

Bakeoff Pipeline

Anna Norberg

02 April 2018

Bakeoff is a pipeline for fitting various single and joint species distribution models to community data. The main script used is ‘bakeoff.pipeline.r’, which guides through the 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](https://github.com/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. Please make 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.

The pipeline works in R environment, except for the method HMSC, for which all models are fitted and predictions made in Matlab (see the document ‘bakeoffHMSCvignette’). The workflow is the following: 1) first we construct the folder structure needed, 2) then fit the HMSC models in Matlab, 3) after which we fit all models in R, 4) and finally produce all predictions and performance measures.

1 Preparations

First clean the workspace.

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

Then identify and indicate your operating system.

```
get_os <- function() {  
  if (.Platform$OS.type == "windows") {  
    "win"  
  } else if (Sys.info()["sysname"] == "Darwin") {  
    "osx"  
  } else if (.Platform$OS.type == "unix") {  
    "unix"  
  } else {  
    stop("Unknown OS")  
  }  
}  
OS<-get_os()
```

Define the path for the location of the ‘bakeoff’ folder, which defines the paths to all files of the pipeline, including the script ‘settings.r’, which and runs a series of setting.

```
pth<-"~/OneDrive - University of Helsinki/"  
SETT<-paste(pth,"bakeoff/pipeline/SCRIPTS/settings.r",sep="")  
source(SETT)  
setwd(WD)
```

Next we will create the results folders unless they already exist, in which case the following lines 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,"150/",sep=""))
dir.create(paste(RDfinal,"300/",sep=""))
```

At this point, the user should switch to fitting the HMSC models: open the ‘bakeoffHMSCvignette’ and follow the workflow until you have successfully produced the fits and predictions.

After returning to the R pipeline, 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 (“.../MODELS/coordinSAMsv2.R”). The code is sourced within the model fitting procedure.

Then select the number of cores (‘crs’) you wish to use in parallel computing and set the clusters. The default is two cores.

```
crs<-2
if (OS=="osx") {
  require(doParallel)
  registerDoParallel(cl=makeCluster(crs))
}
if (OS=="win") {
  require(doMC)
  registerDoMC(cores=crs)
}
```

2 Model fitting and predictions

The provided data sets have been divided into training and validation beforehand (see the main text of Norberg et al. for details). The set of 300 sampling units is randomly sampled from the full set and the set of 150 is randomly sampled from this set.

Then the user has to choose whether to fit the Bayesian methods with shorter or longer chains. The default (‘FALSE’) means shorter chains (max. MCMC 50 000 iterations and/or 24h fitting time), and ‘TRUE’ mean double the default.

```
MCMC2<-FALSE
```

Then we can fit the models and make predictions. Within the model fitting scripts, also computation times are calculated and they are saved in the results directory.

```
for (sz in 1:2) {

  for (d in 1:length(Sets)) {

    set_no <- Sets[d]
    source(readdata)

    # fit
    #####
    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.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.manyglm.r",sep=""))  # MVABUND
    source(paste(MD,"fit.bc.r",sep=""))       # BayesComm
    source(paste(MD,"fit.sam.r",sep=""))      # SAMs
    source(paste(MD,"fit.boral.r",sep=""))    # BORAL

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

```

rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.gam.r",sep=""))
# RFs
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.rf.r",sep=""))
# MRTs
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.mrt.r",sep=""))
# GJAMS
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.gjam.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=""))
# BayesComm
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.bc.r",sep=""))
# BORAL
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.boral.r",sep=""))
# ssHMSC
rm(list=ls()[!(ls() %in% saveobjs)]); gc(); source(SETT); source(readdata)
source(paste(PD,"predict.hmsc.ss.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

First we calculate species-specific occurrence probabilities, species richnesses and beta index values.

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

```
ENS<-list(NULL,unlist(mod_names),c("HMSC3", "MARS1", "BRT1", "MISTN1", "GNN1"))
PRV<-list(NA, 0.1)
```

```

for (e in 1:length(ENS)) {
  for (p in 1:length(PRV)) {
    opts<-list(modelEnsemble=ENS[[e]],prevalenceThreshold=PRV[[p]])
    for (sz in 1:2) {
      for (d in 1:length(Sets)) {
        set_no <- Sets[d]; source(readdata)
        source(paste(modpredsFolder,"sp_occ_probs.r",sep="/"))
      }
      for (d in 1:length(Sets)) {
        set_no <- Sets[d]; source(readdata)
        source(paste(modpredsFolder,"sp_rich_site.r",sep="/"))
      }
    }
  }
}

```

```

    }
    for (d in 1:length(Sets)) {
      set_no <- Sets[d]; source(readdata)
      source(paste(modpredsFolder,"beta_inds.r",sep="/"))
    }
  }
}

```

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

```
rm(list=ls()[!(ls() %in% saveobjs2)]); gc(); source(SETT); setwd(WD)
```

```

ENS<-list(NULL,
  unlist(mod_names),
  c("HMSC3", "MARS1", "BRT1", "MISTN1", "GNN1"))
PRV<-list(NA, 0.1)

```

```

for (e in 1:length(ENS)) {
  for (p in 1:length(PRV)) {
    opts<-list(modelEnsemble=ENS[[e]],prevalenceThreshold=PRV[[p]])
    for (sz in 1:2) {
      PMS<-NA
      PMs<-list()
      for (d in 1:length(Sets)) {
        source(SETT); set_no <- Sets[d]; source(readdata)
        source(paste(RD,"expct.r",sep=""))
        source(paste(RD,"aucs.r",sep=""))
        source(paste(RD,"sqrtPrs.r",sep=""))
        source(paste(RD,"probBinRMSE.r",sep=""))
        source(paste(RD,"mse.r",sep=""))
        source(paste(RD,"spearm.r",sep=""))
        source(paste(RD,"sds.r",sep=""))
        source(paste(RD,"p50.r",sep=""))
        pms<-simplify2array(PMs)
        PMS<-rbind(PMS,pms)
      }
      PMS<-PMS[-1,]

      filebody<-paste(RDfinal,dataN[sz],"/meta_analysis/PMS",sep="")
      if (is.numeric(prevThrs)) {
        filebody<-paste(filebody,"_spThr",prevThrs*100,sep="")
      }
      if (is.null(ensmblModels)!=TRUE){
        if(length(ensmblModels)==nmodels){
          ensmbl<-"all"
        }
        if(length(ensmblModels)!=nmodels){
          ensmbl<-paste(ensmblModels,collapse="_")
        }
        filebody<-paste(filebody,"_ensmbl_",ensmbl,sep="")
      }
      save(PMS, file=paste(filebody,".RData",sep=""))
    }
  }
}

```

```

    }
  }
}

```

And then we compile all the results into one table.

```

rm(list=ls()[!(ls() %in% saveobjs2)]); gc(); source(SETT); setwd(WD)

TBL_PMS_ALL<-NA
for (e in 1:length(ENS)) {
  for (p in 1:length(PRV)) {
    opts<-list(modelEnsemble=ENS[[e]],prevalenceThreshold=PRV[[p]])
    TBL_PMS_all<-NA
    for (sz in 1:2) {
      filebody<-paste(RDfinal,dataN[sz],"/meta_analysis/PMS",sep="")
      if (is.numeric(opts$prevalenceThreshold)) {
        filebody<-paste(filebody,"_spThr",opts$prevalenceThreshold*100,sep="")
      }
      if (is.null(opts$modelEnsemble)!=TRUE) {
        if(length(opts$modelEnsemble)==nmodels){
          ensmb1<-"all"
        }
        if(length(opts$modelEnsemble)!=nmodels){
          ensmb1<-paste(opts$modelEnsemble,collapse="_")
        }
        filebody<-paste(filebody,"_ensmb1_",ensmb1,sep="")
      }
      load(file=paste(filebody,".RData",sep=""))

      noMod<-dim(PMS)[1]/(3*length(Sets))

      if (is.null(opts$modelEnsemble)!=TRUE) {
        tblrep<-matrix(rep(c(paste("ENS",ensmb1,sep="_"),"ensemble",
          1*(colSums(feats[ENS[[e]],3:ncol(feats)])>0)),
          each=(length(Sets)*3)),nrow=length(Sets)*3)
        colnames(tblrep)<-colnames(feats)
        TBL<-cbind(tblrep,dataN[sz],rep(rep(c("i","e1","e2"),
          each=noMod),length(Sets)),rep(Sets,each=(noMod*3)))
      } else {
        tblrep<-apply(feats,2,rep,times=3*length(Sets))
        TBL<-cbind(tblrep,dataN[sz],rep(rep(c("i","e1","e2"),each=noMod),
          length(Sets)),rep(Sets,each=(noMod*3)))
      }
      colnames(TBL)[(ncol(TBL)-2):ncol(TBL)]<-c("dataSize","predType","dataSet")
      TBL_PMS<-cbind(TBL,PMS,PRV[[p]])
      colnames(TBL_PMS)[ncol(TBL_PMS)]<-"prevThr"
      #print(paste("Number of NAs",sum(is.na(TBL_PMS)),"for data size",dataN[sz]))
      TBL_PMS_all<-rbind(TBL_PMS_all,TBL_PMS)
    }
  }
  TBL_PMS_all<-TBL_PMS_all[-1,]

  filebody<-paste(RDfinal,dataN[sz],"/meta_analysis/finalTBLall",sep="")
  if (is.numeric(opts$prevalenceThreshold)) {
    filebody<-paste(filebody,"_spThr",opts$prevalenceThreshold*100,sep="")
  }
}

```

```

    }
    if (is.null(opts$modelEnsemble)!=TRUE) {
      if(length(opts$modelEnsemble)==nmodels){
        ensmb1<-"all"
      }
      if(length(opts$modelEnsemble)!=nmodels){
        ensmb1<-paste(opts$modelEnsemble,collapse="_")
      }
      filebody<-paste(filebody,"_ensmb1_",ensmb1,sep="")
    }
    save(TBL_PMS_all, file=paste(filebody,".RData",sep=""))
    TBL_PMS_ALL<-rbind(TBL_PMS_ALL,TBL_PMS_all)
  }
}
TBL_PMS_ALL<-TBL_PMS_ALL[-1,]
rownames(TBL_PMS_ALL)<-NULL
save(TBL_PMS_ALL, file=paste(RDfinal,"TBL_PMS_ALL.RData",sep=""))
write.table(TBL_PMS_ALL,file=paste(RDfinal,"TBL_PMS_ALL.csv",sep=""),
            sep=";",row.names=F,col.names=T)

```

Let's plot the results. The results are otherwise raw, except that we'll take the reciprocal of some of the measures, so that a bigger number is always better for all the measures.

```

rm(list=ls()[!(ls() %in% c("OS","pth","SETT"))]); gc()
source(SETT)
setwd(WD)

load(file=paste(RDfinal,"TBL_PMS_ALL.RData",sep=""))
resTBL<-data.frame(TBL_PMS_ALL)
resTBL[,c(3:8,11:(ncol(resTBL)-1))]<-apply(resTBL[,c(3:8,11:(ncol(resTBL)-1))],2,as.numeric)
pms<-colnames(resTBL[,11:(ncol(resTBL)-1)])
pms<-pms[c(1,2,4,3,
           5,9,17,13,
           6,10,18,14,
           7,11,19,15,
           8,12,20,16)]
resTBL[,minTomaxBest] <- -resTBL[,minTomaxBest]

pdf(file=paste(RDfinal,"raw_res_fig.pdf",sep=""),bg="transparent",width=15,height=15)
par(family="serif",mfrow=c(5,4),mar=c(7,3,2,1))
for (p in 1:length(pms)) {
  plot(0,0,xlim=c(0,length(mod_names2)),ylim=c(min(resTBL[,pms[p]]),max(resTBL[,pms[p]])),
       type='n',xaxt='n',xlab="",ylab="",main=pms[p])
  for (m in 1:length(mod_names)) {
    lines(resTBL[which(resTBL$Abbreviation==mod_names[[m]]),pms[p]],
          x=rep(m,times=length(resTBL[which(resTBL$Abbreviation==mod_names[[m]]),pms[p]])),
          lwd=2)
    points(mean(resTBL[which(resTBL$Abbreviation==mod_names[[m]]),pms[p]]),
           x=m,pch=21,col="black",bg="red3",cex=2)
  }
  axis(1,1:length(mod_names),unlist(mod_names),las=2)
}
dev.off()

```

Finally, let's have a look at the computation times. First compile the results.

```
rm(list=ls()[!(ls() %in% c("OS","pth","SETT"))]); gc(); source(SETT); setwd(WD)
require(rmatio)
require(R.matlab)

for (sz in 1:2) {
  for (d in 1:length(Sets)) {
    set_no <- Sets[d]
    comtimes<-list()
    for (m in 1:length(mod_names)) {
      if (m <= 20) {
        load(file=paste(FD,set_no,"/comTimes_",mod_names[m],"_",dataN[sz],
                        ".RData",sep=""))
        comtimes[[m]]<-as.numeric(comTimes, units = "mins")
      }
      if (m > 20 & m <= 22) {
        tmp<-read.mat(filename=paste(FD,set_no,"/ssHMSC/compTime_",Sets[d],"_",
                                    mod_names[m],"_",dataN[sz],".mat",sep=""))
        comtimes[[m]]<-as.numeric(tmp)/60
      }
      if (m > 22) {
        tmp<-read.mat(filename=paste(FD,set_no,"/compTime_hmsc_",Sets[d],"_",
                                    m-22,"_",dataN[sz],".mat",sep=""))
        comtimes[[m]]<-as.numeric(tmp)/60
      }
    }
    save(comtimes, file=paste(RD2,set_no,"/comtimes_",dataN[sz],".RData",sep=""))
  }
}
```

And the plot them.

```
rm(list=ls()[!(ls() %in% c("OS","pth","SETT"))]); gc(); source(SETT); setwd(WD)

# define the ensemble
ens<-c("HMSC3", "BC2", "BRT1", "MISTN1", "GNN1")

ComTimesAll<-NA
for (sz in 1:2) {
  set_no <- Sets[1]
  load(file=paste(RD2,set_no,"/comtimes_",dataN[sz],".RData",sep=""))
  ComTimes<-unlist(comtimes)
  for (d in 2:length(Sets)) {
    set_no <- Sets[d]
    load(file=paste(RD2,set_no,"/comtimes_",dataN[sz],".RData",sep=""))
    ComTimes<-cbind(ComTimes,unlist(comtimes))
  }
  ComTimes<-cbind(mod_names,ComTimes,rep(dataN[sz],times=nrow(ComTimes)))
  colnames(ComTimes)<-c("Variant",Sets,"dataSize")
  ComTimesAll<-rbind(ComTimesAll,ComTimes)
}
ComTimesAll<-ComTimesAll[-1,]
```



```

comtimes<-ComTimesAll[,2:7]
comtimes<-matrix(unlist(comtimes),ncol=6)
rownames(comtimes)<-ComTimesAll[,1]
colnames(comtimes)<-c(Sets,"size")
modNams<-rownames(comtimes[comtimes[,ncol(comtimes)]=="150",])
comtimes150<-apply(comtimes[comtimes[,ncol(comtimes)]=="150",],2,as.numeric)
comtimes300<-apply(comtimes[comtimes[,ncol(comtimes)]=="300",],2,as.numeric)

# ensembles
comtimes150 <- rbind(comtimes150, c(colSums(comtimes150[, -ncol(comtimes150)]),150))
rownames(comtimes150)<-c(modNams,"ENS_all")
comtimes300 <- rbind(comtimes300, c(colSums(comtimes300[, -ncol(comtimes300)]),300))
rownames(comtimes300)<-c(modNams,"ENS_all")

comtimes150 <- rbind(comtimes150,c(colSums(comtimes150[ens,-ncol(comtimes150)]),150))
rownames(comtimes150)[nrow(comtimes150)]<-paste(c("ENS",ens),collapse="_")
comtimes300 <- rbind(comtimes300,c(colSums(comtimes300[ens,-ncol(comtimes300)]),300))
rownames(comtimes300)[nrow(comtimes300)]<-paste(c("ENS",ens),collapse="_")
comtimes<-rbind(comtimes150,comtimes300)
comtimes<-cbind(rownames(comtimes),comtimes)
write.table(comtimes,file=paste(RDfinal,"comtimes.csv",sep=""),
            row.names=FALSE,col.names=TRUE,sep="," , quote=FALSE)

comTimes<-apply(comtimes[, -1],2,as.numeric)
rownames(comTimes)<-rownames(comtimes)

pdf(file=paste(RDfinal,"/comptimes.pdf",sep=""),bg="transparent",width=10,height=7)
par(family="serif",mar=c(18,4,4,1))
plot(log10(apply(comTimes[which(comTimes[,ncol(comTimes)]==300),-ncol(comTimes)],1,max)),
     col="black",pch=17, xlab="",ylab="",yaxt="n",xaxt="n",
     ylim=c(log10(0.0004),log10(4000)), main="Computation times")
points(log10(apply(comTimes[which(comTimes[,ncol(comTimes)]==150),-ncol(comTimes)],1,min)))
axis(side=2,at=log10(c(0.0005,0.0167,1,10,60,1440,3900)),
     labels=c("0.03 sec","1 sec","1 min","10 min","1 h","24 h","65 h"),las=2)
axis(side=1,at=1:nrow(comTimes[which(comTimes[,ncol(comTimes)]==150),]),
     labels=rownames(comTimes[which(comTimes[,ncol(comTimes)]==150),]),las=2)
for (i in 1:nrow(comTimes[which(comTimes[,ncol(comTimes)]==300),-ncol(comTimes)])) {
  lines(
    y=c(log10(apply(comTimes[which(comTimes[,ncol(comTimes)]==150),-ncol(comTimes)],1,min))[i],
        log10(apply(comTimes[which(comTimes[,ncol(comTimes)]==300),-ncol(comTimes)],1,max))[i]),
    x=c(i,i))
}
abline(h=log10(1440))
dev.off()

```