MD,"fit.glm.r",sep=""))        # ssGLM 1
    source(paste(MD,"fit.glmmPQL.r",sep=""))    # ssGLM 2
    source(paste(MD,"fit.gam.r",sep=""))        # GAM
    source(paste(MD,"fit.spaMM.r",sep=""))      # ssGLM 3
    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.sam.r",sep=""))        # SAMs
    source(paste(MD,"fit.bc.r",sep=""))         # BayesComm
    source(paste(MD,"fit.boral.r",sep=""))      # BORAL
    source(paste(MD,"fit.traitglm.r",sep=""))   # MVABUND
    source(paste(MD,"fit.mvrf.r",sep=""))       # MVRF
    source(paste(MD,"fit.spbayes.r",sep=""))    # SPBAYES


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

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

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

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

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

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

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

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

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

    # GNNs
    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(readdata); setwd(WD)
```

```
source(paste(PD,"predict.gnn.r",sep=""))

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

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

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

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

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

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

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

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

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

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

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

}
```

3 Measures for evaluating the predictive performance of the models

Finally we produce the measures for evaluating the predictive performance of the modelling frameworks and their variants.

```
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); setwd(WD)

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

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

    set_no <- Sets[d]
    source(readdata)
    source(modpreds)

    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata); setwd(WD)
    source(paste(RD,"aucs.r",sep=""))

    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata); setwd(WD)
    source(paste(RD,"liks.r",sep=""))

    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata); setwd(WD)
    source(paste(RD,"mse.r",sep=""))

    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata); setwd(WD)
    source(paste(RD,"spearm.r",sep=""))

    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata); setwd(WD)
    source(paste(RD,"sds.r",sep=""))

    rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata); setwd(WD)
    source(paste(RD,"p50.r",sep=""))

    }
```