Orthopoxvirus Host Prediction Model Code

Katie Tseng, Dan Becker, Colin Carlson, Pilar Fernandez, and Stephanie Seifer

Introduction

The following code reproduces the analyses from <. . . >, pertaining to the host prediction model. The code is subdivided into five parts: "1. Data Preparation", "2. Phylogenetic Analysis", "3. Boosted Regression Trees", "4. BRT Figures", "5. Mapping Host Distributions".

To reproduce the analyses pertaining to the link prediction model, please see the markdown file LinkPrediction_Code.Rmd located in the PoxHost repository on GitHub: https://github.com/viralemergence/PoxHost.

To run the following script, three files are required in your working directory: (1) Data_raw.RData, the raw data file; (2) "Output" folder, where all output will be saved - e.g., cleaned datasets, model results, figures, and tables; (3) MAMMALS.shp, the shape file of mammal geographical range obtained from IUCN Red List Spatial Database https://www.iucnredlist.org/resources/spatial-data-download. This file (>1GB) is only required in the last section of the code ("5. Mapping host distribution") and should be downloaded to your working directory before proceeding with part five.

Before proceeding, we recommend setting knit options and your working directory

1. Data Preparation

Load required packages and set system

```
#(1) Libraries for preparing data for analysis
library(ape)
library(dplyr)
library(nlme)
library(tidyverse)
library(stringr)
library(vroom)
## treespace dependencies include XQuartz v2.7.11 (https://www.xquartz.org/releases/XQuartz-2.7.11.html
library(rgl) # >install.packages("rgl"); >options(rgl.useNULL=TRUE)
library(treespace)

#(2) Clean environment
rm(list=ls())
graphics.off()
```

Load raw data

```
#(1) Load raw data
load("Data_raw.RData")
# load(url('https://github.com/viralemergence/PoxHost/blob/fbc89ca541c580e8eb7f521d4e0cd5f1198a3d9f/Tse

#(2) Pox data: host-OPV interactions detected via PCR/isolation from Virion database
##virion <- vroom('https://github.com/viralemergence/virion/blob/main/Virion/Virion.csv.gz')
poxdata <- virion %>% filter(VirusGenus == "orthopoxvirus" & (DetectionMethod %in% c("PCR/Sequencing","

#(3) Taxa: mammal species taxonomy from vertlife
##vertlife <- read.csv(url('https://data.vertlife.org/mammaltree/taxonomy_mamPhy_5911species.csv'))
taxa <- vertlife

#(4) Host traits: mammal traits from the COMBINE database <https://doi.org/10.1002/ecy.3344>
##path: ecy3344-sup-0001-datas1.zip > COMBINE_archives > trait_data_imputed.csv)
hostTraits <- combine

#(5) Host tree: mammal phylogeny tree from Dryad, <https://doi.org/10.5061/dryad.tb03d03>
##path: Data_S8_finalFigureFiles > _DATA > MamPhy_fullPosterior_BDvr_Completed_5911sp_topoCons_NDexp_MC
hostTree <- dryad

#(6) Clean environment
rm(virion, vertlife, dryad, combine)
```

Aggregate poxdata to genus-level

```
#(1) Exclude if host genus or virus is NA; Exclude variola (smallpox) virus
poxdata <- poxdata[!is.na(poxdata$HostGenus),]
poxdata <- poxdata[!is.na(poxdata$Virus),]
poxdata <- poxdata[!(poxdata$Virus=="variola virus"),]

#(2) Extract PCR-positive data
pcr <- subset(poxdata[which(poxdata$DetectionMethod=="PCR/Sequencing"),], select=c("Host","HostGenus","
pcr$Host <- ifelse(is.na(pcr$Host),"sp.",pcr$Host)
pcr$pcr <- 1
pcr <- aggregate(.~Host+HostGenus+Virus, data=pcr, sum)

#(3) Extract competence-positive data
competence <- subset(poxdata[which(poxdata$DetectionMethod=="Isolation/Observation"),], select=c("Host"
competence$Host <- ifelse(is.na(competence$Host),"sp.",competence$Host)
competence$competence <- 1
competence <- aggregate(.~Host+HostGenus+Virus, data=competence, sum)

#(4) Merge PCR and competence data
poxdata <- merge(pcr, competence, by=c("Host","HostGenus","Virus"), all=TRUE)

#(5) Create studies variable
poxdata$studies <- ifelse(is.na(poxdata$pcr),0,poxdata$pcr) + ifelse(is.na(poxdata$competence),0,poxdata

#(6) Create binary variables for detection via PCR and competence
```

```
poxdata$pcr=ifelse(is.na(poxdata$pcr),0,1)
poxdata$competence=ifelse(is.na(poxdata$competence),0,1)

#(7) Aggregate by genus and virus
agg_pcr <- aggregate(pcr~HostGenus+Virus, data=poxdata, max)
agg_competence <- aggregate(competence~HostGenus+Virus, data=poxdata, max)
agg_studies <- aggregate(studies~HostGenus+Virus, data=poxdata, sum)

#(8) Merge PCR, competence and studies variables
poxdata <- merge(agg_pcr,agg_competence)
poxdata <- merge(poxdata,agg_studies)

#(9) Rename variables
poxdata <- plyr::rename(poxdata,c('HostGenus'='gen','Virus'='virus'))
poxdata$gen <- str_to_title(poxdata$gen)

#(10) Clean environment
rm(pcr,competence,agg_pcr, agg_competence, agg_studies)
```

Merge poxdata with broader mammal taxa to create pseudoabsences

```
#(1) Drop duplicate genera in taxa
gtaxa <- taxa[!duplicated(taxa$gen),]
gtaxa <- gtaxa[c('gen','fam','ord')]

#(2) Check for mismatched names; Then merge poxdata with taxa
poxdata$gen[!poxdata$gen %in% taxa$gen]
poxdata <- merge(gtaxa,poxdata,by='gen',all.x=TRUE)

#(3) Keep only genera from orders in which positive associations exist
keep <- subset(poxdata, pcr==1 | competence==1)
poxdata$keep <- ifelse(poxdata$ord %in% keep$ord,TRUE,FALSE)
poxdata <- subset(poxdata,keep==TRUE)
poxdata$keep=NULL

#(6) Create binary variable for sampled host-OPV pairs
poxdata$sampled=ifelse(is.na(poxdata$pcr) & is.na(poxdata$competence),0,1)

#(6) Calculate number of pseudoabsences
length(which(is.na(poxdata$virus)))

#(7) Reclassify NAs as pseudo-absences for viral detection
poxdata$pcr=ifelse(is.na(poxdata$pcr),0,poxdata$pcr)
poxdata$competence=ifelse(is.na(poxdata$competence),0,poxdata$competence)
poxdata$studies=ifelse(is.na(poxdata$studies),0,poxdata$studies)

#(8) Replace NA taxonomic values based on host genera
poxdata=merge(poxdata,gtaxa,by='gen',all.x=TRUE)
poxdata <- plyr::rename(poxdata,c('fam.y'='fam','ord.y'='ord'))
poxdata$fam.x=NULL
poxdata$ord.x=NULL
```

```
#(9) Clean environment
rm(taxa,gtaxa,keep)
```

Aggregate hostTraits to genus-level

```
#(1) Observe variable names
colnames(hostTraits)

#(2) To aggregate continuous/integer variables, use the median as the summary measure
hostTraits_continuous=aggregate(cbind(adult_mass_g,brain_mass_g,adult_body_length_mm,adult_forearm_leng
                                      max_longevity_d,maturity_d,female_maturity_d,male_maturity_d,
                                      age_first_reproduction_d,gestation_length_d,teat_number_n,
                                      litter_size_n,litters_per_year_n,interbirth_interval_d,
                                      neonate_mass_g,weaning_age_d,weaning_mass_g,generation_length_d,
                                      dispersal_km,density_n_km2,home_range_km2,social_group_n,
                                      dphy_invertebrate,dphy_vertebrate,dphy_plant,
                                      det_inv,det_vend,det_vect,det_vfish,det_vunk,det_scav,det_fruit,det_
                                      upper_elevation_m,lower_elevation_m,altitude_breadth_m,habitat_bread
                                ~ order+family+genus, data=hostTraits, FUN=median, na.action=na.pass, na.r
##'na.action=na.pass, na.rm=TRUE' is specified such that if species w/in a genus has a combination of r

#(3) To aggregate binary variables, use the mean as the summary measure
hostTraits$fossoriality[hostTraits$fossoriality==2]<-0  #recode 0/1
hostTraits_binary=aggregate(cbind(hibernation_torpor,fossoriality,freshwater,marine,terrestrial_non.vola
                                  island_dwelling,disected_by_mountains,glaciation) ~ order+family+genus,

#(4) To aggregate categorical variables, first transform the variables to binary
hostTraits_cat <- hostTraits
hostTraits_cat$trophic_herbivores <- ifelse(hostTraits_cat$trophic_level==1,1,0)
hostTraits_cat$trophic_omnivores <- ifelse(hostTraits_cat$trophic_level==2,1,0)
hostTraits_cat$trophic_carnivores <- ifelse(hostTraits_cat$trophic_level==3,1,0)
hostTraits_cat$activity_nocturnal <- ifelse(hostTraits_cat$activity_cycle==1,1,0)
hostTraits_cat$activity_crepuscular <- ifelse(hostTraits_cat$activity_cycle==2,1,0) #nocturnal/crepuscu
hostTraits_cat$activity_diurnal <- ifelse(hostTraits_cat$activity_cycle==3,1,0)
hostTraits_cat$forager_marine <- ifelse(hostTraits_cat$foraging_stratum=="M",1,0)
hostTraits_cat$forager_ground <- ifelse(hostTraits_cat$foraging_stratum=="G",1,0)
hostTraits_cat$forager_scansorial <- ifelse(hostTraits_cat$foraging_stratum=="S",1,0)
hostTraits_cat$forager_arboreal <- ifelse(hostTraits_cat$foraging_stratum=="Ar",1,0)
hostTraits_cat$forager_aerial <- ifelse(hostTraits_cat$foraging_stratum=="A",1,0)
hostTraits_cat$island_end_marine <- ifelse(hostTraits_cat$island_endemicity=="Exclusively marine",1,0)
hostTraits_cat$island_end_mainland <- ifelse(hostTraits_cat$island_endemicity=="Occurs on mainland",1,0
hostTraits_cat$island_end_lgbridge <- ifelse(hostTraits_cat$island_endemicity=="Occurs on large land br
##hostTraits_cat$island_end_smbridge <- ifelse(hostTraits_cat$island_endemicity=="Occurs on small land
hostTraits_cat$island_end_isolated <- ifelse(hostTraits_cat$island_endemicity=="Occurs only on isolated
hostTraits_cat$biogeo_afrotropical <- ifelse(grepl("Afrotropical",hostTraits_cat$biogeographical_realm)
hostTraits_cat$biogeo_antarctic <- ifelse(grepl("Antarctic",hostTraits_cat$biogeographical_realm),1,0)
hostTraits_cat$biogeo_australasian <- ifelse(grepl("Australasian",hostTraits_cat$biogeographical_realm)
hostTraits_cat$biogeo_indomalayan <- ifelse(grepl("Indomalayan",hostTraits_cat$biogeographical_realm),1
hostTraits_cat$biogeo_nearctic <- ifelse(grepl("Nearctic",hostTraits_cat$biogeographical_realm),1,0)
hostTraits_cat$biogeo_neotropical <- ifelse(grepl("Neotropical",hostTraits_cat$biogeographical_realm),1
hostTraits_cat$biogeo_oceanian <- ifelse(grepl("Oceanian",hostTraits_cat$biogeographical_realm),1,0)
```

```
hostTraits_cat$biogeo_palearctic <- ifelse(grepl("Palearctic",hostTraits_cat$biogeographical_realm),1,0)

#(5) To aggregate transformed categorical-to-binary variables, use the mean as the summary measure
hostTraits_cat=aggregate(cbind(trophic_herbivores,trophic_omnivores,trophic_carnivores,
                               activity_nocturnal,activity_crepuscular,activity_diurnal,
                               forager_marine,forager_ground,forager_scansorial,forager_arboreal,forager_a
                               island_end_marine,island_end_mainland,island_end_lgbridge,island_end_isolat
                               biogeo_afrotropical,biogeo_antarctic,biogeo_australasian,biogeo_indomalayan
                         ~ order+family+genus, data=hostTraits_cat, FUN=mean, na.action=na.pass, na.rm=TRU

#(6) Merge continuous variables with binary variables and simplify dataframe
hostTraits <- full_join(hostTraits_continuous, hostTraits_binary, by = c("order","family","genus"),keep=
hostTraits <- plyr::rename(hostTraits,c('order.x'='order','family.x'='family','genus.x'='genus'))
hostTraits=subset(hostTraits, select=-c(order.y,family.y,genus.y))

#(7) Merge transformed categorical variables and simplify dataframe
hostTraits <- full_join(hostTraits, hostTraits_cat, by = c("order","family","genus"),keep=TRUE)
hostTraits <- plyr::rename(hostTraits,c('order.x'='order','family.x'='family','genus.x'='genus'))
hostTraits <- subset(hostTraits, select=-c(order.y,family.y,genus.y))

#(8) Clean environment
rm(hostTraits_binary,hostTraits_cat,hostTraits_continuous)
```

Collapse hostTree to genus-level

```
#(1) Reformat
hostTree$tip.label[hostTree$tip.label=="_Anolis_carolinensis"] <- "Anolis_carolinensis"

#(2) Create dataframe linking tip labels with their corresponding categories (genus and species)
tdata <- data.frame(matrix(NA,nrow=length(hostTree$tip.label),ncol=0))
tdata$genus <- sapply(strsplit(hostTree$tip.label,'_'),function(x) paste(x[1],sep='_'))
tdata$species <- hostTree$tip.label

#(3) Collapse tree to genus level
hostTree <- makeCollapsedTree(tree=hostTree,df=tdata[c('genus','species')])

#(4) Clean environment
rm(tdata)
```

Check for mismatched genera names in poxdata, hostTraits and hostTree

```
#(1) Check if all poxdata genera are in hostTree
poxdata$gtip <- poxdata$gen
hostTree$gtip <- hostTree$tip.label
poxdata$intree <- ifelse(poxdata$gtip%in%setdiff(poxdata$gtip,hostTree$gtip),'missing','upham')

#(2) Check if all poxdata genera are in hostTraits
hostTraits$gtip <- hostTraits$genus
poxdata$intraits <- ifelse(poxdata$gtip%in%setdiff(poxdata$gtip,hostTraits$gtip),'missing','traits')
```

```
#(3) Create a dataframe of just the observations with mismatched names
fix <- poxdata[c('gtip','intree','intraits')]
fix <- fix[fix$intree=='missing'|fix$intraits=='missing',]
fix <- unique(fix)

#(4) For those with mismatched names, identify homotypic synonyms or proxy species via IUCN (https://www
fix$treename <- NA
fix$traitname <- NA
fix$proxy <- NA
fix$proxy <- ifelse(fix$gtip=="Calassomys","Delomys",fix$proxy)
  ##source: https://academic.oup.com/jmammal/article/95/2/201/860032
fix$traitname <- ifelse(fix$gtip=="Liomys","Heteromys",fix$traitname)
  ##source: https://www.iucnredlist.org/species/40768/22345036
fix$traitname <- ifelse(fix$gtip=="Oreonax","Lagothrix",fix$traitname)
  ##source: https://www.iucnredlist.org/species/39924/192307818
fix$traitname <- ifelse(fix$gtip=="Paralomys","Phyllotis",fix$traitname)
  ##source: https://www.iucnredlist.org/species/17226/22333354
fix$traitname <- ifelse(fix$gtip=="Pearsonomys","Geoxus",fix$traitname)
  ##source: https://www.iucnredlist.org/species/40768/22345036
fix$traitname <- ifelse(fix$gtip=="Pipanacoctomys","Tympanoctomys",fix$traitname)
  ##source: https://www.iucnredlist.org/species/136557/78324400#taxonomy
fix$traitname <- ifelse(fix$gtip=="Pseudalopex","Lycalopex",fix$traitname)
  ##source: https://www.iucnredlist.org/species/6926/87695615
## hostTraits$genus[which(grepl('Tympanoctomys',hostTraits$genus))]

#(5) Merge revised names with poxdata
fix <- subset(fix, select=-c(intree,intraits))
poxdata <- merge(poxdata,fix,by='gtip',all.x=T)

#(6) If 'treename' is missing, first relabel as NA, then relabel with 'gtip'
poxdata$treename <- ifelse(poxdata$treename=='',NA,as.character(poxdata$treename))
poxdata$treename <- ifelse(is.na(poxdata$treename),as.character(poxdata$gtip),as.character(poxdata$treen

#(7) If 'traitname' is missing, first relabel as NA; If 'traitname' is NA and missing in 'intraits', th
poxdata$traitname <- ifelse(poxdata$traitname=='',NA,as.character(poxdata$traitname))
poxdata$traitname <- ifelse(poxdata$intraits=='missing' & is.na(poxdata$traitname),as.character(poxdata$
                       ifelse(poxdata$intraits=='missing' & !is.na(poxdata$traitname),as.character(poxda
                              as.character(poxdata$gtip)))

#(8) Simplify and clean environment
poxdata <- subset(poxdata, select=-c(intree,intraits,proxy))
rm(fix)
```

Merge poxdata with hostTraits and trim hostTree to mirror poxdata

```
#(2) Merge traits with poxdata
hostTraits$traitname <- hostTraits$gtip
poxdata <- merge(poxdata,hostTraits,by=c('traitname'),all.x=T)

#(3) Clean up poxdata
poxdata <- plyr::rename(poxdata,c('gtip.x'='gtip'))
```

```
poxdata <- subset(poxdata,select=-c(order, family, genus, gtip.y))

#(4) Trim hostTree to mirror poxdata
hostTree <- keep.tip(hostTree,hostTree$tip.label[hostTree$tip.label%in%poxdata$treename])
hostTree$gtip <- NULL
hostTree=makeLabel(hostTree)

#(5) Clean environment
rm(hostTraits)
```

Add PubMed citations and evolutionary distinctiveness measure

```
#(1) Load library for PubMed citations
library(easyPubMed)

#(2) Create function to count citations
counter=function(name){
  as.numeric(as.character(get_pubmed_ids(gsub('_','-',name))$Count))
}
citations=c()

#(3) Extract unique genera from poxdata
treename <- unique(poxdata$treename)

#(4) Apply counter function while looping through treenames
for(i in 1:length(treename)) {
  citations[i]=counter(treename[i])
  print(i)
}

#(5) Compile citation numbers
cites <- data.frame(treename=treename,cites=citations)

#(6) Merge cites with poxdata
poxdata <- merge(poxdata,cites,by='treename')

#(7) Load library for evolutionary distinctiveness (ed) measure
library(picante)  #before loading picante, make sure latest version of nlme package is loaded
ed <- evol.distinct(hostTree,type='equal.splits') #calculates ed measures for a suite of species by equ

#(8) Rename variables in ed
ed <- plyr::rename(ed,c('Species'='treename','w'='ed_equal'))

#(9) Merge ed with poxdata
poxdata <- merge(poxdata,ed,by='treename')

#(10) Clean environment
rm(cites,ed,citations,i,treename,counter)
```

Aggregate to genus level

```
#(1) Remove virus variable
poxdata <- subset(poxdata,select=-c(virus))

#(2) Remove duplicate genera: aggregate to genus-level taking the max value of pcr/comp and the sum of
agg_pcr <- aggregate(pcr~gen, data=poxdata, max)
agg_competence <- aggregate(competence~gen, data=poxdata, max)
agg_studies <- aggregate(studies~gen, data=poxdata, sum)

#(3) Remove duplicate genera: merge pcr and competence data back in
poxdata$pcr=NULL
poxdata$competence=NULL
poxdata$studies=NULL
poxdata <- poxdata[!duplicated(poxdata$gen),]
poxdata <- list(poxdata,agg_pcr,agg_competence,agg_studies) %>% reduce(full_join, by='gen')

#(4) Reorder variables
poxdata <- poxdata %>%
  dplyr::relocate(gen,fam,ord,gtip,treename,traitname,pcr,competence,studies,sampled,cites,ed_equal)

#(4) Clean environment
rm(agg_competence,agg_pcr,agg_studies)
```

Save cleaned data

```
#(3) Save cleaned for analysis
#save(poxdata, hostTree, file="Output/HostData_clean.RData")
```

2. Phylogenetic Analysis

Load required packages and set system

```
#(1) Libraries for phylogenetic analysis
library(ape)
library(caper)
library(data.table)
library(BiocManager)  ## BiocManager::install(c("Biostrings","ggtree"))
library(phylofactor)  ## devtools::install_github('reptalex/phylofactor'); more info at: https://reptal
library(treeio)       ## BiocManager::install("treeio")
library(ggtree)

#(2) Clean environment
rm(list=ls())
graphics.off()

# #(3) Set working directory
setwd("~/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernandez Lab/Projects
```

Phylogenetic patterns

```
#(1) Load data and trim unnecessary columns
load("Output/HostData_clean.RData")
# load("/Users/katietseng/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernan
data <- poxdata

#(2) Check that genus name in poxdata is also in hostTree
which(data$treename%in%setdiff(data$treename,hostTree$tip.label))

#(3) Create variables label and Species (required in later functions)
data$label <- data$treename
data$Species <- data$treename

#(4) Merge phylogeny w/ data ensuring consistent structure & ordering (caper::comparative.data)
cdata=comparative.data(phy=hostTree,data=data,names.col=treename,vcv=T,na.omit=F,warn.dropped=T)
cdata$data$tree=NULL

#(5) What proportion of genera have evidence of infection?
nrow(data)
count(data$pcr==1)
round(prop.table(table(data$pcr)),4)*100
count(data$competence==1)
round(prop.table(table(data$competence)),4)*100
##values in each cell divided by the sum of the 4 cells

#(6) Does the raw data display a phylogenetic signal in response?
## D of 0 = Brownian model, D of 1 = random (no phylogenetic signal)
set.seed(1)
mod1 <- phylo.d(cdata,binvar=pcr,permut=10000); mod1
set.seed(1)
mod2 <- phylo.d(cdata,binvar=competence,permut=10000); mod2
```

Phylofactorization

```
#(1) Create dataframe of taxonomy
cdata$data$taxonomy=paste(cdata$data$ord,cdata$data$fam,cdata$data$gen,sep='; ')
taxonomy <- data.frame(cdata$data$taxonomy)
names(taxonomy) <- "taxonomy"
taxonomy$Species <- rownames(cdata$data)
taxonomy <- taxonomy[c("Species","taxonomy")]
taxonomy$taxonomy <- as.character(taxonomy$taxonomy)

#(2) Holm rejection procedure: pf=phylofactor and FWER=family-wise error rate (alpha .05)
HolmProcedure <- function(pf,FWER=0.05){
  ## get split variable
  cs=names(coef(pf$models[[1]]))[-1]
      ### returns names of model coefficients (var names) extracted by 'coef' in
      ### the 1st list element of 'pf$models' minus the 1st element among those
      ### names; double brackets access a list element
  split=ifelse(length(cs)>1,cs[3],cs[1])
```

```
      ### returns 3rd element in 'cs' if length of the number of elements in
      ### 'cs' >1; else returns 1st element

  ## obtain p values
  if (pf$models[[1]]$family$family%in%c('gaussian',"Gamma","quasipoisson")){
      ### if fam$fam of 1st list element of pf$models is in columns 'gaussian'...
    pvals <- sapply(pf$models,FUN=function(fit) summary(fit)$coefficients[split,'Pr(>|t|)'])
      ### then to each element of pf$models, apply summary function w/ argument
      ### 'fit' and assign output to 'pvals';
      ### specifically, we use 'summary(fit)' to call the output of 'pf$models',
      ### extracting the 'coefficients' section, whereby we index the column
      ### named 'Pr(>|t\)' and split the data in that column; see sample output
      ### of linear model of R for reference (https://feliperego.github.io/blog/2015/10/23/Interpreting
  } else {
    pvals <- sapply(pf$models,FUN=function(fit) summary(fit)$coefficients[split,'Pr(>|z|)'])
      ### else extract p-val based on z statistic
  }
  D <- length(pf$tree$tip.label)
      ### returns number of elements in pf$tree$tip.label

  ## this is the line for Holm's sequentially rejective cutoff, where HB = Target alpha / (n - rank + 1
  keepers <- pvals<=(FWER/(2*D-3 - 2*(0:(pf$nfactors-1))))
      ### returns TRUE/FALSE if p-values are <= to 0.05/(n-rank+1)

  if (!all(keepers)){
      ### if not all pvals were keepers (i.e., all items in keepers were true)...
    nfactors <- min(which(!keepers))-1
      ### then assign nfactors to minimum/earliest position of items in keepers that were false, minus
  } else {
    nfactors <- pf$nfactors
      ###:else, assign nfactors as the value of pf$nfactors
  }
  return(nfactors)
}

## get species in a clade
cladeget=function(pf,factor){
  ### creates function 'cladeget' w/ arguments 'pf' and 'factor'
  spp=pf$tree$tip.label[pf$groups[[factor]][[1]]]
    ### returns n'th element of the pf$tree$tip.label based on the value of
    ### the first component inside the n'th ('factor') component of  'pf$groups'
  return(spp)
}

#(3) Summarize pf object
pfsum=function(pf){

  ## get formula
  chars=as.character(pf$frmla.phylo)[-1]  ### returns pf$frmla.phylo minus 1st element

  ## response
  resp=chars[1]               ###returns 1st element of chars
```

```
## holm
hp=HolmProcedure(pf)

## save model
model=chars[2]

## set key
setkey(pf$Data,'Species')   ### creates key on sorted pf$Datacolumn 'Species'

## make data
dat=data.frame(pf$Data)

## make clade columns in data
for(i in 1:hp){

  dat[,paste0(resp,'_pf',i)]=ifelse(dat$Species%in%cladeget(pf,i),'factor','other')
  ### paste0 concatenates all elements w/o a separator

}

## make data frame to store taxa name, response, mean, and other
results=data.frame(matrix(ncol=6, nrow = hp))
colnames(results)=c('factor','taxa','tips','node',"clade",'other')

## set taxonomy
taxonomy=dat[c('Species','taxonomy')]
taxonomy$taxonomy=as.character(taxonomy$taxonomy)

## loop
for(i in 1:hp){

  ## get taxa
  tx=pf.taxa(pf,taxonomy,factor=i)$group1            #gets taxonomic order

  ## get tail
  tx=sapply(strsplit(tx,'; '),function(x) tail(x,1)) #gets tax family as list

  ## combine
  tx=paste(tx,collapse=', ')         #collapses tax family into single string

  # save
  results[i,'factor']=i         #returns index number in 'factor' column
  results[i,'taxa']=tx          #returns string element (tx) in 'taxa' column

  ## get node
  tips=cladeget(pf,i)
  node=ggtree::MRCA(pf$tree,tips)
  ### MRCA = finds Most Recent Common Ancestor among a vector of tips
  results[i,'tips']=length(tips)
  results[i,'node']=ifelse(is.null(node) & length(tips)==1,'species',
                           ifelse(is.null(node) & length(tips)!=1,NA,node))

  ## get means
```

```
    ms=(tapply(dat[,resp],dat[,paste0(resp,'_pf',i)],FUN=mean))
    ### tapply takes mean of '1 vs. 0' (dat[,resp]) by 'other'/'factor' type (dat[,paste...]

    ## add in
    results[i,'clade']=ms['factor']
    results[i,'other']=ms['other']

  }

  ## return
  return(list(set=dat,results=results))       #returns number of clades with significantly greater prop
}

#(4) Phylofactorization of infection data
set.seed(1)
pcr_pf=gpf(Data=cdata$data,tree=cdata$phy,
           frmla.phylo=pcr~phylo,
           family=binomial,algorithm='phylo',nfactors=10,min.group.size=5)

#(5) Summarize infection PF results
HolmProcedure(pcr_pf)
pcr_pf_results=pfsum(pcr_pf)$results

#(6) Phylofactorization of competence data
set.seed(1)
hc_pf=gpf(Data=cdata$data,tree=cdata$phy,
          frmla.phylo=competence~phylo,
          family=binomial,algorithm='phylo',nfactors=2,min.group.size=5)

#(7) Summarize competence PF results
HolmProcedure(hc_pf)
hc_pf_results=pfsum(hc_pf)$results
```

Plot results of phylofactorization

```
#(1) Save tree for plotting
cdata$data$infect=factor(cdata$data$pcr)
cdata$data$comp=factor(cdata$data$competence)
dtree=treeio::full_join(as.treedata(cdata$phy),cdata$data,by="label")

#(2) Fix palette
AlberPalettes <- c("YlGnBu","Reds","BuPu", "PiYG")
AlberColours <- sapply(AlberPalettes, function(a) RColorBrewer::brewer.pal(5, a)[4])
afun=function(x){
  a=AlberColours[1:x]
  return(a)
}

#(3) Make low and high, and set x max
pcols=afun(2)
plus=1
```

```
pplus=plus+1

#(4) Fix taxa font formatting
pcr_pf_results$taxa
pcr_pf_results$taxa[1]="Rodentia"
hc_pf_results$taxa
hc_pf_results$taxa[1]="italic(Felidae)"

#(5) Plot pcr infection w/ ggtree
pcr_gg=ggtree(dtree,size=0.25)+
  geom_tippoint(aes(colour=infect),shape=15)+
  scale_colour_manual(values=c("grey80","black"))+
  guides(colour="none")

#(6) Add clades to plot
for(i in 1:nrow(pcr_pf_results)){

  pcr_gg=pcr_gg+
    geom_hilight(node=pcr_pf_results$node[i],
                 alpha=0.25,
                 fill=ifelse(pcr_pf_results$clade>
                               pcr_pf_results$other,pcols[2],pcols[1])[i])+
    geom_cladelabel(node=pcr_pf_results$node[i],
                    label=pcr_pf_results$taxa[i],
                    offset=pplus,
                    hjust=0.75,
                    offset.text=pplus*2,
                    parse=T,
                    angle=90)
}
pcr_gg=pcr_gg

#(7) Plot competence
comp_gg=ggtree(dtree,size=0.25)+
  geom_tippoint(aes(colour=comp),shape=15)+
  scale_colour_manual(values=c("grey80","black"))+
  guides(colour=F)

#(8) Add clades to plot
for(i in 1:nrow(hc_pf_results)){

  comp_gg=comp_gg+
    geom_hilight(node=hc_pf_results$node[i],
                 alpha=0.25,
                 fill=ifelse(hc_pf_results$clade>
                               hc_pf_results$other,pcols[2],pcols[1])[i])+
    geom_cladelabel(node=hc_pf_results$node[i],
                    label=hc_pf_results$taxa[i],
                    offset=pplus,
                    hjust=0.75,
                    offset.text=pplus*2,
                    parse=T,
                    angle=90)
```

```
}
comp_gg=comp_gg

#(9) Print tree figures for infection and competence
library(ggpubr)
png("Output/Figure1.png",width=6,height=6,units="in",res=300)
ggarrange(pcr_gg,comp_gg,ncol=2,widths=c(1.2,1),
          labels=c("(a) RT-PCR","(b) virus isolation"),
          label.x=c(-0.1,-0.2),
          font.label=list(face="plain",size=12))
dev.off()
```

Additional phylofactorization models

```
#(1) Create log-transformed variable of pubmed cites
cdata$data$logcites=log1p(cdata$data$cites)

#(2) Model PCR with pubmed cites as weight variable
set.seed(1)
pcr_pf_pm=gpf(Data=cdata$data,tree=cdata$phy,
                frmla.phylo=pcr~phylo,
                weights=cdata$data$logcites,
                family=binomial,algorithm='phylo',nfactors=10,min.group.size=5)

#(3) Summarize
HolmProcedure(pcr_pf_pm)
pcr_pf_pm_results=pfsum(pcr_pf_pm)$results

#(4) Model competence with pubmed cites as weight variable
set.seed(1)
hc_pf_pm=gpf(Data=cdata$data,tree=cdata$phy,
               frmla.phylo=competence~phylo,
               weights=cdata$data$logcites,
               family=binomial,algorithm='phylo',nfactors=10,min.group.size=5)

#(5) Summarize
HolmProcedure(hc_pf_pm)
hc_pf_pm_results=pfsum(hc_pf_pm)$results

#(6) Model cites themselves (not log1pm-transformed)
set.seed(1)
pm_pf=gpf(Data=cdata$data,tree=cdata$phy,
            frmla.phylo=cites~phylo,
            family=poisson,algorithm='phylo',nfactors=10,min.group.size=5)
HolmProcedure(pm_pf)
pm_pf_results=pfsum(pm_pf)$results
```

3. Boosted Regression Trees

Load required packages and set system

```
#(1) Libraries for BRT model
library(gbm)
library(fastDummies)
library(rsample)
library(ROCR)
library(sciplot)
library(ggplot2)
library(pdp)
library(PresenceAbsence)
library(tidyr)
library(viridis)
library(caper)
library(phylofactor)
library(ggtree)
library(treeio)
library(caret)
library(InformationValue)
library(mgcv)

#(2) Clean environment
rm(list=ls())
graphics.off()

# #(3) Set working directory
# setwd("~/Tseng2022")
```

Create taxonomic variables as predictors for the model

```
#(1) Load data and clean environment
load("Output/HostData_clean.RData")
data <- poxdata
rm(poxdata)

#(2) Classify true negatives
data$type=ifelse(data$pcr==0 & data$competence==0,"true negative","other")

#(3) Which species is competent but no PCR record?
set=data
set$treename[set$pcr==0 & set$competence==1]

#(4) Tabulate PCR/infection and isolation
set$inf=ifelse(set$pcr==0,"PCR negative","PCR positive")
set$iso=ifelse(set$competence==0,"no isolation","isolation")
table(set$inf,set$iso)

#(5) Make binary variables for each taxonomic family; remove any duplicates
```

```
dums=dummy_cols(data["fam"])
dums=dums[!duplicated(dums$fam),]

#(6) Ensure all family vars are factor
for(i in 1:ncol(dums)){
  dums[,i]=factor(dums[,i])
}

#(7) Merge family taxa variables with dataset as predictors
data=merge(data,dums,by="fam",all.x=T)

#(8) Drop unnecessary columns and clean environment
data$traitname=NULL
rm(dums,set,ag_columns)
```

Assess variation and availability of data

Are there zero or near-zero variance predictors?

```
#(1) Mode function
mode.prop <- function(x) {
  ux <- unique(x[is.na(x)==FALSE])         # creates array of unique values
  tab <- tabulate(match(na.omit(x), ux))   # creates array of the frequency (number of times) a unique v
  max(tab)/length(x[is.na(x)==FALSE])      # max-frequency / number of elements in each column that are
}

#(2) Assess variation across columns (2 indicates columns)
vars=data.frame(apply(data,2,function(x) mode.prop(x)),
                apply(data,2,function(x) length(unique(x))))    # number of unique elements in each col

#(3) Get names
vars$variables=rownames(vars)
names(vars)=c("var","uniq","column")

# ## round values
# vars$var=round(vars$var,2)

#(4)Label variables "cut" if homogeneous (100%)
vars$keep=ifelse(vars$var<1,"keep","cut")
vars$keep=ifelse(vars$column%in%c('fam','virus','gen','pcr','competence','fam'),'keep',vars$keep) # ens
vars=vars[order(vars$keep),]

#(5) Trim (creates array of column names to cut and removes from df)
keeps=vars[-which(vars$keep=="cut"),]$column

#(6) Drop if no variation
data=data[keeps]
rm(keeps,vars)

#(7) Assess missing values
mval=data.frame(apply(data,2,function(x) length(x[!is.na(x)])/nrow(data))) # proportion of values that
```

```
#(8) Get names
mval$variables=rownames(mval)
names(mval)=c("comp","column")
#
# #(9) visualize distribution of NA
# png("Output/Figure S1.png", width=4,height=4,units="in",res=600)
# ggplot(mval[!mval$column%in%c("gen","treename","pcr","competence","tip.label","fam"),],
#        aes(comp))+
#   geom_histogram(bins=50)+
#   geom_vline(xintercept=0.70,linetype=2,size=0.5)+
#   theme_bw()+
#   theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
#   theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
#   theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))+
#   labs(y="frequency",
#        x="trait coverage across mammal species (genus)")+
#   scale_x_continuous(labels = scales::percent)
# dev.off()

#(10) Label variables "cut" if >30% values are NA
mval$keep=ifelse(mval$comp>=0.70,"keep","cut")
table(mval$keep)
mval=mval[order(mval$keep),]

#(11) Trim (creates array of column names to cut and removes from df)
keeps=mval[-which(mval$keep=="cut"),]$column

#(12) Drop if not well represented
data=data[keeps]
rm(keeps,mval)

#(14) Save list of covariates and their coverage as table S1
set <- subset(data,select=-c(gen,fam,ord,gtip,treename,type,studies,sampled))
ts1=data.frame(apply(set,2,function(x) length(x[!is.na(x)])/nrow(set)))

#(15) Rename and reorder columns
ts1$variables=rownames(ts1)
names(ts1)=c("coverage","feature")
rownames(ts1)=NULL
ts1=ts1[!ts1$feature%in%c("pcr","competence"),]
ts1 <- subset(ts1,select=c(feature,coverage))

#(16) Save Table S1 to results
write.csv(ts1, "Output/TableS1.csv")

#(17) Check that binary variables are numeric and not factor (with the exception of fam_* variables)
str(set)
```

Model tuning to asses model performance for each combination of tuning parameters

```
#(1) Hyperparameter tuning ifelse
#hok="ok"
```

```r
hok="notok"
if(hok!="ok"){

  ## hyperparameter grid
  hgrid=expand.grid(n.trees=5000,                                  #creates df from all combinations of fac
                    interaction.depth=c(2,3,4),
                    shrinkage=c(0.01,0.001,0.0005),
                    n.minobsinnode=4,
                    seed=seq(1,10,by=1))

  # fix trees
  hgrid$n.trees=ifelse(hgrid$shrinkage<0.001,hgrid$n.trees*3,hgrid$n.trees)

  ## trees, depth, shrink, min, prop
  hgrid$id=with(hgrid,paste(n.trees,interaction.depth,shrinkage,n.minobsinnode))   #creates var 'id' co

  ## sort by id then seed
  hgrid=hgrid[order(hgrid$id,hgrid$seed),]

  ## now add rows
  hgrid$row=1:nrow(hgrid)                                          #adds var 'row' based on row number in

  ## factor id
  hgrid$id2=factor(as.numeric(factor(hgrid$id)))                   #creates 9-level factor var 'id2'


  ## function to assess each hyperpar combination
  hfit=function(row,response){

    ## make new data
    ndata=set

    ## correct response
    ndata$response=ndata[response][,1]                             #creates var 'response'

    ## remove raw
    ndata$pcr=NULL
    ndata$competence=NULL

    ## use rsample to split
    set.seed(hgrid$seed[row])                                      #sets seed value of 1-10
    split=initial_split(ndata,prop=0.7,strata="response")          #creates single binary split of data i

    ## test and train
    dataTrain=training(split)
    dataTest=testing(split)

    ## yTest and yTrain
    yTrain=dataTrain$response                                      #create array of just response values
    yTest=dataTest$response

    ## BRT
    set.seed(1)
```

```
  gbmOut=gbm(response ~ . ,data=dataTrain,                    #y~x; gbmOut contains list of 29 eleme
             n.trees=hgrid$n.trees[row],                      #total number of trees to fit (number
             distribution="bernoulli",
             shrinkage=hgrid$shrinkage[row],                  #equiv to learning rate or step-size r
             interaction.depth=hgrid$interaction.depth[row],  #max depth of each tree (highest level
             n.minobsinnode=hgrid$n.minobsinnode[row],        #min. number of obs in terminal nodes
             cv.folds=5,class.stratify.cv=TRUE,               #no. of cross-val folds to perform; fo
             bag.fraction=0.5,train.fraction=1,               #fraction of training set obs randomly
             n.cores=5,                                       #no. of CPU cores to use
             verbose=F)
             # par.details=(gbmParallel(num_threads=5)),

  ## performance
  par(mfrow=c(1,1),mar=c(4,4,1,1))                            #sets graphical parameters such that s
  best.iter=gbm.perf(gbmOut,method="cv")                      #estimates optimal number of boosting

  ## predict with test data
  preds=predict(gbmOut,dataTest,n.trees=best.iter,type="response")  #number of trees based on the opt

  ## known
  result=dataTest$response

  # ##estimate threshold value for classification of predicted probability
  # #library(pROC)
  # analysis <- roc(result,preds)  #roc([actual values],[predicted values])
  # e <- cbind(analysis$thresholds,analysis$sensitivities+analysis$specificities) #pulls each array a
  #
  # ##optimum threshold value
  # opt_t <- subset(e,e[,2]==max(e[,2]))[,1] #subsets dataframe and returns the max (sens+spec) value
  # #threshold<-opt_t #set as threshold value
  # #threshold = 0.2

  ## sensitivity and specificity                             #e.g., test run produced sensitivity of
  sen=InformationValue::sensitivity(result,preds)              #calculates sensitivity (# of obs with
  spec=InformationValue::specificity(result,preds)             #calculates specificity (# of obs w/o

  ## AUC on train
  auc_train=gbm.roc.area(yTrain,predict(gbmOut,dataTrain,n.trees=best.iter,type="response"))   #compu

  ## AUC on test
  auc_test=gbm.roc.area(yTest,predict(gbmOut,dataTest,n.trees=best.iter,type="response"))

  ## print
  print(paste("hpar row ",row," done; test AUC is ",auc_test,sep=""))  #prints "hpar row [x] done; te

  ## save outputs
  return(list(best=best.iter,                    #saves optimal number of iterations, AUC on training
              trainAUC=auc_train,
              testAUC=auc_test,
              spec=spec,
              sen=sen,
              wrow=row))
}
```

```
  ## run the function for PCR
  hpars=lapply(1:nrow(hgrid),function(x) hfit(x,response="pcr"))

  ## get results
  hresults=data.frame(sapply(hpars,function(x) x$trainAUC),
                      sapply(hpars,function(x) x$testAUC),
                      sapply(hpars,function(x) x$spec),
                      sapply(hpars,function(x) x$sen),
                      sapply(hpars,function(x) x$wrow),
                      sapply(hpars,function(x) x$best))
  names(hresults)=c("trainAUC","testAUC",
                    "spec","sen","row","best")

  ## combine and save
  hsearch=merge(hresults,hgrid,by="row")

  ## save
  hsearch$type="PCR"

  ## rerun the function for competence
  hpars=lapply(1:nrow(hgrid),function(x) hfit(x,response="competence"))

  ## get results
  hresults=data.frame(sapply(hpars,function(x) x$trainAUC),
                      sapply(hpars,function(x) x$testAUC),
                      sapply(hpars,function(x) x$spec),
                      sapply(hpars,function(x) x$sen),
                      sapply(hpars,function(x) x$wrow),
                      sapply(hpars,function(x) x$best))
  names(hresults)=c("trainAUC","testAUC",
                    "spec","sen","row","best")

  ## combine and save
  csearch=merge(hresults,hgrid,by="row")

  ## assign data type
  csearch$type="competence"

  ## combine
  search=rbind.data.frame(csearch,hsearch)
  search$type=factor(search$type,levels=c("PCR","competence"))

  ## export
  write.csv(search,"Output/par_tuning_data_summary.csv")

}else{

  ## load
  search=read.csv("Output/par_tuning_data_summary.csv")

}
```

Model tuning results: Figure S2

```
#(1) Convert parameters to factor and relabel values
search$shrinkage=factor(search$shrinkage)
lvl=rev(sort(unique(search$shrinkage)))  #sorts unique shrinkage par from large to small
search$shrinkage=factor(search$shrinkage,levels=lvl); rm(lvl)  #applies as factor
search$interaction.depth=factor(search$interaction.depth)
search$type=plyr::revalue(search$type,    #replace specified values w/ new values
                          c("PCR"="RT-PCR",
                            "competence"="virus isolation"))

#(2) PCR beta regression for AUC
mod=gam(testAUC~interaction.depth*shrinkage,   #gen additive models (gam) w/ integrated smoothness esti
        data=search[search$type=="RT-PCR",],method="REML",family=betar)
anova(mod)

#(3) Competence beta regression for AUC
mod=gam(testAUC~interaction.depth*shrinkage,
        data=search[search$type=="virus isolation",],method="REML",family=betar)
anova(mod)

#(4) PCR beta regression for sensitivity
mod=gam(sen~interaction.depth*shrinkage,
        data=search[search$type=="RT-PCR",],method="REML",family=betar)
anova(mod)

#(5) Competence beta regression for sensitivity
mod=gam(sen~interaction.depth*shrinkage,
        data=search[search$type=="virus isolation",],method="REML",family=betar)
anova(mod)


#(6) PCR beta regression for specificity
mod=gam(spec~interaction.depth*shrinkage,
        data=search[search$type=="RT-PCR",],method="REML",family=betar)
anova(mod)

#(7) Competence beta regression for specificity
mod=gam(spec~interaction.depth*shrinkage,
        data=search[search$type=="virus isolation",],method="REML",family=betar)
anova(mod)

#(8) Recast from wide to long
search2=gather(search,measure,value,testAUC:sen)

#(9) Relabel values and convert to factor
search2$measure=plyr::revalue(search2$measure,
                              c("sen"="sensitivity",
                                "spec"="specificity",
                                "testAUC"="test AUC"))
search2$measure=factor(search2$measure,
                       levels=c("test AUC","sensitivity","specificity"))
```

```r
#(10) Visualize - Figure S2
png("Output/FigureS2.png",width=5,height=8,units="in",res=600)
set.seed(1)
ggplot(search2,aes(shrinkage,value,
                   colour=interaction.depth,fill=interaction.depth))+
  geom_boxplot(alpha=0.25)+
  geom_point(alpha=0.75,
             position = position_jitterdodge(dodge.width=0.75))+
  theme_bw()+
  theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
  theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
  theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))+
  facet_grid(measure~type,scales="free_y",switch="y")+
  theme(strip.placement="outside",
        strip.background=element_blank())+
  theme(axis.text=element_text(size=10),
        axis.title=element_text(size=12),
        strip.text=element_text(size=12))+
  theme(legend.position="top")+
  scale_color_brewer(palette="Pastel2")+
  scale_fill_brewer(palette="Pastel2")+
  guides(colour=guide_legend(title="interaction depth"),
         fill=guide_legend(title="interaction depth"))+
  labs(y=NULL,
       x="learning rate")+
  scale_y_continuous(n.breaks=4)
dev.off()

#(11) To determine optimal parameters for model training, subset tuning results by number of trees
search_nt5000 <- search[search$n.trees==5000,]
search_nt15000 <- search[search$n.trees==15000,]
search_nt5000_sh0.01 <- search_nt5000[search_nt5000$shrinkage==0.010,]  #subset models with shrinakge==

#(12) Plot best.iter by type (pcr/competence) to see max number of trees to include
search_nt5000 %>%
  ggplot( aes(x=best, fill=type)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080"))
search_nt15000 %>%
  ggplot( aes(x=best, fill=type)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080"))
search_nt5000_sh0.01 %>%
  ggplot( aes(x=best, fill=type)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080"))

#(13) Clean
rm(search,search2,hok,mod,search_nt5000,search_nt15000,search_nt5000_sh0.01)
```

BRT function for applying across multiple data partitions

```
#(1) BRT function to use different data partitions
brt_part=function(seed,response){

  ## make new data
  ndata=set

  ## correct response
  ndata$response=ndata[response][,1]

  ## remove raw
  ndata$pcr=NULL
  ndata$competence=NULL

  ## fix cites if response
  if(response=="cites"){

    ## plus 1 for 0
    ndata$cites=ifelse(ndata$cites==0,1,ndata$cites)

  }else{

    ndata=ndata

  }

  ## use rsample to split
  set.seed(seed)
  split=initial_split(ndata,prop=0.7,strata="response")

  ## test and train
  dataTrain=training(split)
  dataTest=testing(split)

  ## yTest and yTrain
  yTrain=dataTrain$response
  yTest=dataTest$response

  ## dist
  dist=ifelse(response=="cites","poisson","bernoulli")

  ## n.trees
  nt=ifelse(response=="cites",10000,
     ifelse(response=="pcr",4500,5000)) #see plots of best.iter

  ## BRT
  set.seed(1)
  gbmOut=gbm(response ~ . ,data=dataTrain,
             n.trees=nt,
             distribution=dist,
             shrinkage=0.01, #see plots of best.iter
             interaction.depth=3,
```

```
            n.minobsinnode=4,
            cv.folds=5,class.stratify.cv=TRUE,
            bag.fraction=0.5,train.fraction=1,
            n.cores=5,
            verbose=F)
           # par.details=(gbmParallel(num_threads=5)),

## performance
par(mfrow=c(1,1),mar=c(4,4,1,1))
best.iter=gbm.perf(gbmOut,method="cv")  #estimates optimal number of boosting iterations for a gbm ob

## predict with test data
preds=predict(gbmOut,dataTest,n.trees=best.iter,type="response")

## known
result=dataTest$response

## sensitivity and specificity
sen=InformationValue::sensitivity(result,preds)
spec=InformationValue::specificity(result,preds)

## AUC on train
auc_train=gbm.roc.area(yTrain,predict(gbmOut,dataTrain,n.trees=best.iter,type="response"))

## AUC on test
auc_test=gbm.roc.area(yTest,predict(gbmOut,dataTest,n.trees=best.iter,type="response"))

## skip if poisson
if(response=="cites"){

  perf=NA

}else{

  ## inner loop if yTest is all 0
  if(var(yTest)==0){

    perf=NA
  }else{

    ## ROC
    pr=prediction(preds,dataTest$response)
    perf=performance(pr,measure="tpr",x.measure="fpr")         #pr=prediction object; measure=perform
    perf=data.frame(perf@x.values,perf@y.values)
    names(perf)=c("fpr","tpr")

    ## add seed
    perf$seed=seed

  }
}

## relative importance
```

```
  bars=summary(gbmOut,n.trees=best.iter,plotit=F)
  bars$rel.inf=round(bars$rel.inf,2)

  ## predict with cites
  preds=predict(gbmOut,data,n.trees=best.iter,type="response")
  pred_data=data[c("gtip",'treename',"fam","ord","pcr","competence")]
  pred_data$pred=preds
  pred_data$type=response

  ## predict with mean cites
  pdata=data
  pdata$cites=mean(pdata$cites)
  pred_data$cpred=predict(gbmOut,pdata,n.trees=best.iter,type="response")

  ## sort
  pred_data=pred_data[order(pred_data$pred,decreasing=T),]

  ## print
  print(paste("BRT ",seed," done; test AUC = ",auc_test,sep=""))

  ## save outputs
  return(list(mod=gbmOut,
              best=best.iter,
              trainAUC=auc_train,
              testAUC=auc_test,
              spec=spec,
              sen=sen,
              roc=perf,
              rinf=bars,
              predict=pred_data,
              traindata=dataTrain,
              testdata=dataTest,
              seed=seed))
}
```

Apply BRT function across 100 partitions to generate ensemble

```
#(1) Apply across 100 splits each
# smax=101
smax=100
pcr_brts=lapply(1:smax,function(x) brt_part(seed=x,response="pcr"))
comp_brts=lapply(1:smax,function(x) brt_part(seed=x,response="competence"))

#(2) Run wos brts
pm_brts=lapply(1:(smax-1),function(x) brt_part(seed=x,response="cites"))

#(3) Save results to wd
save(pcr_brts,comp_brts,pm_brts,file="Output/HostData_results.RData")
```

4. BRT Figures

Load required packages and set system

```
#(1) Libraries for BRT figures
library(tidyr)
library(ggplot2)
library(sciplot)
library(fastDummies)
library(caper)
library(ape)
library(phylofactor)
library(treeio)
library(ggtree)
library(plotrix)
library(rstatix)
library(ggrepel)
library(ggpubr)
library(plyr)

#(2) Clean environment
rm(list=ls())
graphics.off()

#(3) Set working directory
setwd("~/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernandez Lab/Projects
```

Evaluate performance measures:

How accurately did infection and competence BRT models distinguish OPV positive and negative species?

```
#### If needed, increase vector memory in R environment and reboot R before proceeding (https://stackov

#(1) Load data
load("Output/HostData_results.RData")
# pcr_brts <- readRDS("/Users/katietseng/Fernandez Lab Dropbox/Katie Tseng/Mac/Desktop/PoxHost(copy)/da
# comp_brts <- readRDS("/Users/katietseng/Fernandez Lab Dropbox/Katie Tseng/Mac/Desktop/PoxHost(copy)/d
# pm_brts <- readRDS("/Users/katietseng/Fernandez Lab Dropbox/Katie Tseng/Mac/Desktop/PoxHost(copy)/dat

#(2) Index non-missing
pcr_keep=which(!is.na(sapply(pcr_brts,function(x) x$testAUC)))
comp_keep=which(!is.na(sapply(comp_brts,function(x) x$testAUC)))

#(3) All
keep=intersect(pcr_keep,comp_keep)

#(4) Trim
pcr_brts=pcr_brts[keep]
comp_brts=comp_brts[keep]

#(5) Get net AUC
```

```
mean(c(sapply(pcr_brts,function(x) x$testAUC),sapply(comp_brts,function(x) x$testAUC)))
se(c(sapply(pcr_brts,function(x) x$testAUC),sapply(comp_brts,function(x) x$testAUC)))

#(6) Get net sensitivity
mean(c(sapply(pcr_brts,function(x) x$sen),sapply(comp_brts,function(x) x$sen)))
se(c(sapply(pcr_brts,function(x) x$sen),sapply(comp_brts,function(x) x$sen)))

#(7) Get net specificity
mean(c(sapply(pcr_brts,function(x) x$spec),sapply(comp_brts,function(x) x$spec)))
se(c(sapply(pcr_brts,function(x) x$spec),sapply(comp_brts,function(x) x$spec)))

#(8) Get net AUC for cites
mean(sapply(pm_brts,function(x) x$testAUC))
se(sapply(pm_brts,function(x) x$testAUC))

#(9) Clean environment
rm(pm_brts)

#(10) Get independent AUC for PCR and Comp models
mean(sapply(pcr_brts,function(x) x$testAUC))
se(sapply(pcr_brts,function(x) x$testAUC))
mean(sapply(comp_brts,function(x) x$testAUC))
se(sapply(comp_brts,function(x) x$testAUC))

#(11) Get independent sensitivity for PCR and Comp models
mean(sapply(pcr_brts,function(x) x$sen))
se(sapply(pcr_brts,function(x) x$sen))
mean(sapply(comp_brts,function(x) x$sen))
se(sapply(comp_brts,function(x) x$sen))

#(11) Get independent specificity for PCR and Comp models
mean(sapply(pcr_brts,function(x) x$spec))
se(sapply(pcr_brts,function(x) x$spec))
mean(sapply(comp_brts,function(x) x$spec))
se(sapply(comp_brts,function(x) x$spec))
```

Compare performance between BRTs trained on infection vs. competence:

```
#(1) Function for extracting data, performing unpaired t-test and determining effect size via Cohen's d
tfun=function(measure){

  ## format data
  n=length(sapply(pcr_brts,function(x) x$testAUC))
  adata=data.frame(y=c(sapply(pcr_brts,function(x) x[measure][[1]]),
                       sapply(comp_brts,function(x) x[measure][[1]])),
                   response=c(rep('infection',n),rep('competence',n)),
                   seed=c(sapply(pcr_brts,function(x) x$seed),
                          sapply(comp_brts,function(x) x$seed)))
  rm(n)

  ## factor
```

```
  adata$response=factor(adata$response,levels=c('infection','competence'))

  ## make jitter position
  adata$x=as.numeric(factor(adata$response))
  set.seed(1)
  adata$xj=jitter(adata$x,0.5)

  ## fix response
  adata$response2=plyr::revalue(adata$response,c("infection"="RT-PCR",
                                                 "competence"="virus isolation"))

  ## t-test
  tsum=t.test(y~response,data=adata,
              alternative='two.sided',
              var.equal=F,paired=F)

  ## effect size
  csum=cohens_d(y~response,data=adata,paired=F,var.equal=F)

  ## return
  return(list(adata=adata,tsum=tsum,csum=csum))
}

#(2) Compare AUC w/ tfun function; extract t-stat & Cohen's d
adata=tfun("testAUC")
adata$tsum$statistic
adata$csum$effsize

#(4) Compare sensitivity w/ tfun function; extract t-stat & Cohen's d
sedata=tfun("sen")
sedata$tsum$statistic
sedata$csum$effsize

#(6) Compare specificity w/ tfun function; extract t-stat & Cohen's d
spdata=tfun("spec")
spdata$tsum$statistic
spdata$csum$effsize

#(7) Adjust p-values with Benjamini Hochberg correction method
ps=c(adata$tsum$p.value,
     sedata$tsum$p.value,
     spdata$tsum$p.value)
round(p.adjust(ps,method="BH"),4)    #"BH" (aka "fdr") = Benjamini & Hochberg (1995) method control the
```

###Generate boxplot of model performance: Figure S3 and Figure 2A

```
#(1) Aggregate dataset
data1=sedata$adata
data2=spdata$adata

#(2) Types
data1$type="sensitivity"
data2$type="specificity"
```

```
sdata=rbind.data.frame(data1,data2)
rm(data1,data2)

#(3) Figure S3 - boxplot of model performance for supplement
png("Output/FigureS3.png",width=4,height=5,units="in",res=300)
set.seed(1)
ggplot(sdata)+
  #geom_violin(aes(x=x,y=auc,group=x),trim=T,scale="count",width=0.5)+
  geom_boxplot(aes(x=x,y=y,group=x),width=0.25,alpha=0.25,outlier.alpha=0)+
  geom_point(aes(x=xj,y=y),size=1.5,alpha=0.5)+
  scale_x_continuous(breaks=c(1,2),
                     labels=levels(sdata$response2),
                     limits=c(0.5,2.5))+
  theme_bw()+
  facet_wrap(~type,scales="free_y",strip.position="left",ncol=1)+
  theme(strip.placement="outside",
        strip.background=element_blank())+
  labs(x="Response variable",
       y=NULL)+
  theme(axis.text.y=element_text(size=10),
        axis.text.x=element_text(size=12),
        axis.title=element_text(size=12),
        strip.text=element_text(size=12))+
  theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
  theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
  theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))+
  guides(colour="none")
dev.off()

#(4) Figure 2A - plot of AUC for main text
set.seed(1)
f2A=ggplot(adata$adata)+
  geom_boxplot(aes(x=x,y=y,group=x),width=0.25,alpha=0.25,outlier.alpha=0)+
  geom_point(aes(x=xj,y=y),size=1.5,alpha=0.5)+
  scale_x_continuous(breaks=c(1,2),
                     labels=levels(adata$adata$response2),
                     limits=c(0.5,2.5))+
  theme_bw()+
  labs(x="Response variable",
       y="Model performance (AUC)")+
  theme(axis.text=element_text(size=10),
        axis.text.x=element_text(size=12),
        axis.title=element_text(size=12))+
  theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
  theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
  theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))+
  guides(colour="none")
```

Identify relative feature importance

```
#(1) Relative importance for PCR
vinf=lapply(pcr_brts,function(x) x$rinf)
```

```
pcr_vinf=do.call(rbind,vinf)

#(2) Relative importance for competence
vinf=lapply(comp_brts,function(x) x$rinf)
comp_vinf=do.call(rbind,vinf)

#(3) Aggregate mean, SE, var for PCR
vdata_pcr=data.frame(aggregate(rel.inf~var,data=pcr_vinf,mean),
                     aggregate(rel.inf~var,data=pcr_vinf,se)["rel.inf"],
                     aggregate(rel.inf~var,data=pcr_vinf,var)["rel.inf"])
names(vdata_pcr)=c("var","rel.inf","rse","rvar")
vdata_pcr=vdata_pcr[order(vdata_pcr$rel.inf,decreasing=T),]

#(4) Aggregate mean, SE, var for competence
vdata_comp=data.frame(aggregate(rel.inf~var,data=comp_vinf,mean),
                      aggregate(rel.inf~var,data=comp_vinf,se)["rel.inf"],
                      aggregate(rel.inf~var,data=comp_vinf,var)["rel.inf"])
names(vdata_comp)=c("var","rel.inf","rse","rvar")
vdata_comp=vdata_comp[order(vdata_comp$rel.inf,decreasing=T),]

#(5) Compare mean var
mean(vdata_pcr$rvar)
mean(vdata_comp$rvar)

#(6) Compare variance in mean var
var(vdata_pcr$rel.inf)
var(vdata_comp$rel.inf)

#(7) Clean environment
rm(pcr_vinf,comp_vinf,vinf)
```

Rank features by relative importance: Table S5

```
#(1) Rank for pcr and competence
vdata_pcr$pcr_rank=1:nrow(vdata_pcr)
vdata_comp$comp_rank=1:nrow(vdata_comp)

#(2) Relative infuence
vdata_pcr$pcr_imp=vdata_pcr$rel.inf/100
vdata_comp$comp_imp=vdata_comp$rel.inf/100

#(3) Combine ranks
ranks=merge(vdata_pcr[c("var","pcr_rank","pcr_imp")],
            vdata_comp[c("var","comp_rank","comp_imp")],
            by="var")

#(4) Table S5 - Ranks
ts5=ranks
ts5$feature=ts5$var
ts5=ts5[c("feature","pcr_imp","comp_imp","pcr_rank","comp_rank")]
write.csv(ts5,"Output/TableS5.csv")
```

```
#(6) Extract list of top 10 variables
keep_pcr <- ranks$var[which(ranks$pcr_rank<=10)]
keep_comp <- ranks$var[which(ranks$comp_rank<=10)]

#(7) Subset df of relative importance values associated with top 10 vars for pcr
vinf=lapply(pcr_brts,function(x) x$rinf)
pcr_vinf=do.call(rbind,vinf)
pcr_vinf <- pcr_vinf[which(pcr_vinf$var%in%keep_pcr),]
pcr_vinf$rel.inf <- pcr_vinf$rel.inf/100

#() Boxplot of Relative Feature Importance - Infection
f1A <- ggplot(pcr_vinf) + ggtitle("(A) Host-only infection model") +
        geom_boxplot(aes(x=rel.inf, y=reorder(var,rel.inf), group=var), width=0.5, alpha=0.25) +
        theme_bw() +
        labs(x="Relative influence",
             y="Features") +
        theme(axis.text.y=element_text(size=10),
              axis.text.x=element_text(size=12),
              axis.title.x=element_text(size=12, margin=margin(t=10,r=0,b=0,l=0)),
              axis.title.y=element_text(size=12, margin=margin(t=0,r=10,b=0,l=0)),
              strip.text=element_text(size=12))

#(7) Subset df of relative importance values associated with top 10 vars for competence
vinf=lapply(comp_brts,function(x) x$rinf)
comp_vinf=do.call(rbind,vinf)
comp_vinf <- comp_vinf[which(comp_vinf$var%in%keep_pcr),]
comp_vinf$rel.inf <- comp_vinf$rel.inf/100

#() Boxplot of Relative Feature Importance - Competence
f1B <- ggplot(comp_vinf) + ggtitle("(B) Host-only competence model") +
        geom_boxplot(aes(x=rel.inf, y=reorder(var,rel.inf), group=var), width=0.5, alpha=0.25) +
        theme_bw() +
        labs(x="Relative influence",
             y="Features") +
        theme(axis.text.y=element_text(size=10),
              axis.text.x=element_text(size=12),
              axis.title.x=element_text(size=12, margin=margin(t=10,r=0,b=0,l=0)),
              axis.title.y=element_text(size=12, margin=margin(t=0,r=10,b=0,l=0)),
              strip.text=element_text(size=12))

#(9) Figure S2 - Combine Boxplots of Relative Feature Importance
png("Output/trait_rank_f1.png",width=10,height=10,units="in",res=300)
ggarrange(f1A,f1B,ncol=2,nrow=1,
          font.label=list(face="plain",size=12))
dev.off()

rm(f1A, f1B, vdata_comp,vdata_pcr)
```

Identify consistently important/unimportant host traits: Figure 2B

```
#(1) Correlate: Were the rankings of relative feature importance sig. correlated?
cor.test(ranks$pcr_rank,ranks$comp_rank,method="spearman")

#(2) Trim to non-zero and rerank
ranks2=ranks[-which(ranks$pcr_imp==0 & ranks$comp_imp==0),]
ranks2=ranks2[order(ranks2$pcr_imp,decreasing=T),]
ranks2$pcr_rank=1:nrow(ranks2)
ranks2=ranks2[order(ranks2$comp_imp,decreasing=T),]
ranks2$comp_rank=1:nrow(ranks2)

#(3) Correlate: Were the rankings still correlated after removing traits with zero/no relative importan
cor.test(ranks2$pcr_rank,ranks2$comp_rank,method="spearman")

#(4) Identify features with high residuals and plot
ranks2$resid=abs(resid(lm(comp_rank~pcr_rank,data=ranks2)))  # extract residuals from linear regression

#(5) Plot residuals
plot(ranks2$pcr_rank,ranks2$resid,
     ylab="Residuals",xlab="pcr_rank",
     main="comp_rank")

#(6) Flag if resid>10
#ranks2$select=ifelse(ranks2$resid>10,"yes","no")
ranks2$select=ifelse(ranks2$resid>18,"yes","no")
which(ranks2$resid>20) # returns 5 values

#(7) Flag if consistently low or consistently high
n=7
ranks2$select=ifelse(ranks2$comp_rank<n & ranks2$pcr_rank<n,"yes",ranks2$select)
ranks2$select=ifelse(ranks2$comp_rank%in%tail(1:nrow(ranks2),n) & ranks2$pcr_rank%in%tail(1:nrow(ranks2)

#(8) Flag if high or low ranks
# rnk=c(head(ranks2$comp_rank,n),tail(ranks2$comp_rank,n))
# ranks2$select=ifelse(ranks2$comp_rank%in%rnk,"yes",ranks2$select)

#(9) Just yes
rset=ranks2[ranks2$select=="yes",]

#(10) rset as ranks2
rset=ranks2
rset$var=ifelse(rset$select=="yes",rset$var,"")

#(11) Figure 2B - Which traits were consistently important or unimportant?
set.seed(1)
f2B=ggplot(ranks2,aes(pcr_rank,comp_rank))+
  #geom_label(data=rset,aes(label=var),size=2,fill=col,alpha=0.2)+
  geom_text_repel(data=rset,aes(label=var),
                  size=2,
                  force=4,
                  #nudge_y=-2,
                  #nudge_x=1,
```

```
                  direction="both",
                  segment.size=0.5,
                  segment.color="grey")+
  geom_point()+
  #scale_y_reverse(limits=c(max(c(ranks$comp_rank,ranks$pcr_rank))+3,0))+
  #scale_x_reverse(limits=c(max(c(ranks$comp_rank,ranks$pcr_rank))+3,0))+
  scale_y_reverse(limits=c(max(c(ranks2$comp_rank,ranks2$pcr_rank))+4,0))+
  scale_x_reverse(limits=c(max(c(ranks2$comp_rank,ranks2$pcr_rank))+4,0))+
  #geom_abline(slope=1,linetype=2,size=0.5)+
  theme_bw()+
  labs(x="Feature rank for RT-PCR",
       y="Feature rank for virus isolation")+
  theme(axis.text=element_text(size=10),
        axis.title=element_text(size=12))+
  theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
  theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
  theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))

#(12) Figure 2 - Combine Figure 2A and 2B
png("Output/Figure2.png",width=7,height=3.5,units="in",res=300)
ggarrange(f2A,f2B,ncol=2,widths=c(1,1),
          labels=c("(a)","(b)"),
          label.x=c(0.21,0.18),
          label.y=0.97,
          font.label=list(face="plain",size=12))
dev.off()
```

Determine effect directions of each feature on the predicted outcome - Figure S4

```
#(1) Partial dependence plots w/ pdp package
# detach("package:purrr", unload=TRUE)
library(pdp)  #partial dependence plots help visualize the relationship b/w a subset of features and th
library(gbm)

#(2) Create function for compiling across BRTs for a given predictor, all else equal
pdp_agg=function(mod,feature){

  ## just the plot function
  pdep=plot(mod$mod,feature,
            return.grid=T,
            n.trees=mod$best,
            plot=F,
            continuous.resolution=200,
            type="response")

  ## add seed
  pdep$seed=unique(mod$roc$seed)

  ## save predictor
  pdep$predictor=pdep[feature][,1]

  ## order
```

```
  pdep=pdep[order(pdep$predictor),]

  ## get rank
  pdep$rank=1:nrow(pdep)

  ## save yhat
  pdep$yhat=pdep$y

  ## return
  return(pdep)

}

#(3) Function to plot
pdp_plot=function(bmods,feature){

  ## pdp_agg
  agg=do.call(rbind,lapply(bmods,function(x) pdp_agg(x,feature)))

  ## get class of the feature
  cl=class(data[feature][,1])

  ## if else based on type
  if(cl%in%c("numeric","integer")){

    ## get element-wise means
    x=with(agg,tapply(predictor,rank,mean))
    y=with(agg,tapply(yhat,rank,mean))

    ## save as mean
    pmean=data.frame(predictor=x,yhat=y)

    ## get yrange
    yrange=range(agg$yhat,pmean$yhat,na.rm=T)

    ## get histogram
    hi=hist(data[feature][,1],breaks=30,plot=F)
    hi=with(hi,data.frame(breaks[1:(length(breaks)-1)],counts))
    names(hi)=c("mids","counts")

    ## ggplot it
    ggplot(agg,aes(predictor,yhat,group=seed))+

      ## add histogram
      geom_segment(data=hi,inherit.aes=F,
                   aes(x=mids,xend=mids,
                       y=yrange[1],yend=plotrix::rescale(counts,yrange)),
                   size=1,colour="grey",alpha=0.25)+

      ## add lines
      geom_line(linewidth=1,alpha=0.25,colour="grey")+

      ## add mean
```

```
    geom_line(data=pmean,linewidth=2,inherit.aes=F,
              aes(predictor,yhat))+

    ## theme
    theme_bw()+
    theme(axis.text=element_text(size=6),
          axis.title=element_text(size=7))+
    theme(axis.title.x=element_text(margin=margin(t=5,r=0,b=0,l=0)))+
    theme(axis.title.y=element_text(margin=margin(t=0,r=5,b=0,l=0)))+
    theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
    labs(x=feature,y="marginal effect")+
    scale_y_continuous(labels=scales::number_format(accuracy=0.01))

  ## end numeric
}else{ ## factor-based plot

  ## get element-wise means
  y=with(agg,tapply(yhat,predictor,mean))

  ## save as mean
  #pmean=data.frame(predictor=x,yhat=y)
  pmean=data.frame(y)
  names(pmean)="yhat"
  pmean$predictor=rownames(pmean)
  rownames(pmean)=NULL

  ## make temp data
  temp=data
  temp$predictor=temp[feature][,1]

  ## do nothing
  agg=agg
  pmean=pmean
  temp=temp

  ## get yrange
  yrange=range(agg$yhat,pmean$yhat,na.rm=T)

  ## fix temp to yrange
  temp$yhat=ifelse(temp$pcr==1,max(yrange),min(yrange))

  ## ggplot with rug
  set.seed(1)
  ggplot(agg,aes(predictor,yhat,group=seed))+

    ## add individual BRTs
    geom_jitter(size=1,alpha=0.25,colour="grey",width=0.1)+

    ## add mean
    geom_point(data=pmean,size=2,inherit.aes=F,shape=15,
               aes(predictor,yhat))+

    ## add rug
```

```
      geom_rug(data=temp,inherit.aes=F,
               aes(predictor,yhat),
               sides="b",position="jitter",
               colour="grey",alpha=0.25,
               na.rm=T)+

      ## theme
      theme_bw()+
      theme(axis.text=element_text(size=6),
            axis.title=element_text(size=7))+
      theme(axis.title.x=element_text(margin=margin(t=5,r=0,b=0,l=0)))+
      theme(axis.title.y=element_text(margin=margin(t=0,r=5,b=0,l=0)))+
      theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
      labs(x=feature,y="marginal effect")+
      scale_y_continuous(limits=c(yrange[1]-0.01,yrange[2]+0.01),
                         labels=scales::number_format(accuracy=0.01))

  }

}

#(4) Load cleaned data file
load("Output/HostData_clean.RData")
# load("/Users/katietseng/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernan

data <- poxdata

#(5) Make binary columns for family
dums=fastDummies::dummy_cols(data["fam"])

#(6) Unique
dums=dums[!duplicated(dums$fam),]

#(7) Ensure all factor
for(i in 1:ncol(dums)){

  ## column as factor
  dums[,i]=factor(dums[,i])

}

#(8) Merge
data=merge(data,dums,by="fam",all.x=T)
rm(dums)

#(9) Top PCR
ranks2=ranks2[order(ranks2$pcr_rank),]
p1=pdp_plot(pcr_brts,ranks2$var[1])
p2=pdp_plot(pcr_brts,ranks2$var[2])
p3=pdp_plot(pcr_brts,ranks2$var[3])
p4=pdp_plot(pcr_brts,ranks2$var[4])
p5=pdp_plot(pcr_brts,ranks2$var[5])
p6=pdp_plot(pcr_brts,ranks2$var[6])
```

```
p7=pdp_plot(pcr_brts,ranks2$var[7])
p8=pdp_plot(pcr_brts,ranks2$var[8])
p9=pdp_plot(pcr_brts,ranks2$var[9])
p10=pdp_plot(pcr_brts,ranks2$var[10])

#(10) Top competence
ranks2=ranks2[order(ranks2$comp_rank),]
c1=pdp_plot(comp_brts,ranks2$var[1])
c2=pdp_plot(comp_brts,ranks2$var[2])
c3=pdp_plot(comp_brts,ranks2$var[3])
c4=pdp_plot(comp_brts,ranks2$var[4])
c5=pdp_plot(comp_brts,ranks2$var[5])
c6=pdp_plot(comp_brts,ranks2$var[6])
c7=pdp_plot(comp_brts,ranks2$var[7])
c8=pdp_plot(comp_brts,ranks2$var[8])
c9=pdp_plot(comp_brts,ranks2$var[9])
c10=pdp_plot(comp_brts,ranks2$var[10])

#(11) Figure S4 - Compile top ranked features by PCR and competence models
library(patchwork)
pcr_pdp_plots <- p1+p2+p3+p4+p5+p6+p7+p8+p9+p10+plot_layout(nrow=10,ncol=1,byrow=F)
comp_pdp_plots <- c1+c2+c3+c4+c5+c6+c7+c8+c9+c10+plot_layout(nrow=10,ncol=1,byrow=F)
png("Output/FigureS4.png",width=4,height=10,units="in",res=300)
ggarrange(pcr_pdp_plots,comp_pdp_plots,ncol=2,nrow=2,widths=c(4,4),heights=c(22,1),
          labels=c("(A) Infection","(B) Competence"),
          label.x=c(0,-0.1), label.y=0.001,
          font.label=list(face="plain",size=12))
dev.off()

#(12) Clean environment
rm(pcr_pdp_plots,comp_pdp_plots,p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,
   c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,
   ranks,ranks2,ts5,f2A,f2B,adata)
```

Model predictions: Extract and save predicted probabilities

```
#(1) Average predictions: PCR
pcr_apreds=lapply(pcr_brts,function(x) x$predict)
pcr_apreds=do.call(rbind,pcr_apreds)

#(2) Aggregate
pcr_apreds=data.frame(aggregate(pred~treename,data=pcr_apreds,mean),
                      aggregate(cpred~treename,data=pcr_apreds,mean)['cpred'], ## holding wos constant
                      aggregate(pcr~treename,data=pcr_apreds,prod)["pcr"],
                      aggregate(competence~treename,data=pcr_apreds,prod)["competence"])

#(3) Generate type variable
pcr_apreds$type='PCR'

#(4) Average predictions: competence
comp_apreds=lapply(comp_brts,function(x) x$predict)
comp_apreds=do.call(rbind,comp_apreds)
```

```r
#(5) Aggregate
comp_apreds=data.frame(aggregate(pred~treename,data=comp_apreds,mean),
                       aggregate(cpred~treename,data=comp_apreds,mean)['cpred'], ## holding wos constan
                       aggregate(pcr~treename,data=comp_apreds,prod)["pcr"],
                       aggregate(competence~treename,data=comp_apreds,prod)["competence"])

#(6) Generate type variable
comp_apreds$type='competence'

#(7) Combine apreds
apreds=rbind.data.frame(pcr_apreds,comp_apreds)

#(8) Add study
apreds=merge(apreds,data[c("treename","studies")],by="treename")

#(9) Generate positivity variable
apreds$positivity=ifelse(apreds$pcr==1 & apreds$type=="PCR",1,
                         ifelse(apreds$competence==1 & apreds$type=='competence',1,0))

#(10) Generate cat variable from studied
apreds$cat=ifelse(apreds$studies==0,"unsampled",
                  ifelse(apreds$positivity==1,"positive","negative"))

#(11) Generate type variable
library(plyr)
apreds$type=factor(apreds$type,levels=c("PCR","competence"))
apreds$type2=revalue(apreds$type,c("PCR"="infection"))

#(12) Transform apreds long to wide
apreds2=tidyr::spread(apreds[c('treename','type','cpred')],type,cpred)
comp_apreds$comp=comp_apreds$competence

#(13) Merge with comp_apreds
apreds2=merge(apreds2,comp_apreds[c("treename","pcr","comp")],by="treename")

#(14) Fix names
names(apreds2)=c("treename","pred_pcr","pred_comp","PCR","competence")

#(15) Classify true negatives
data$type=ifelse(data$studies>0 & data$pcr==0 & data$competence==0,"true negative","other")

#(16) Merge with data
apreds2=merge(apreds2,data[c("treename","type","studies","ord","fam","gen")],by='treename')

#(17) Fix type
apreds2$cat=ifelse(apreds2$studies==0,"unsampled",
                   ifelse(apreds2$PCR==0 & apreds2$competence==0,"negative","positive"))

#(18) Fix cat
apreds2$cat=factor(apreds2$cat,c("positive",'negative','unsampled'))
apreds$cat=factor(apreds$cat,levels=levels(apreds2$cat))

#(19) Fix type2
```

```
apreds$type2=revalue(apreds$type2,
                     c("infection"="Infection",
                       "competence"="Competence"))

#(20) Save
preds=apreds2
preds$fam=NULL
preds$gen=NULL

#(21) Write file
write.csv(preds,"Output/PoxHost_predictions.csv")
```

Model predictions: Figures of predicted probabilities distribution - Figure 3

```
#(1) Figure 3a - Density plot of predicted probabilities
remotes::install_github("awhstin/awtools")
library(awtools)
cc=mpalette[2:4]
cc=rev(cc)
f3A=ggplot(apreds,aes(cpred))+
  geom_density(aes(fill=cat,colour=cat),alpha=0.5)+
  facet_wrap(~type2,ncol=1,strip.position='top',scales="free_y")+
  theme_bw()+
  theme(legend.position="top")+
  labs(x=expression(paste("Predicted probability (",italic(P),") of hosting")))+
  xlim(0,1)+
  theme(axis.text=element_text(size=10),
        axis.title=element_text(size=12),
        legend.title=element_text(size=12),
        legend.text=element_text(size=11),
        strip.text=element_text(size=11),
        legend.margin=margin(0,0,0,0),
        legend.box.margin=margin(20,20,20,20))+
  theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
  theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
  theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))+
  scale_colour_manual(values=cc)+
  scale_fill_manual(values=cc)+
  guides(colour=guide_legend(title="(a) Orthopoxvirus positivity"),
         fill=guide_legend(title="(a) Orthopoxvirus positivity"))
f3A

#(2) Figure 3b - Scatterplot of predicted probabilities
f3B=ggplot(apreds2,aes(pred_pcr,pred_comp))+
  geom_point(alpha=0.5,size=2,aes(colour=cat,fill=cat))+
  geom_smooth(method='gam',colour="grey")+
  labs(x=expression(paste(italic(P),' from RT-PCR models')),
       y=expression(paste(italic(P),' from virus isolation models')))+
  theme_bw()+
  theme(axis.text=element_text(size=10),
        axis.title=element_text(size=12),
```

```
        legend.title=element_text(size=12),
        legend.text=element_text(size=11),
        strip.text=element_text(size=11),
        legend.margin=margin(0,0,0,0),
        legend.box.margin=margin(20,20,20,20))+
  theme(legend.position="top")+
  theme(panel.grid.major=element_blank(),panel.grid.minor=element_blank())+
  theme(axis.title.x=element_text(margin=margin(t=10,r=0,b=0,l=0)))+
  theme(axis.title.y=element_text(margin=margin(t=0,r=10,b=0,l=0)))+
  scale_colour_manual(values=cc)+
  scale_fill_manual(values=cc)+
  guides(colour=guide_legend(title="(a) Orthohantavirus positivity"),
         fill=guide_legend(title="(a) Orthohantavirus positivity"))
f3B

#(3) Figure 3 - Combine figure 3a and 3b
png("Output/Figure3.png",width=6.5,height=4,units="in",res=300)
f3=ggarrange(f3A,f3B,common.legend=T)
f3
dev.off()
```

Threshold the results

```
#(1) load libraries
library(PresenceAbsence)
library(openxlsx)

#(2) load files
set.seed(12345)
pred <- read.csv("/Users/katietseng/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.e

### Calculate threshold values for PCR Model ###
ts_pcr <- optimal.thresholds(data.frame(pred[,c('treename','PCR','pred_pcr')]),
                             threshold = 10001,
                             opt.methods = c(2,3,4,5,10),
                             req.sens = 0.80,
                             na.rm = TRUE)

ts_pcr[nrow(ts_pcr) + 1,] <- optimal.thresholds(data.frame(pred[,c('treename','PCR','pred_pcr')]),
                             threshold = 10001,
                             opt.methods = c(10),
                             req.sens = 0.85,
                             na.rm = TRUE)

ts_pcr[nrow(ts_pcr) + 1,] <- optimal.thresholds(data.frame(pred[,c('treename','PCR','pred_pcr')]),
                             threshold = 10001,
                             opt.methods = c(10),
                             req.sens = 0.9,
                             na.rm = TRUE)

ts_pcr[nrow(ts_pcr) + 1,] <- optimal.thresholds(data.frame(pred[,c('treename','PCR','pred_pcr')]),
                             threshold = 10001,
```

```
                            opt.methods = c(10),
                            req.sens = 0.95,
                            na.rm = TRUE)

cut_pcr1 <- function(x) {sum(pred$pred_pcr > x)}
cut_pcr2 <- function(x) {sum(pred$pred_pcr[pred$PCR==0] > x)}

sapply(unlist(ts_pcr[2]), cut_pcr1)
sapply(unlist(ts_pcr[2]), cut_pcr2)
sum(pred$PCR)
#save threshold values
# write.csv(ts_pcr, file='Output/ts_pcr.csv')

### Calculate threshold values for COMPETENCE MODEL ###

ts_comp <- optimal.thresholds(data.frame(pred[,c('treename','competence','pred_comp')]),
                            threshold = 10001,
                            opt.methods = c(2,3,4,5,10),
                            req.sens = 0.80,
                            na.rm = TRUE)

ts_comp[nrow(ts_comp) + 1,] <- optimal.thresholds(data.frame(pred[,c('treename','competence','pred_comp
                            threshold = 10001,
                            opt.methods = c(10),
                            req.sens = 0.85,
                            na.rm = TRUE)

ts_comp[nrow(ts_comp) + 1,] <- optimal.thresholds(data.frame(pred[,c('treename','competence','pred_comp
                            threshold = 10001,
                            opt.methods = c(10),
                            req.sens = 0.9,
                            na.rm = TRUE)

ts_comp[nrow(ts_comp) + 1,] <- optimal.thresholds(data.frame(pred[,c('treename','competence','pred_comp
                            threshold = 10001,
                            opt.methods = c(10),
                            req.sens = 0.95,
                            na.rm = TRUE)

cut_comp1 <- function(x) {sum(pred$pred_comp > x)}
cut_comp2 <- function(x) {sum(pred$pred_comp[pred$competence==0] > x)}

sapply(unlist(ts_comp[2]), cut_comp1)
sapply(unlist(ts_comp[2]), cut_comp2)
sum(pred$competence)
#save threshold values
# write.csv(ts_comp, file='Output/ts_comp.csv')

### APPLY THRESHOLDS TO PREDICTIONS ###

# Extract selected optimum threshold values from ts_pcr
ts_pcr_rs0.8 <- as.data.frame(ts_pcr[5,])
ts_pcr_rs0.9 <- as.data.frame(ts_pcr[7,])
```

```
ts_pcr_rs0.95 <- as.data.frame(ts_pcr[8,])
ts_pcr_mss3 <- as.data.frame(ts_pcr[2,])

ts_comp_rs0.8 <- as.data.frame(ts_comp[5,])
ts_comp_rs0.9 <- as.data.frame(ts_comp[7,])
ts_comp_rs0.95 <- as.data.frame(ts_comp[8,])
ts_comp_mss3 <- as.data.frame(ts_comp[2,])

#(5) Threshold the results to binary outputs
pred %>%
  mutate(bin_pcr_rs0.8 = ifelse(pred_pcr > ts_pcr_rs0.8$pred_pcr, 1, 0),
         bin_pcr_rs0.9 = ifelse(pred_pcr > ts_pcr_rs0.9$pred_pcr, 1, 0),
         bin_pcr_rs0.95 = ifelse(pred_pcr > ts_pcr_rs0.95$pred_pcr, 1, 0),
         bin_pcr_mss3 = ifelse(pred_pcr > ts_pcr_mss3$pred_pcr, 1, 0),
         bin_comp_rs0.8 = ifelse(pred_comp > ts_comp_rs0.8$pred_comp, 1, 0),
         bin_comp_rs0.9 = ifelse(pred_comp > ts_comp_rs0.9$pred_comp, 1, 0),
         bin_comp_rs0.95 = ifelse(pred_comp > ts_comp_rs0.95$pred_comp, 1, 0),
         bin_comp_mss3 = ifelse(pred_comp > ts_comp_mss3$pred_comp, 1, 0)) -> pred

#(9) Pull out the relevant lists
pred %>% filter(PCR==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> known_pcr #n=71
pred %>% filter(bin_pcr_rs0.8==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_pcr_rs0.8 #n=25
pred %>% filter(bin_pcr_rs0.9==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_pcr_rs0.9 #n=25
pred %>% filter(bin_pcr_rs0.95==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_pcr_rs0.95
pred %>% filter(bin_pcr_mss3==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_pcr_mss3 #n=250
pred %>% filter(competence==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> known_comp #n=58
pred %>% filter(bin_comp_rs0.8==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_comp_rs0.8 #n=
pred %>% filter(bin_comp_rs0.9==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_comp_rs0.9 #n=
pred %>% filter(bin_comp_rs0.95==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_comp_rs0.95
pred %>% filter(bin_comp_mss3==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred_comp_mss3 #n=15

#(5) Sort and create table of known and unknown hosts (hosts that do not overlap)
pred_kpcr <- as.data.frame(sort(known_pcr)) #n=58
pred_upcr_rs0.8 <- as.data.frame(sort(pred_pcr_rs0.8[!(pred_pcr_rs0.8 %in% known_pcr)])) #n=74
pred_upcr_rs0.9 <- as.data.frame(sort(pred_pcr_rs0.9[!(pred_pcr_rs0.9 %in% known_pcr)])) #n=74
pred_upcr_rs0.95 <- as.data.frame(sort(pred_pcr_rs0.95[!(pred_pcr_rs0.95 %in% known_pcr)]))
pred_upcr_mss3 <- as.data.frame(sort(pred_pcr_mss3[!(pred_pcr_mss3 %in% known_pcr)])) #n=88
pred_kcomp <- as.data.frame(sort(known_comp)) #n=41
pred_ucomp_rs0.8 <- as.data.frame(sort(pred_comp_rs0.8[!(pred_comp_rs0.8 %in% known_comp)])) #n=67
pred_ucomp_rs0.9 <- as.data.frame(sort(pred_comp_rs0.9[!(pred_comp_rs0.9 %in% known_comp)])) #n=67
pred_ucomp_rs0.95 <- as.data.frame(sort(pred_comp_rs0.95[!(pred_comp_rs0.95 %in% known_comp)]))
pred_ucomp_mss3 <- as.data.frame(sort(pred_comp_mss3[!(pred_comp_mss3 %in% known_comp)])) #n=152

#(6) How many predicted hosts are undiscovered by PCR?
nrow(pred_upcr_rs0.8)
nrow(pred_upcr_rs0.9)
nrow(pred_upcr_rs0.95)
nrow(pred_upcr_mss3)

#(7) How many predicted undiscovered hosts by competence
nrow(pred_ucomp_rs0.8)
nrow(pred_ucomp_rs0.9)
nrow(pred_ucomp_rs0.95)
```

```
nrow(pred_ucomp_mss3)

#(7) Create list of dataframes/tables
list_df <- list(pred_kpcr, pred_upcr_rs0.8, pred_upcr_rs0.9, pred_upcr_rs0.95, pred_upcr_mss3, pred_kcor

#(6) Rename variables
list_df <- lapply(list_df, function(x) {colnames(x) <- c("gen"); x})

#(11) Incorporate taxonomic family & order for each model
taxa <- apreds2[,c("gen","fam","ord")]
taxa <- unique(taxa)
list_df <- lapply(list_df, function(x) {x <- merge(x, taxa, by = "gen", all=FALSE); x})

#(12) Sort by virus, order, family, and genus
list_df <- lapply(list_df, function(x) {x <- x[order(x$ord, x$fam, x$gen),]; x})

#(13) Reorder columns
list_df <- lapply(list_df, function(x) {x <- x[,c("ord","fam","gen")]; x})

#(14) Reformat to proper
library(stringr)
list_df <- lapply(list_df, function(x) {x$fam <- str_to_title(x$fam); x})
list_df <- lapply(list_df, function(x) {x$ord <- str_to_title(x$ord); x})

#(15) Unlist
pred_kpcr <- as.data.frame(list_df[[1]])
pred_upcr_rs0.8 <- as.data.frame(list_df[[2]])
pred_upcr_rs0.9 <- as.data.frame(list_df[[3]])
pred_upcr_rs0.95 <- as.data.frame(list_df[[4]])
pred_upcr_mss3 <- as.data.frame(list_df[[5]])
pred_kcomp <- as.data.frame(list_df[[6]])
pred_ucomp_rs0.8 <- as.data.frame(list_df[[7]])
pred_ucomp_rs0.9 <- as.data.frame(list_df[[8]])
pred_ucomp_rs0.95 <- as.data.frame(list_df[[9]])
pred_ucomp_mss3 <- as.data.frame(list_df[[10]])

#(16) Save for known hosts and unknown hosts where req.sens=0.8, req.sens=0.85, req.sens=0.9, req.sens=
pred_pcr <- list('pcr_known'=pred_kpcr,
                 'pcr_unknown_rs0.8'=pred_upcr_rs0.8,
                 'pcr_unknown_rs0.9'=pred_upcr_rs0.9,
                 'pcr_unknown_rs0.95'=pred_upcr_rs0.95,
                 'pcr_unknown_mss3'=pred_upcr_mss3)
pred_comp <- list('comp_known'=pred_kcomp,
                  'comp_unknown_rs0.8'=pred_ucomp_rs0.8,
                  'comp_unknown_rs0.9'=pred_ucomp_rs0.9,
                  'comp_unknown_rs0.95'=pred_ucomp_rs0.95,
                  'comp_unknown_mss3'=pred_ucomp_mss3)
write.xlsx(pred_pcr, file='Output/Predicted_Hosts_PCR.xlsx')
write.xlsx(pred_comp, file='Output/Predicted_Hosts_Competence.xlsx')
```

Model predictions: Explore model correlation and phylogenetic signal

```
#(1) Test correlation between the predicted probabilities of infection vs. competence models
cor(apreds2$pred_pcr,apreds2$pred_comp,method='spearman') #computes Spearment correlation coefficient
cor.test(apreds2$pred_pcr,apreds2$pred_comp,method='spearman') #tests for correlation b/w paires sample

#(2) Load phylogeny
load("Output/HostData_clean.RData")
# load("/Users/katietseng/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernan
mtree=hostTree

#(3) Setdiff
apreds2$tree=ifelse(apreds2$treename%in%setdiff(apreds2$treename,mtree$tip.label),'cut','keep')
table(apreds2$tree)
## keep: 945

#(4) Trim and match
bdata=subset(apreds2,tree=='keep')
bdata=bdata[match(mtree$tip.label,bdata$treename),]

#(5) Save
bdata$label=bdata$treename
bdata$Species=bdata$treename

#()Generate mean of pcr and comp predicted probabilities
bdata <- bdata %>% mutate(pred_mean = rowMeans(across(pred_pcr:pred_comp), na.rm = TRUE))

#()Generate variable of whether a link was predicted based on threshold value
bdata$pcr_rs0.8 <- ifelse(bdata$pred_pcr > ts_pcr_rs0.8$pred_pcr, 1, 0)
bdata$comp_rs0.8 <- ifelse(bdata$pred_comp > ts_comp_rs0.8$pred_comp, 1, 0)
bdata$pcrcomp_rs0.8 <- ifelse(bdata$pcr_rs0.8==1 & bdata$comp_rs0.8==1, "Both",
                              ifelse(bdata$pcr_rs0.8==1 & bdata$comp_rs0.8==0, "PCR",
                                     ifelse(bdata$pcr_rs0.8==0 & bdata$comp_rs0.8==1, "Virus isolation"
bdata$pcr_mss3 <- ifelse(bdata$pred_pcr > ts_pcr_mss3$pred_pcr, 1, 0)
bdata$comp_mss3 <- ifelse(bdata$pred_comp > ts_comp_mss3$pred_comp, 1, 0)
bdata$pcrcomp_mss3 <- ifelse(bdata$pcr_mss3==1 & bdata$comp_mss3==1, "Both",
                              ifelse(bdata$pcr_mss3==1 & bdata$comp_mss3==0, "PCR",
                                     ifelse(bdata$pcr_mss3==0 & bdata$comp_mss3==1, "Virus isolation",

#Make factor
bdata$pcrcomp_rs0.8 <- factor(bdata$pcrcomp_rs0.8,levels=c("PCR","Virus isolation","Both","Not predicte
bdata$pcrcomp_mss3 <- factor(bdata$pcrcomp_mss3,levels=c("PCR","Virus isolation","Both","Not predicted"

#(6) Merge
cdata=comparative.data(phy=mtree,data=bdata,names.col=treename,vcv=T,na.omit=F,warn.dropped=T)    #vcv=
### Returned error indicating missing values: Error in '.rowNamesDF<- '(x, value = value) : missing val

#Identify which rows are NA
which(is.na(bdata))

#Subsetting only non-missing data
bdata_nonNA = bdata[-which(is.na(bdata)),]
```

```
#Try merging again
cdata=comparative.data(phy=mtree,data=bdata_nonNA,names.col=treename,vcv=T,na.omit=T,warn.dropped=T)

#(7) Fix
cdata$data$tree=NULL

#(8) Measure phylogenetic signal (Pagel's lambda) of model predictions
pcr_lmod=pgls(pred_pcr~1,data=cdata,lambda="ML")     #pgls fits a linear model while taking into accoun
comp_lmod=pgls(pred_comp~1,data=cdata,lambda="ML")   #lambda = value for lambda transformation; 'ML' us
###for more info: https://static1.squarespace.com/static/5459da8ae4b042d9849b7a7b/t/57ea64eae58c62718aa

#(9) Summarize
summary(pcr_lmod)
summary(comp_lmod)
```

Model predictions: Identify taxonomic patterns

```
#(1) Extract taxonomy
cdata$data$taxonomy=paste(cdata$data$ord,cdata$data$fam,cdata$data$Species,sep='; ') #We refer to genus

#(2) Set taxonomy
taxonomy=data.frame(cdata$data$taxonomy)
names(taxonomy)="taxonomy"
taxonomy$Species=rownames(cdata$data)
taxonomy=taxonomy[c("Species","taxonomy")]
taxonomy$taxonomy=as.character(taxonomy$taxonomy)

#(3) Holm rejection procedure (counteract the problem of multiple comparisons and controls FWER)
HolmProcedure <- function(pf,FWER=0.05){

  ## get split variable
  cs=names(coef(pf$models[[1]]))[-1]
  split=ifelse(length(cs)>1,cs[3],cs[1])

  ## obtain p values
  if (pf$models[[1]]$family$family%in%c('gaussian',"Gamma","quasipoisson")){
    pvals <- sapply(pf$models,FUN=function(fit) summary(fit)$coefficients[split,'Pr(>|t|)'])
  } else {
    pvals <- sapply(pf$models,FUN=function(fit) summary(fit)$coefficients[split,'Pr(>|z|)'])
  }
  D <- length(pf$tree$tip.label)

  ## this is the line for Holm's sequentially rejective cutoff
  keepers <- pvals<=(FWER/(2*D-3 - 2*(0:(pf$nfactors-1))))


  if (!all(keepers)){
    nfactors <- min(which(!keepers))-1
  } else {
    nfactors <- pf$nfactors
  }
  return(nfactors)
```

```
}

#(13) Get species in a clade
cladeget=function(pf,factor){
  spp=pf$tree$tip.label[pf$groups[[factor]][[1]]]
  return(spp)
}

#(4) Summarize pf object
pfsum=function(pf){

  ## get formula
  chars=as.character(pf$frmla.phylo)[-1]

  ## response
  resp=chars[1]

  ## fix
  resp=ifelse(resp=='cbind(pos, neg)','prevalence',resp)

  ## holm
  hp=HolmProcedure(pf)

  ## save model
  model=chars[2]

  ## set key
  setkey(pf$Data,'Species')

  ## make data
  dat=data.frame(pf$Data)

  ## make clade columns in data
  for(i in 1:hp){

    dat[,paste0(resp,'_pf',i)]=ifelse(dat$Species%in%cladeget(pf,i),'factor','other')

  }

  ## make data frame to store taxa name, response, mean, and other
  results=data.frame(matrix(ncol=6, nrow = hp))
  colnames(results)=c('factor','taxa','tips','node',"clade",'other')

  ## set taxonomy
  taxonomy=dat[c('Species','taxonomy')]
  taxonomy$taxonomy=as.character(taxonomy$taxonomy)

  ## loop
  for(i in 1:hp){

    ## get taxa
    tx=pf.taxa(pf,taxonomy,factor=i)$group1
```

```
    ## get tail
    tx=sapply(strsplit(tx,'; '),function(x) tail(x,1))

    ## combine
    tx=paste(tx,collapse=', ')

    # save
    results[i,'factor']=i
    results[i,'taxa']=tx

    ## get node
    tips=cladeget(pf,i)
    node=ggtree::MRCA(pf$tree,tips)
    results[i,'tips']=length(tips)
    results[i,'node']=ifelse(is.null(node) & length(tips)==1,'species',
                             ifelse(is.null(node) & length(tips)!=1,NA,node))

    ## get means
    ms=(tapply(dat[,resp],dat[,paste0(resp,'_pf',i)],mean))

    ## add in
    results[i,'clade']=ms['factor']
    results[i,'other']=ms['other']

  }

  ## return
  return(list(set=dat,results=results))
}

#(5) Fix palette
AlberPalettes <- c("YlGnBu","Reds","BuPu", "PiYG")
AlberColours <- sapply(AlberPalettes, function(a) RColorBrewer::brewer.pal(5, a)[4])
afun=function(x){
  a=AlberColours[1:x]
  return(a)
}

#(6) Make low and high
pcols=afun(2)

#(7) PCR predictions
set.seed(1)
pcrpred_pf=gpf(Data=cdata$data,tree=cdata$phy,
               frmla.phylo=pred_pcr~phylo,
               family=gaussian,algorithm='phylo',nfactors=10,min.group.size=5)

#(8) Comp predictions
set.seed(1)
comppred_pf=gpf(Data=cdata$data,tree=cdata$phy,
                frmla.phylo=pred_comp~phylo,
                family=gaussian,algorithm='phylo',nfactors=10,min.group.size=5)
```

```
#(9) Summarize
pcrpred_pf_results=pfsum(pcrpred_pf)$results  # runs pfsum fxn on PCR predictions (pcrpred_pf); HolmPro
comppred_pf_results=pfsum(comppred_pf)$results

#(10) Add model
pcrpred_pf_results$model="infection"
comppred_pf_results$model="competence"

#(11) Bind
predpfs=rbind.data.frame(pcrpred_pf_results,comppred_pf_results)

#(12) Round
predpfs$clade=round(predpfs$clade,2)
predpfs$other=round(predpfs$other,2)

#(13) Write
write.csv(predpfs,"Output/TableS6.csv")
```

Model predictions: Re-plot predicted probabilities with phylogenetic tree - Figure 3(C)

```
#(1) Combine tree and data
dtree=treeio::full_join(as.treedata(cdata$phy),cdata$data,by="label")

#(2) Plot base tree
pbase=ggtree(dtree,layout="fan",branch.length="none",size=0.25)

#(3) Get tree data
tdata=pbase$data

#(4) Get tips only
tdata=tdata[which(tdata$isTip==T),]

#(5) Set x max
xmax=max(tdata$x)+10

#(6) Make data frame
samp=data.frame(x=tdata$x,
                y=tdata$y,
                yend=tdata$y,
                xend_pcr=rescale(cdata$data$pred_pcr,c(max(tdata$x),xmax)),
                xend_comp=rescale(cdata$data$pred_comp,c(max(tdata$x),xmax)),
                pred_pcr=(cdata$data$pred_pcr),
                pred_comp=(cdata$data$pred_comp),
                treename=tdata$label)

#(7) Merge with cat
samp=merge(samp,apreds2[c("treename","cat")],by="treename",all.x=T)

#(8) PCR
gg=pbase
for(i in 1:nrow(pcrpred_pf_results)){
```

```
  gg=gg+
    geom_hilight(node=pcrpred_pf_results$node[i],
                 alpha=ifelse(pcrpred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25),
                 fill="black")
}

#(9) Add preds
p1=gg+
  geom_segment(data=samp,aes(x=x,y=y,xend=xend_pcr,yend=yend,colour=cat),linewidth=0.75)+
  scale_colour_manual(values=cc)+
  scale_fill_manual(values=cc)+
  guides(colour="none")

#(10) Competence
gg=pbase
for(i in 1:nrow(comppred_pf_results)){

  gg=gg+
    geom_hilight(node=comppred_pf_results$node[i],
                 alpha=ifelse(comppred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25),
                 fill="black")
}

#(11) Add preds
p2=gg+
  geom_segment(data=samp,aes(x=x,y=y,xend=xend_comp,yend=yend,colour=cat),linewidth=0.75)+
  scale_colour_manual(values=cc)+
  scale_fill_manual(values=cc)+
  guides(colour="none")

#(12) Figure F3C - Combine p1 and p2
f3C=p1+p2
f3C=ggarrange(p1,p2,
              labels=c("(b) RT-PCR predictions","(c) Virus isolation predictions"),
              label.x=c(-0.03,-0.1),
              label.y=0.1,
              font.label=list(face="plain",size=13))

#(13) Revise Figure 3
png("Output/Figure3.png",width=7,height=7.25,units="in",res=300)
#f3+f3C+plot_layout(nrow=2,heights=c(1.25,1))
ggarrange(f3,f3C,nrow=2,heights=c(1.1,1))
#f3B+f3C+plot_layout(nrow=2,heights=c(1,1.5))
#f3B|(p1/p2)+plot_layout(widths=c(1.5,1))
dev.off()
```

Model predictions V2: Re-plot predicted probabilities with phylogenetic tree for MANUSCRIPT

```
####For manuscript, create phylotree of hosts and non-hosts

#view distribution of predicted probabilities
```

```
# hist(cdata$data$pred_pcr)
# hist(cdata$data$pred_comp)

#(1) Combine tree and data
dtree=treeio::full_join(as.treedata(cdata$phy),cdata$data,by="label")

#(2) Plot base tree
pbase=ggtree(dtree,layout="fan",branch.length="none",size=0.25)

#(3) Get tree data
tdata=pbase$data

#(4) Get tips only
tdata=tdata[which(tdata$isTip==T),]

#(5) Set x max
xmax=max(tdata$x)+10

#make dataframe
samp=data.frame(x=tdata$x,
                y=tdata$y,
                yend=tdata$y,
                xend_pcr=rescale(cdata$data$pred_pcr,c(max(tdata$x),xmax)),
                xend_comp=rescale(cdata$data$pred_comp,c(max(tdata$x),xmax)),
                xend_mean=rescale(cdata$data$pred_mean,c(max(tdata$x),xmax)),
                pcr_rs0.8=(cdata$data$pcr_rs0.8),
                pcr_mss3=(cdata$data$pcr_mss3),
                comp_rs0.8=(cdata$data$comp_rs0.8),
                comp_mss3=(cdata$data$comp_mss3),
                factor_pcr_rs0.8=as.factor((cdata$data$pcr_rs0.8)),
                factor_pcr_mss3=as.factor((cdata$data$pcr_mss3)),
                factor_comp_rs0.8=as.factor((cdata$data$comp_rs0.8)),
                factor_comp_mss3=as.factor((cdata$data$comp_mss3)),
                treename=tdata$label)

#separate df into predicted host genera and non-predicted host genera for different thresholds
samp_nopcr_rs0.8 <- samp[samp$pcr_rs0.8==0,]
samp_pcr_rs0.8 <- samp[samp$pcr_rs0.8>0,] #121
samp_nopcr_mss3 <- samp[samp$pcr_mss3==0,]
samp_pcr_mss3 <- samp[samp$pcr_mss3>0,] #n=136

samp_nocomp_rs0.8 <- samp[samp$comp_rs0.8==0,]
samp_comp_rs0.8 <- samp[samp$comp_rs0.8>0,] #100
samp_nocomp_mss3 <- samp[samp$comp_mss3==0,]
samp_comp_mss3 <- samp[samp$comp_mss3>0,] #191

#() load library for color palette
library(viridis)

#(8) Plot base tree and highlight significant clades for PCR model
gg=pbase
for(i in 1:nrow(pcrpred_pf_results)){
  gg=gg+
```

```
    geom_hilight(node=pcrpred_pf_results$node[i],
                 alpha=ifelse(pcrpred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25),
                 fill="black")
}

#(9) Add PCR pred-probs and assign color based on binary classification assuming rs0.8
p1=gg+
  geom_segment(
    data=samp_nopcr_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_pcr,yend=yend,
                alpha=factor_pcr_rs0.8),
    color="gray", linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_pcr_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_pcr,yend=yend,
                colour=factor(factor_pcr_rs0.8)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      name="Evidence type")

#(9) Add PCR pred-probs and assign color based on binary classification assuming mss3
p2=gg+
  geom_segment(
    data=samp_nopcr_mss3,
    mapping=aes(x=x,y=y,xend=xend_pcr,yend=yend,
                alpha=factor_pcr_mss3),
    color="gray", linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_pcr_mss3,
    mapping=aes(x=x,y=y,xend=xend_pcr,yend=yend,
                colour=factor(factor_pcr_mss3)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      name="Evidence type")

#(10) Competence
gg=pbase
for(i in 1:nrow(comppred_pf_results)){
  gg=gg+
    geom_hilight(node=comppred_pf_results$node[i],
                 alpha=ifelse(comppred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25),
                 fill="black")
}

#(11) Add preds to comp results based on rs0.8
p3=gg+
  geom_segment(
    data=samp_nocomp_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_comp,yend=yend,
                alpha=factor_comp_rs0.8),
    color="gray", linewidth=0.75)+
```

```
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_comp_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_comp,yend=yend,
                colour=factor(factor_comp_rs0.8)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      name="Evidence type")

#(11) Add preds to comp results based on mss3
p4=gg+
  geom_segment(
    data=samp_nocomp_mss3,
    mapping=aes(x=x,y=y,xend=xend_comp,yend=yend,
                alpha=factor_comp_mss3),
    color="gray", linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_comp_mss3,
    mapping=aes(x=x,y=y,xend=xend_comp,yend=yend,
                colour=factor(factor_comp_mss3)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      name="Evidence type")

pp_tree2=ggarrange(p1,p2,p3,p4,
              labels=c("(A) PCR ReqSens0.8: Th=0.38","(B) PCR MaxSensSpec: Th=0.36", "(C) Comp ReqSens0
              # font.label(size=10),
              hjust=-0.6,
              label.x=c(0.1,0.1,0.1,0.1),
              label.y=c(1,1,1,1),
              font.label=list(face="plain",size=13),
              ncol=2,nrow=2,
              common.legend = TRUE, legend="left")

#Save
png("Output/pp_tree2_f1.png",width=12,height=8,units="in",res=300)
pp_tree2
dev.off()
```

Model predictions V3: Re-plot predicted probabilities with phylogenetic tree for MANUSCRIPT

```
####For manuscript, create phylotree of hosts and non-hosts

#view distribution of predicted probabilities
# hist(cdata$data$pred_pcr)
# hist(cdata$data$pred_comp)

#(1) Combine tree and data
dtree=treeio::full_join(as.treedata(cdata$phy),cdata$data,by="label")

#(2) Plot base tree
```

```
pbase=ggtree(dtree,layout="fan",branch.length="none",size=0.25)

#(3) Get tree data
tdata=pbase$data

#(4) Get tips only
tdata=tdata[which(tdata$isTip==T),]

#(5) Set x max
xmax=max(tdata$x)+10

#make dataframe
samp=data.frame(x=tdata$x,
                y=tdata$y,
                yend=tdata$y,
                xend_pcr=rescale(cdata$data$pred_pcr,c(max(tdata$x),xmax)),
                xend_comp=rescale(cdata$data$pred_comp,c(max(tdata$x),xmax)),
                xend_mean=rescale(cdata$data$pred_mean,c(max(tdata$x),xmax)),

                pcrcomp_rs0.8=(cdata$data$pcrcomp_rs0.8),
                pcrcomp_mss3=(cdata$data$pcrcomp_mss3),
                factor_pcr_rs0.8=as.factor((cdata$data$pcr_rs0.8)),
                factor_pcr_mss3=as.factor((cdata$data$pcr_mss3)),
                factor_comp_rs0.8=as.factor((cdata$data$comp_rs0.8)),
                factor_comp_mss3=as.factor((cdata$data$comp_mss3)),
                treename=tdata$label)

#separate df into predicted host genera and non-predicted host genera for different thresholds
samp_no_rs0.8 <- samp[samp$pcrcomp_rs0.8=="Not predicted",]
samp_pcr_rs0.8 <- samp[samp$pcrcomp_rs0.8=="PCR",]
samp_comp_rs0.8 <- samp[samp$pcrcomp_rs0.8=="Virus isolation",]
samp_both_rs0.8 <- samp[samp$pcrcomp_rs0.8=="Both",]
samp_no_mss3 <- samp[samp$pcrcomp_mss3=="Not predicted",]
samp_pcr_mss3 <- samp[samp$pcrcomp_mss3=="PCR",]
samp_comp_mss3 <- samp[samp$pcrcomp_mss3=="Virus isolation",]
samp_both_mss3 <- samp[samp$pcrcomp_mss3=="Both",]

#() load library for color palette
library(viridis)

#(8) Plot base tree and highlight significant clades for PCR model
gg=pbase
for(i in 1:nrow(pcrpred_pf_results)){
  gg=gg+
    geom_hilight(node=pcrpred_pf_results$node[i],
                 alpha=ifelse(pcrpred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25),
                 fill="black")
}

#(9) Add PCR pred-probs and assign color based on whether host genera had any predicted host-virus link
p1=gg+
  geom_segment(
    data=samp_no_rs0.8,
```

```
    mapping=aes(x=x,y=y,xend=xend_mean,yend=yend,
                alpha=factor_pcr_rs0.8),
    color="gray", linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_pcr_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_pcr,yend=yend,
                colour=factor(pcrcomp_rs0.8)),linewidth=0.75,)+
  geom_segment(
    data=samp_comp_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_comp,yend=yend,
                colour=factor(pcrcomp_rs0.8)),linewidth=0.75,)+
  geom_segment(
    data=samp_both_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_mean,yend=yend,
                colour=factor(pcrcomp_rs0.8)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      name="Evidence type")
  # geom_text2(aes(subset = !is.na(label)), label = cdata$phy$tip.label, size = 3, color = "black")

#(10) Competence
gg=pbase
for(i in 1:nrow(comppred_pf_results)){
  gg=gg+
    geom_hilight(node=comppred_pf_results$node[i],
                 alpha=ifelse(comppred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25),
                 fill="black")
}

#(11) Add preds
p2=gg+
  geom_segment(
    data=samp_no_mss3,
    mapping=aes(x=x,y=y,xend=xend_mean,yend=yend,
                alpha=factor_pcr_mss3),
    color="gray", linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_pcr_mss3,
    mapping=aes(x=x,y=y,xend=xend_pcr,yend=yend,
                colour=factor(pcrcomp_mss3)),linewidth=0.75,)+
  geom_segment(
    data=samp_comp_mss3,
    mapping=aes(x=x,y=y,xend=xend_comp,yend=yend,
                colour=factor(pcrcomp_mss3)),linewidth=0.75,)+
  geom_segment(
    data=samp_both_mss3,
    mapping=aes(x=x,y=y,xend=xend_mean,yend=yend,
                colour=factor(pcrcomp_mss3)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      name="Evidence type")
```

```
pp_tree3=ggarrange(p1,p2,
                labels=c("(A) ReqSens0.8; Th_inf=0.38 & Th_comp=0.26","(B) MaxSensSpec; Th_inf=0.36 & Th_
                # font.label(size=10),
                hjust=-0.6,
                label.x=c(0.1,0.1),
                label.y=c(1,1),
                font.label=list(face="plain",size=13),
                ncol=1,nrow=2,
                common.l egend = TRUE, legend="left")

#Save
png("Output/pp_tree3_f1.png",width=12,height=8,units="in",res=300)
pp_tree3
dev.off()
```

5. Mapping Host Distributions

Load required packages and set system

```
#(1) Libraries for generating maps
library(classInt)
library(tidyverse)
library(raster)
library(rgdal)  # switches to sf in 2023
library(dismo)
library(XML)
library(maps)
library(sp)
library(dplyr)
library(devtools)
install_github("hunzikp/velox", force=TRUE)
library(velox)
library(fasterize)
library(sf)

#(2) Clean environment
rm(list=ls())
graphics.off()

# Set working directory
setwd("~/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernandez Lab/Projects
```

Let's make some maps! - Figure 4

```
## Before proceeding, make sure you have downloaded "MAMMALS.shp" to your working directory. This shape

####Let's combine comp and pcr for manuscript
pred %>% filter(competence==1|PCR==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> known
```

```
pred %>% filter(bin_comp==1|bin_pcr==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred.pcrcomp #
sort(pred.pcrcomp[!(pred.pcrcomp %in% known)]) -> unknown #235

#(1) Load shape file of mammal geographic range
iucn <- sf::st_read(dsn = "/Users/katietseng/Fernandez Lab Dropbox/Katie Tseng/Mac/Desktop/PoxHost(copy)

#(2) Make a blank raster (must be connected to wifi for the disaggregate function)
r <- raster::disaggregate(getData("worldclim",var="alt",res=2.5)*0,2)

#(3) Create four layers
iucn$treename=sapply(strsplit(iucn$binomial,' '),function(x) paste(x[1],sep=' '))

iucn.1 <- iucn[iucn$treename %in% known.comp,]
iucn.2 <- iucn[iucn$treename %in% known.pcr,]
iucn.3 <- iucn[iucn$treename %in% pred.comp,]
iucn.4 <- iucn[iucn$treename %in% pred.pcr,]

map.knc <- (fasterize(iucn.1, r, fun="sum"))
map.knp <- (fasterize(iucn.2, r, fun="sum"))
map.prc <- (fasterize(iucn.3, r, fun="sum"))
map.prp <- (fasterize(iucn.4, r, fun="sum"))

#(4) Add zeros for the continental area
fix <- function(x) {sum(x,r,na.rm=TRUE)+r}

map.knc <- fix(map.knc)
map.knp <- fix(map.knp)
map.prc <- fix(map.prc)
map.prp <- fix(map.prp)

raster::stack(map.knp, map.knc, map.prp, map.prc) %>%    #alternatively, can use tera package
raster::trim() -> maps

names(maps) <- c('KnownPCR', 'KnownComp', 'PredPCR', 'PredComp')

#(5) Generate the actual visualization
library(rasterVis)
library(RColorBrewer)

mycolors <- colorRampPalette(rev(brewer.pal(10,"Spectral")))(21)
mycolors[1] <- "#C0C0C0"

png("Output/map_figure_f1.png",width=10,height=10,units="in",res=300)
rasterVis::levelplot(maps,
                     col.regions = mycolors,
                     #at = seq(0, 15, 1),
                     alpha = 0.5,
                     scales=list(alternating=FALSE),
                     par.strip.text=list(cex=0),
                     xlab = NULL, ylab = NULL,
                     #labels = labels,
                     maxpixels = 5e6)
dev.off()
```

```
### Generate map for manuscript combining predictions from infection and competence models ###

#create layers
iucn_known <- iucn[iucn$treename %in% known,]
iucn_unknown <- iucn[iucn$treename %in% unknown,]
map_known <- (fasterize(iucn_known, r, fun="sum"))
map_unknown <- (fasterize(iucn_unknown, r, fun="sum"))

#add zeros for the continental area
map_known <- fix(map_known)
map_unknown <- fix(map_unknown)

raster::stack(map_known, map_unknown) %>%    #alternatively, can use tera package
raster::trim() -> maps

names(maps) <- c('KnownPCR+Comp', 'PredUnknownPCR+Comp')

png("Output/map_figure_f1.png",width=10,height=10,units="in",res=300)
rasterVis::levelplot(maps,
                     col.regions = mycolors,
                     #at = seq(0, 15, 1),
                     alpha = 0.5,
                     scales=list(alternating=FALSE),
                     par.strip.text=list(cex=0),
                     xlab = NULL, ylab = NULL,
                     #labels = labels,
                     maxpixels = 5e6)
dev.off()
```