Orthopoxvirus Link Prediction Model Code

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

Table of Contents

Introduction

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

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

To run the following script, four files are required in your working directory: (1) Data_raw.RData, the raw data file; (2) OPVnew_nowwithVirus.xlsx, the excel file of OPV genome annotations (3) "Output" folder, where all output will be saved - e.g., cleaned datasets, model results, figures, and tables; (4) 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

Dimension Reduction

In this section, we draw from our dataset of annotated OPV genomes (n=197 unique sequences; see genome annotation pipeline in methods section for more information), which compose our known host-OPV associations for incorporation in the link prediction model. Each sequence is classified by its source or virus species and its host, as well as the presence (1) or absence (0) of a suite of OPV accessory genes. Because our data consists of over 981 OPV accessory genes, the goal of this section is to reduce the number of viral predictors for incorporation in the link prediction model, while maintaining maximal variance. Using principal components analysis (PCA), a method of dimension reduction, we distill the variables down to their most important features. We also explore which sequences have the most similar values (aka, how do they group?) in the principal components using k-means clustering. Because we are analyzing presence/absence data (binary variables), we use multiple correspondence analysis (MCA) to further validate our results. Though similar to PCA, MCA is used to analyze datasets with multiple categorical variables.

Load packages, clean environment, and set working directory

```
#(1) Libraries for preparing data for analysis
library(ape)
library(dplyr)
library(nlme)
library(tidyverse)
library(vroom)
library(readxl)
library(ggplot2)

#(2) Libraries for PCA (principal components analysis)
library(vegan)
library(factoextra) #fviz_eig
library(ggfortify)

#(3) Libraries for MCA
library(FactoMineR)
library(dplyr)
library(factoextra) #fviz_eig

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

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

Prepare genomic data for dimension reduction

Let's clean our sequence data and explore the variability in the data!

```
#(1) Load genome annotations and trim
genes <- read_xlsx("OPVnew_nowwithVirus.xlsx", sheet="PoxHost")

#(2) Rename variables and exclude unnecessary variables
genes <- plyr::rename(genes, c("Virus"="VirusSpecies","Host Genus"="HostGenus","Host Species"="HostSpeci
genes <- subset(genes, select=-c(HostSpecies))

#(3) Correct sequence MT903347_1 - 'HostGenus' var lists Family name instead of Genus
genes$HostGenus <- ifelse(genes$HostGenus=="Gliridae","Graphiurus",genes$HostGenus)

#(4) Add unique identifier
genes$rownames <- rownames(genes)
genes$Sequence <- paste(genes$Genome,genes$VirusSpecies,genes$HostGenus,sep="_",genes$rownames)
genes$rownames=NULL
genes <- genes %>% dplyr::select(Sequence, everything())

#(5) View frequency of various virus species
prop_table <- subset(genes, select=-c(Sequence,Genome))
prop_table$Frequency = 1
prop_table <- aggregate(Frequency ~ VirusSpecies + HostGenus, data=prop_table, FUN=sum)
```

```r
prop_table <- prop_table[order(prop_table[,c("VirusSpecies")],prop_table[,c("HostGenus")]) ,]
prop_table$Perc <- prop_table$Frequency/sum(prop_table$Frequency)*100
print(prop_table)

#(6) Save frequency table to Output folder
write.csv(prop_table, "Output/dim_genes_t1.csv")

#(7) Create function (mode.prop) to assess variation in the presence/absence of OPV genes
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) each uniqu
  max(tab)/length(x[is.na(x)==FALSE])     # max-frequency / number of elements in each column that are
}

#(8) Assess variation across columns (2 indicates columns)
vars=data.frame(apply(genes,2,function(x) mode.prop(x)),
                apply(genes,2,function(x) length(unique(x)))) # number of unique elements in each colum
vars$variables=rownames(vars)
colnames(vars) <- c("var","uniq","column")

#(9) Trim
vars <- vars[-c(1:4), ]

#(10) Any variables with no variation? If so drop
which(vars$var==1)
# vars <- subset(vars,vars$var<1)

#(11) Visualize distribution of variation
gene_var <- ggplot(vars,
       aes(var))+
  geom_histogram(bins=50)+
  geom_vline(xintercept=0.70,linetype=2,linewidth=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="Variance in the presence/absence of genes")+
  scale_x_continuous(labels=scales::percent)

#() Save figure
png("Output/dim_genes_f1.png", width=4,height=4,units="in",res=600)
gene_var
dev.off()

#(12) Clean environment
rm(list=setdiff(ls(), c("genes")))
```

PCA(1) of viral accessory genes

Using principal components analysis, can we distill the variables down to their most important features? Which genes contribute the most to each feature?

```r
# Subset data and reformat as numeric matrix
# genes_mat <- subset(genes,select=-c(Genome,VirusSpecies,HostGenus))
# mat <- as.matrix(genes_mat[,-1])
# rownames(mat) <- genes_mat[,1] %>% pull()
# class(mat) <- "numeric"

#(1) Apply PCA using stats::prcomp
pca1 <- prcomp(genes[,5:985])      #scaling/centering not appropriate
relvar <- pca1$sdev^2 / sum(pca1$sdev^2)
relvar_per <- round(relvar*100,1)

#(2) View summary results
# summary(pca1)
# View(pca1$x) #sequence (individuals)
# View(pca1$rotation) #genes (variables)

#(3) Table of importance of components: Eigenvalue, SD, Proportion of Variance, Cumulative Prop
importance <- as.data.frame(t(summary(pca1)$importance))
importance$Eigenvalue <- importance$`Standard deviation`^2
importance <- importance %>% dplyr::relocate(Eigenvalue)
importance <- importance[c(1:12),]
write.csv(importance,"Output/dim_pca1_t1.csv")
### Eigenvalue: the variance explained by each PC

#(4) Table of loadings: $rotation is the matrix of variable loadings where columns are eigenvectors
loadings <- as.data.frame(pca1$rotation)
loadings <- loadings[,c(1:10)]
# loadings <- abs(loadings) #get absolute values
### Why are some loadings > |1|? Loading is the covariances/correlations b/w original vars and unit-sca

#(5) For each dimension, create df of accessory genes
for(i in 1:ncol(loadings)){
  assign(colnames(loadings)[i], data.frame(loadings[,i]))
}

#(6) Create list of dataframes of PC loadings
list <- colnames(loadings)
list_df = lapply(list, get)

#(7) To each dataframe in that list, add corresponding gene name and sort in descending order (genes wi
for (i in 1:length(list)) {
  colnames(list_df[[i]]) <- "Loadings"
  list_df[[i]]$Gene <- rownames(loadings)
  list_df[[i]]=list_df[[i]][order(-list_df[[i]]$Loadings),]
}

for (i in 1:length(list)) {
  colnames(list_df[[i]]) <- "Loadings"
}

for (i in 1:length(list)) {
  list_df[[i]]$Gene <- rownames(loadings)
}
```

```
for (i in 1:length(list)) {
  list_df[[i]]=list_df[[i]][order(-list_df[[i]]$Loadings),]
}

#(8) Create df of just ranked genes (drop loadings)
genes_df <- list_df
for(i in 1:length(list)) {
  genes_df[[i]]$Loadings=NULL
}
rank_genes <- data.frame(matrix(ncol=ncol(loadings), nrow=nrow(loadings)))
colnames(rank_genes) <- colnames(loadings)
for(i in 1:length(list)) {
  rank_genes[,i] = genes_df[[i]]
}
rank_genes <- setNames(rank_genes, paste0(names(loadings), '_', 'Gene'))

#(9) Create df of just ranked loadings
loadings_df <- list_df
for(i in 1:length(list)) {
  loadings_df[[i]]$Gene=NULL
}
rank_loadings <- data.frame(matrix(ncol=ncol(loadings), nrow=nrow(loadings)))
colnames(rank_loadings) <- colnames(loadings)
for(i in 1:length(list)) {
  rank_loadings[,i] = loadings_df[[i]]
}
rank_loadings <- setNames(rank_loadings, paste0(names(loadings), '_', 'Loadings'))

#(10) Combine dfs of ranked genes and loadings and reorder columns
rank_PC <- cbind(rank_genes, rank_loadings)
rank_PC <- rank_PC[,order(colnames(rank_PC))]
rank_PC <- rank_PC %>% relocate(c("PC10_Gene","PC10_Loadings"), .after = last_col())

#(11) Save table of PCA Loadings Ranked
write.csv(rank_PC, "Output/dim_pca2_t2.csv")

#() Distribution of loadings for each PC (particularly PC1, PC3, PC4, PC9)
loadings <- as.data.frame(pca1$rotation)
loadings <- loadings[,c(1:10)]
loadings_abs <- abs(loadings) #get absolute values

library(reshape2)
hist_loadings=ggplot(melt(loadings),aes(x=value)) + geom_histogram() + facet_wrap(~variable)
hist_loadings_abs=ggplot(melt(loadings_abs),aes(x=value)) + geom_histogram() + facet_wrap(~variable)
hist_loadings_abslim=ggplot(melt(loadings_abs),aes(x=value)) + geom_histogram() + xlim(0.05,0.20) + face

#() save figure
png("Output/dim_pca1_f1.png", width=4,height=4,units="in",res=600)
hist_loadings
dev.off()

#(12) Save datafile of PC scores for link prediction
wgs_PCs <- genes[,1:4]
```

```
wgs_PCs <- cbind(wgs_PCs, pca1$x[,1:10])
# save(wgs_PCs, file='Output/wgs_PCs.RData')

#(13) Clean environment
rm(list=setdiff(ls(), c("genes","pca1","relvar","relvar_per")))
```

PCA(1) visualizations

Do all of the dimensions spark joy?

```
#-----------------------------------------------------------------------------
#Vizualize variance: screeplots, cumulative variance, etc.
#-----------------------------------------------------------------------------

#(1) Screeplot variance (eigenvalues) to show the decreasing rate at which variance is explained by add
png("Output/dim_pca1_f2.png", width=4,height=4,units="in",res=600)
screeplot(pca1, type="lines", npcs=15, main="Scree plot of Eigenvalues for the first 15 PCs")
abline(h=1, col="red", lty=5)
legend("topright", legend=c("Eigenvalue = 1"), col=c("red"), lty=5, cex=1)
dev.off()
### suggests cutoff at PC12

#(2) Screeplot cumulative variance to show the % variance explained by additional PCs
screeplot_var <- barplot(relvar_per[1:10], xlab='PC', ylab='Percentage of explained variances', main='S
text(screeplot, 0, y=relvar_per[1:10], label=relvar_per[1:10],cex=0.8, pos=3, col="red")
# fviz_eig(pca, choice=c("variance"), main = "Scree plot of explained variances") # these values agree

#(3) Plot cumulative variance to show the proportion of variance explained with each add'l PC
cumpro <- cumsum(pca1$sdev^2 / sum(pca1$sdev^2))
png("Output/dim_pca1_f3.png", width=4,height=4,units="in",res=600)
plot(cumpro[0:15], xlab = "Dimension", ylab = "Proportion of explained variance", main = "Cumulative va
abline(v = 10, col="blue", lty=5)
abline(h = 0.7, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC10"), col=c("blue"), lty=5, cex=1)
dev.off()

#() Clean
rm(relvar,relvar_per)
#-----------------------------------------------------------------------------
#Vizualize individuals/scores
#-----------------------------------------------------------------------------

#(4) Plot sequences
fviz_pca_ind(pca1) + ggtitle("PCA Plot of Sequences")
### with ellipses
fviz_pca_ind(pca1, geom.ind = "point", pointshape = 21, pointsize = 2,
             col.ind = "black", addEllipses = TRUE, label = "var",
             col.var = "black", repel = TRUE,
             alpha.ind = 0.7) +
             ggtitle("PCA Plot of Sequences") +
             theme(plot.title = element_text(hjust = 0.5))

#(5) Plot sequences by virus species for dim 1 and 2
```

```
png("Output/dim_pca1_fig4.png", width=6,height=4,units="in",res=600)
autoplot(pca1, data = genes, colour = 'VirusSpecies')
# + ggtitle("Plot of Sequences by Virus Species")
dev.off()

#-------------------------------------------------------------------------------
#Vizualize variables/loadings: by virus family, etc.
#-------------------------------------------------------------------------------

#(6) Plot gene loadings for dim 1 and 2
png("Output/dim_pca1_fig5.png", width=6,height=4,units="in",res=600)
fviz_pca_var(pca1,
             col.var = "Contribution", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE,     # Avoid text overlapping
             label = c("ind", "ind.sup", "quali", "var", "quanti.sup"))
             # + ggtitle("Plot of Gene Loadings for Dimensions 1 and 2")
dev.off()
### Here we see PC1 has large positive associations with a number of AGs like ADZ29556.1, SNB51281.1, a
### QKE61192.1 - hypothetical protein [Vaccinia virus]
### QNP13375.1 - MPXV Viral membrane assembly proteins (VMAP) (Cop-A 30.5L)"

#(7) Plot gene loadings for dim 3 and 4
png("Output/dim_pca1_fig6.png", width=6,height=4,units="in",res=600)
fviz_pca_var(pca1, axes = c(3, 4),
             col.var = "Contribution",
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE)
             # + ggtitle("Plot of Gene Loadings for Dimensions 3 and 4")
dev.off()

#(8) Biplot sequences and gene loadings
png("Output/dim_pca1_fig7.png", width=6,height=4,units="in",res=600)
fviz_pca_biplot(pca1, col.var = "Contribution", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
                repel = TRUE)
                # + ggtitle("Biplot of Sequences and Gene Loadings")
dev.off()

#(9) Biplot top 20 influential scores and loadings
png("Output/dim_pca1_fig8.png", width=6,height=4,units="in",res=600)
fviz_pca_biplot(pca1, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE, select.ind=list(contrib=20), select.var=list(contrib=20), max.overlaps=Inf)
             # + ggtitle("Biplot of Top 20 Contributing Sequences and Gene Loadings")
dev.off()

rm(list=setdiff(ls(), c("genes","pca1")))
```

PCA(1) Hierarchical Cluster Analysis

```
### Determine Optimal Number of Clusters ###

library(clustree)
```

```r
#(1) Extract coordinates for individual sequences
ind.coord <- pca1$x
rownames(ind.coord) <- genes$VirusSpecies
ind.coord <- ind.coord[,1:10]

###Elbow method
# Calculate/graph the sum of squares at each number of clusters. Assess how the slope changes from stee

#() Set seed and execute
set.seed(31)
png("Output/dim_cluster_f1.png", width=4,height=4,units="in",res=600)
fviz_nbclust(ind.coord, kmeans, method = "wss", k.max = 20) + theme_minimal()
# + ggtitle("the Elbow Method")
dev.off()

#### Clustree method
# How do samples change groupings as the number of clusters increases? Clustree shows which clusters ar

# Execution of k-means with 1 to 10 clusters
tmp <- NULL
for (k in 1:10){
  tmp[k] <- kmeans(ind.coord, k, nstart = 30) #generates 30 initial configs & avg all centroid results
}

# Convert to dataframe
df <- data.frame(tmp)

# Add a prefix to the column names of df
colnames(df) <- seq(1:10)
colnames(df) <- paste0("k",colnames(df))

# Combine clustering data w/ PC loading values
df <- bind_cols(as.data.frame(df), as.data.frame(ind.coord))

# Execute clustree
png("Output/dim_cluster_f2.png", width=6,height=8,units="in",res=600)
clustree(df, prefix = "k")
dev.off()
### note: nodes with multiple incoming edges indicate that we over-clustered the data

# Execute clustree, but display stability index by colour as opposed to count
png("Output/dim_cluster_f3.png", width=6,height=8,units="in",res=600)
clustree(df, prefix = "k", node_colour = "sc3_stability")
dev.off()
### note: k=6 appears to be the most stable number of clusters

# Execute clustree_overlay, overlaying PC1 on PC2 to assess quality of the clustering
df_subset <- df %>% dplyr::select(1:10,11:12)
png("Output/dim_cluster_f4.png", width=9, height=12, units="in", res=600)
clustree_overlay(df_subset, prefix = "k", x_value = "PC1", y_value = "PC2")
dev.off()

# Execute clustree overlaying PC vs. k (resolution dimension)
```

```
overlay_list <- clustree_overlay(df_subset, prefix = "k", x_value = "PC1",
                                 y_value = "PC2", plot_sides = TRUE)
overlay_list$x_side  #PC1
overlay_list$y_side  #PC2

### Conduct Hierarchical Cluster Analysis ###
# Each object is assigned its own cluster iteratively, at each stage joining the 2 most similar clusters

#() Extract coordinates for individual sequences
ind.coord <- pca1$x
rownames(ind.coord) <- genes$VirusSpecies
ind.coord <- ind.coord[,1:10]

#() Calculate distance matrix
pc.dist <- dist(ind.coord, method="euclidean")

#() Execute HCA
clusters <- hclust(pc.dist)

#() Plot dendogram
png("Output/dim_cluster_f5", width=8,height=5,units="in",res=600)
plot(clusters, cex=0.2)
dev.off()

#() For k=6, add rectangular outline to each cluster in the dendrogram
png("Output/dim_cluster_f5.png", width=8,height=5,units="in",res=600)
plot(clusters, cex=0.2)
rect.hclust(clusters, k = 6, border = 2:8) # add rectangle
dev.off()

#() For k=6, view count of each cluster
clusterCut <- cutree(clusters, k = 6)
table(clusterCut)

#() Prop tables by virus species
mytable<-table(clusterCut, genes$VirusSpecies)
mytable2 <- data.frame(prop.table(mytable,2))
ggplot(mytable2, aes(x = Var2, y = Freq, fill = clusterCut)) +
  geom_col() +
  labs(fill='Cluster') +
  theme(axis.title.x = element_blank(), axis.text.x = element_text(angle=45,hjust=1)) +
  ggtitle("Distribution by Virus Species")

#() Re-run PCA to color by cluster
  #add cluster to original db
  genes1<-data.frame(cbind(genes,clusterCut))
  genes1$clusterCut <- as.factor(genes1$clusterCut)

#(6) Run PCA as before, but now grouping by cluster
pca1 <- prcomp(genes1[,5:985])


# color-blind friendly palette:
```

```
cbPalette1 <- c("#009292","#ff6db6","#ffb6db","#490092","#6db6ff","#db6d00")
cbPalette2 <- c("#E69F00", "#56B4E9", "#009E73", "#0072B2", "#D55E00", "#CC79A7")

#(7) Plot of sequences by cluster
png("Output/dim_cluster_f6.png", width=8,height=5,units="in",res=600)
fviz_pca_ind(pca1, geom.ind = "point", pointshape = 21,
             pointsize = 2,
             fill.ind = genes1$clusterCut,
             col.ind = "black",
             addEllipses = TRUE,
             label = "var",
             col.var = "black",
             repel = TRUE,
             legend.title = "Cluster",
             palette = cbPalette1,
             alpha.ind = 0.5)
dev.off()
  # + ggtitle("PCA Plot of Sequences by Cluster")

#(8) Biplot of sequences and gene loadings by cluster
fviz_pca_biplot(pca1, geom.ind = "point", pointshape = 21,
             pointsize = 2,
             fill.ind = genes1$clusterCut,
             col.ind = "black",
             label = "var",
             repel = TRUE,
             legend.title = "Cluster",
             addEllipses = TRUE,
             alpha.ind = 0,
             col.var = "grey40")
     # + ggtitle("PCA Biplot of Sequences and Gene Loadings by Cluster")

#(9) Create a table of sequences by cluster
db_cluster <- dplyr::select(genes1, Genome, VirusSpecies, HostGenus,clusterCut)
write.csv(db_cluster,"Output/dim_cluster_t1.csv", row.names = F)

#(10) Are the MPXV sequences in cluster 3 the same as the outliers in PCA4?
print(db_cluster[(db_cluster$VirusSpecies=="Monkeypox virus" & db_cluster$clusterCut==3),])
# print(outliers[,1:4])

#(11) Clean environment
rm(list=setdiff(ls(), c("genes","genes1","mytable2", "pca1")))
```

Visualize clustering on viral phylogentic tree

```
#Resources
#https://github.com/YuLab-SMU/ggtree/issues/295

# load libraries
library(ggtree)
library(tidyverse)
library(clustree)
```

```
library(ape)
library(phylogram)

# load viral tree data
vtree <- ape::read.nexus("OPV_6Jan23.nexus")
print(vtree$tip.label)

# reformat tip label by dropping everything before the underscore
vtree$tip.label=substring(vtree$tip.label, regexpr("_", vtree$tip.label) + 1, nchar(vtree$tip.label))

# repeat this step for tip labels with two underscores
vtree$tip.label=substring(vtree$tip.label, regexpr("_", vtree$tip.label) + 1, nchar(vtree$tip.label))

# drop single quote in tip label
vtree$tip.label=gsub("'","",vtree$tip.label)

# rename tip label Abatino to Abatino macacapox
vtree$tip.label <- ifelse(vtree$tip.label=="Abatino", "Abatino macacapox", vtree$tip.label)

# rename tip label Cetacean to Cetaceanpox
vtree$tip.label <- ifelse(vtree$tip.label=="Cetacean", "Cetaceanpox", vtree$tip.label)

# add ' virus' to tip label
vtree$tip.label=paste0(vtree$tip.label," virus")

# convert tree data from class "phylo" to "dendrogram"
dendrogram <- as.dendrogram(vtree)

# copy dataframe of cluster data (freq table of virus species by cluster)
mytable3 = mytable2

# rename/reformat variables
mytable3$tip.label = mytable2$Var2
mytable3$Cluster = as.factor(mytable2$clusterCut)
mytable3$freq = mytable2$Freq
mytable3 <- subset(mytable3, select=c(tip.label,Cluster,freq))

#() Check that all virus species names in mytable3 are in vtree
mytable3$intree <- ifelse(mytable3$tip.label%in%setdiff(mytable3$tip.label, vtree$tip.label),'missing',
which(mytable3$intree=="missing")
mytable3$intree=NULL

# plot of tree with scale of substitution rate
p <- ggtree(vtree) + geom_tiplab(align=TRUE, size=0) +
     geom_treescale(x=0.1, y=12, width=0.2, label="substitution rate") +
     annotate(geom="text", x=0.2, y=12.3, label="0.2", size=4)
# p <- ggtree(dendrogram) + geom_tiplab(align=TRUE, size=0)

# color-blind friendly palette:
cbPalette1 <- c("#009292","#ff6db6","#ffb6db","#490092","#6db6ff","#db6d00")
cbPalette2 <- c("#E69F00", "#56B4E9", "#009E73", "#0072B2", "#D55E00", "#CC79A7")

# plot of tree combined with bar plot of clustering
```

```
p1 <- facet_plot(p, data = mytable3, geom = geom_bar,
                 panel = "Barplot", colour = "black",
                 mapping = aes(fill=Cluster,x=freq), stat = "identity", orientation="y") +
                 scale_fill_manual(values=cbPalette1)

# plots of tree + clustering with x-axis scales
p2<- p1 + theme_tree2()

# plots of tree + clustering with x-axis scales, x-axis titles, and tiplabel names
p3 <- p2 + xlab(label = c("\n      Substitutions per site
      theme(axis.title.x = element_text(size = 12)) +
      geom_tiplab(as_ylab=TRUE, size = 10)
p3
# save
png("Output/dim_tree_f1.png", width=16,height=8,units="in",res=600)
p3
dev.off()
```

PCA(2) Alternative Analysis

What happens when we exclude accessory genes present in only one virus species?

```
#(1) Drop accessory genes that are present in only one virus species (all 0's except for one)
genes2 <- genes[c(1:4,4 + which(colSums(genes[-(1:4)])>1))]
### 985 variables to 686 variables

#(2) Apply PCA using stats::prcomp
pca2 <- prcomp(genes2[,5:686])

#-------------------------------------------------------------------------------
#Vizualize variance, scores and loadings
#-------------------------------------------------------------------------------

#(3) Plot cumulative variance to show the proportion of variance explained with each add'l PC
cumpro <- cumsum(pca2$sdev^2 / sum(pca2$sdev^2))
plot(cumpro[0:15], xlab = "Dimension", ylab = "Proportion of explained variance", main = "Cumulative va
abline(v = 10, col="blue", lty=5)
abline(h = 0.7, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC10"), col=c("blue"), lty=5, cex=1)

#(4) Biplot sequences and gene loadings
fviz_pca_biplot(pca2, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
                repel = TRUE) +
                ggtitle("Biplot of Sequences and Gene Loadings")

#(5) Biplot top 20 influential scores and loadings
fviz_pca_biplot(pca2, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
              repel = TRUE, select.ind=list(contrib=20), select.var=list(contrib=20), max.overlaps=Inf) +
              ggtitle("Biplot of Top 20 Contributing Sequences and Gene Loadings")

### Summary: Compared to PCA.1, there's an increase in the proportion of variance explained by the firs
```

PCA(3) Alternative Analysis

What happens when we drop duplicate observations within the same host-virus links (sequences with the same identical presence/absence of accessory genes as another sequence of the same host-virus link)?

```
#(1) Identify observations of the same host-virus links with identical presence/absence of accessory ge
genes3 <- genes
genes3$dup <- duplicated(genes3[,-c(1:2)])
table(genes3$dup)
### 42 dups

#(2) Drop duplicate observations
genes3 <- genes3[genes3$dup==FALSE,]
genes3$dup=NULL
### 197 obs to 155 obs

#(3) Apply PCA using stats::prcomp
pca3 <- prcomp(genes3[,5:985])

#-----------------------------------------------------------------------------
#Vizualize variance, scores and loadings
#-----------------------------------------------------------------------------

#(4) Plot cumulative variance to show the proportion of variance explained with each add'l PC
cumpro <- cumsum(pca3$sdev^2 / sum(pca3$sdev^2))
plot(cumpro[0:15], xlab = "Dimension", ylab = "Proportion of explained variance", main = "Cumulative va
abline(v = 10, col="blue", lty=5)
abline(h = 0.7, col="blue", lty=5)
legend("topleft", legend=c("Cut-off @ PC10"), col=c("blue"), lty=5, cex=1)

#(5) Biplot sequences and gene loadings
fviz_pca_biplot(pca3, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
               repel = TRUE) +
               ggtitle("Biplot of Sequences and Gene Loadings")

#(6) Biplot top 20 influential scores and loadings
fviz_pca_biplot(pca3, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE, select.ind=list(contrib=20), select.var=list(contrib=20), max.overlaps=Inf) +
             ggtitle("Biplot of Top 20 Contributing Sequences and Gene Loadings")

### Summary: Compared to PCA.1, there is a decrease in the proportion of variance explained by the firs
```

PCA(4) Alternative Analysis

What happens if we exclude the potential outliers from PCA, and then predict their scores and loadings?

```
#(1) Create df excluding outliers identified in PCA.3
genes4 <- genes[!grepl("MT724769_1|MN346703_1|MT724770_1|DQ011155_1", genes$Genome),]

#(2) Create df of outliers
outliers <- genes[grepl("MT724769_1|MN346703_1|MT724770_1|DQ011155_1", genes$Genome),]

#(3) Apply PCA using stats::prcomp
```

```
pca4 <- prcomp(genes4[,5:985])
relvar <- pca4$sdev^2 / sum(pca4$sdev^2)
relvar_per <- round(relvar*100,1)

#(4) Prediction of PCs for outliers
pred <- predict(pca4, newdata=outliers)
pca4_pred <- pca4
pca4_pred$x <- rbind(pca4_pred$x, pred)

#-------------------------------------------------------------------------------
#Vizualize scores and loadings
#-------------------------------------------------------------------------------

#(5) Plot of individuals by virus species w/ outliers in shaded bullets
COLOR <- c(1:length(unique(genes$VirusSpecies)))
pc <- c(1,2)
plot(pca4$x[,pc], cex=1, col=COLOR,
     xlab=paste0("PC 1", "(", relvar_per[pc[1]], "%)"),
     ylab=paste0("PC 2", "(", relvar_per[pc[2]], "%"))
points(pred[,pc], pch=16) + abline(h = 0, v=0, lty = 2) +
title("Plot of Sequences and Outliers") + theme(plot.title = element_text(hjust = 0.5))

#(6) Biplot of individuals and variables
fviz_pca_biplot(pca4_pred, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE) +
             ggtitle("Biplot of Sequences and Gene Loadings")

#(7) Biplot of top 20 contributing individuals and variables
fviz_pca_biplot(pca4_pred, col.var = "contrib", gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE, select.ind=list(contrib=20), select.var=list(contrib=20)) +
             ggtitle("Biplot of Sequences and Gene Loadings")

#(8) Clean environment
rm(list=setdiff(ls(), c("genes", "pca1","outliers")))

### Summary: Predicted scores of outliers cluster in the fourth quadrant with other sequences. As in pr
```

MCA of viral accessory genes

Multiple Correspondence Analysis (MCA) for dimension reduction of categorical variables.

```
#(1) Subset data and reformat gene variables as factor
genes_cat <- subset(genes,select=-c(Genome,VirusSpecies,HostGenus))
genes_cat[] <- lapply(genes_cat, as.character)
rownames <- genes$Sequence
genes_cat[,-1] <- lapply(genes_cat[,-1], factor)
genes_cat$Sequence=NULL
rownames(genes_cat) <- rownames
#str(genes_cat)

#(2) Apply MCA using FactoMineR::MCA
mca = MCA(genes_cat, graph = FALSE)
```

```r
# pca_relvar <- pca$sdev^2 / sum(pca$sdev^2)
# pca_relvar_per <- round(pca_relvar*100,1)

#(3) List and summarize MCA results
print(mca)
# summary(mca)
head(mca$ind$coord) #sequence (individuals)
head(mca$var$coord) #genes (variables)

#(4) Screeplot - Variance (Eigenvalues)
#mca$eig
fviz_eig(mca, addlabels = TRUE, ylim = c(0, 25))

#(5) Plots of individuals
fviz_mca_ind(mca, repel=TRUE)

#(6) Plots of MCA variables 1 and 2
fviz_mca_var(mca, repel = TRUE) ##

#(7) Biplot
fviz_mca_biplot(mca, repel = TRUE)
fviz_mca_biplot(mca, repel = FALSE, select.ind=list(contrib=20), select.var=list(contrib=20))

#(8) Clean environment
rm(list=ls())



vtree <- ape::read.nexus("/Users/katietseng/Fernandez Lab Dropbox/Katie Tseng/Mac/Downloads/OPV_aLRT.ne

#(1) Extract coordinates for individual sequences
ind.coord <- pca1$x
rownames(ind.coord) <- 1:nrow(ind.coord)
db <- cbind(genes$VirusSpecies, ind.coord)

db <- as.data.frame(cbind(genes$Genome, db))
db <- rename(db,c('V1'='genome','V2'='virus'))
db$genome <- gsub('_', '.', db$genome)
db$virus <- gsub(' virus', '', db$virus)
db$tip.label <- paste0(db$genome,"_",db$virus)
db <- db %>% dplyr::select(tip.label, everything())


#(2) HCA on a set of dissimilarities for objects being clustered, wherein each object is assigned its o
# clusters <- hclust(dist(db[,2:3]))
clusters <- hclust(dist(db[,4:5]))
plot(clusters)
abline(h = 8, col="red", lty=5)
abline(h = 4, col="blue", lty=5)

#(3) Cluster cut
clusterCut <- cutree(clusters, 6)
table(clusterCut)
```

```r
#(4) Prop tables by virus species
# mytable<-table(clusterCut, genes$VirusSpecies)
mytable<-table(clusterCut, db$tip.label)
mytable<-as.data.frame(table(clusterCut, db$tip.label))
mytable2 <- data.frame(prop.table(mytable,2))
ggplot(mytable2, aes(x = Var2, y = Freq, fill = clusterCut)) +
  geom_col() +
  labs(fill='Cluster') +
  theme(axis.title.x = element_blank(), axis.text.x = element_text(angle=45,hjust=1)) +
  ggtitle("Distribution by Virus Species")
```

2. Data Preparation

Load packages and data for merging: i.e., Host-pox/viral traits (PC), taxonomy data, host traits, and host tree

```r
#(1) Load treespace
## treespace dependencies include XQuartz v2.7.11 (https://www.xquartz.org/releases/XQuartz-2.7.11.html
## recommend installing and loading rgl including 'options(rgl.useNULL=TRUE)' below before loading tree
library(rgl) # > install.packages("rgl"); > options(rgl.useNULL=TRUE)
library(treespace)

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

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

#(4) Load raw data
load("Data_raw.RData")
load("Output/wgs_PCs.RData")

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

#(6) Viral traits data (& Host-pox links): host-OPV linked interactions & principal components of OPV g
wgsdata <- wgs_PCs

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

#(8) 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

#(9) 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

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

Merge host-OPV interactions from Virion with host-OPV intx from genome annotations (Stephanie)

```
### Clean up Virion data ###

#(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"),]
#In this model, we keep vaccinia virus!

#(2) Aggregate data at the host genus level
poxdata$link <- 1
poxdata <- aggregate(link~HostGenus+Virus, data=poxdata, mean)
poxdata$link=NULL

#(3) Rename and reformat variables
poxdata <- plyr::rename(poxdata,c('HostGenus'='genus','Virus'='virus'))
poxdata$genus <- str_to_title(poxdata$genus)

#(4) Add datasource variable
poxdata$virion <- 1

### Clean up genomic data ###

#(5) Rename and reformat variables
wgsdata <- plyr::rename(wgsdata,c('HostGenus'='genus','Sequence'='sequence'))
wgsdata$virus <- tolower(wgsdata$VirusSpecies)
wgsdata$VirusSpecies=NULL

#(6) Drop genome variable
wgsdata$Genome=NULL

#(7) Add datasource variable
wgsdata$wgs <- 1

### Merge ###

#(8) Are their virus in wgsdata that are not in poxdata and vice versa?
wgsdata$virus[!wgsdata$virus %in% poxdata$virus]
poxdata$virus[!poxdata$virus %in% wgsdata$virus]

#(9) Correct 'cetaceanpox virus' to 'cetacean pox virus 1' in wgsdata
wgsdata$virus <- ifelse(wgsdata$virus=="cetaceanpox virus", "cetacean poxvirus 1", wgsdata$virus)

#(10) Are their genera in wgsdata that are not in poxdata and vice versa?
wgsdata$genus[!wgsdata$genus %in% poxdata$genus]
```

```
poxdata$genus[!poxdata$genus %in% wgsdata$genus]

#(11) Before merging, get count of unique host-virus associations in each dataset
poxdata_unique <- unique(poxdata[ , c("genus", "virus")])
length(unique(poxdata_unique$genus))
length(unique(poxdata_unique$virus))
wgs_unique <- unique(wgsdata[ , c("genus", "virus")])
length(unique(wgs_unique$genus))
length(unique(wgs_unique$virus))

#(12) Merge poxdata w/ wgsdata
poxdata <- merge(poxdata,wgsdata,by=c('virus','genus'),all=TRUE)

#(13) Create data source variable
poxdata$source <- ifelse(poxdata$wgs==1 & is.na(poxdata$virion),"wgs", ifelse(poxdata$virion==1 & is.na

#(14) Replace sequence==NA with "NA", as this var will be our unique identifier for WGS links
poxdata$sequence <- ifelse(is.na(poxdata$sequence),"NA",poxdata$sequence)

#(15) Get count of unique host-virus associations in each dataset
poxdata_unique <- unique(poxdata[ , c("genus", "virus")])
length(unique(poxdata_unique$genus))
length(unique(poxdata_unique$virus))

#(16) Remove virion and wgs vars; clean environment
poxdata$virion=NULL
poxdata$wgs=NULL
rm(wgsdata, wgsdata_unique, poxdata_unique)
```

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')]
gtaxa <- plyr::rename(gtaxa, c('gen'='genus'))

#(2) Check for mismatched genus names between poxdata and taxa before merging poxdata with taxa
poxdata$genus[!poxdata$genus %in% gtaxa$genus]
poxdata <- merge(gtaxa,poxdata,by='genus',all.x=TRUE)

#(3) To keep only genera from orders in which positive associations exist, first subset known host-viru
keep <- subset(poxdata, !is.na(poxdata$source))

#(4) Next, create a new variable <keep> in poxdata denoting observations with the same host taxonomic o
poxdata$keep <- ifelse(poxdata$ord %in% keep$ord,TRUE,FALSE)

#(5) View/evaluate which taxonomics orders will be kept and which orders will be discarded
uniq <- unique(poxdata[c("ord","keep")])

#(6) Keep only observations with the same host taxonomic order as that of known associations
poxdata <- subset(poxdata,keep==TRUE)
```

```
poxdata$keep=NULL

#(7) Create binary variable for known host-OPV associations/links
poxdata$link=ifelse(is.na(poxdata$source),0,1)

#(8) Reorder variables
poxdata <- poxdata %>% select(link, virus, genus, fam, ord, source, everything())

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

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: 0=above ground, 1=fossorial dwelli
hostTraits_binary=aggregate(cbind(hibernation_torpor,fossoriality,freshwater,marine,terrestrial_non.vola
hostTraits_binary_temp=aggregate(cbind(hibernation_torpor,fossoriality,freshwater,marine,terrestrial_no

#(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

```r
#(1) Check if all poxdata genera are in hostTree
poxdata$gtip <- poxdata$genus
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://ww
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$tree

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

#(4) Trim hostTree (remove species tip) 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'))
```

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

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

## consider adding viral genome length, viral richness (number of virus detected in each genera), and h
```

Add all possible host-OPV combinations for link prediction model

```r
#(1) Create a dataframe of all possible host-OPV combinations (for mammal genera that exist in orders w
uniq_virus <- unique(poxdata$virus[!is.na(poxdata$virus)])
uniq_genus <- unique(poxdata$genus[!is.na(poxdata$genus)])
combinations <- expand.grid(uniq_virus, uniq_genus)
combinations <- plyr::rename(combinations,c('Var1'='virus','Var2'='genus'))

#(2) Create two dataframes: one subsetting host-OPV interaction/link data from poxdata (excluding host
linkdata <- poxdata[,grepl("link|virus|genus|source|sequence|PC", names(poxdata))]
hostTraits <- poxdata[,!grepl("link|virus|source|sequence|PC", names(poxdata))] # include genus b/c we
hostTraits <- hostTraits[!duplicated(hostTraits$genus),]

#(3) Merge linkdata with all possible combinations (drop observations if virus NA)
linkdata <- merge(linkdata, combinations, by=c("virus","genus"),all=TRUE)
linkdata <- linkdata[!is.na(linkdata$virus),]

#(4) Merge linkdata with hostTraits
linkdata <- merge(linkdata, hostTraits, by=c("genus"))

#(5) Reorder variables
linkdata <- linkdata %>% select(source, sequence, link, virus, genus, fam, ord, gtip, traitname, treenam

#(6) Reclassify NAs as pseudo-absences for viral detection
linkdata$link=ifelse(is.na(linkdata$link),0,linkdata$link)

### Next, we need to impute values of PC variables where data exist for that virus.

#(7) First let's subset virus and PC variables, and drop rows w/ NA
pc <- linkdata[,grepl("virus|PC", names(linkdata))]
pc <- pc[!is.na(pc$PC1),]

#(8) Let's take a quick look at which virus do not have PC data
setdiff(unique(linkdata$virus),unique(pc$virus))

#(9) Next, we obtain the median value of each PC variable for each virus using aggregate function (dot
pc_median=aggregate(. ~virus, data=pc, FUN=median)
colnames(pc_median)[c(-1)] <- paste(colnames(pc_median)[c(-1)], "_med", sep="")

#(10) Merge PC median variables w/ poxdata
linkdata <- merge(linkdata,pc_median,by=c("virus"),all.x=TRUE)

#(11) Get PC colnames and replace NA values of PC vars w/ median PC values using for-loop
```

```
pc_names <- colnames(pc)[c(-1)]
for(i in pc_names) {
  linkdata[,i] <- ifelse(is.na(linkdata[,i]),linkdata[,paste(i,"_med",sep="")],linkdata[,i])
}

#(12) Drop median PC variables
pc_med_names <- colnames(pc_median)[c(-1)]
for(i in pc_med_names) {
  linkdata[,i]=NULL
}

#(13) Replace NA values for 'source' and 'sequence' with "NA"
linkdata$source <- ifelse(is.na(linkdata$source),"NA",linkdata$source)
linkdata$sequence <- ifelse(is.na(linkdata$sequence),"NA",linkdata$sequence)

#(14) Clean environment
rm(combinations, hostTraits, uniq_virus, uniq_genus, pc, pc_median, pc_names, pc_med_names, i, poxdata)
```

Save cleaned data

```
#(1) Reorder variables
linkdata <- linkdata %>% select(source, sequence, link, virus, genus, fam, ord, gtip, traitname, treenar

#(2) Save dataframes for analysis **poxdata_temp.RData is for practice analysis**
save(linkdata, hostTree, file='Output/LinkData_clean.RData')
```

3. Boosted Regression Trees (BRT)

Note: The following section is coded to run on your local computer and will likely take a minimum of 72 hours (with parallel processing on 5 cores), outputting an RData file size of 2GB. For tips on running the code on an available HPC node, see 'Tseng2022/HPC Example' on the PoxHost GitHub repository.

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)
###to install ggtree, need to first install BiocManager:
# if (!requireNamespace("BiocManager", quietly = TRUE))
```

```
#     install.packages("BiocManager")
# BiocManager::install("ggtree")
library(ggtree)
library(treeio)
library(caret)
library(InformationValue)
library(mgcv)

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

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

Create taxonomic variables as predictors for the model

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

#(2) Calculate number of pseudoabsences
> length(which(linkdata$link==0))

#(2) Ensure all accessory gene variables are numeric
PC_columns <- colnames(data[which(grepl("PC",names(data)))])
data[,c(PC_columns)] <- lapply(data[c(PC_columns)],as.numeric)
str(data)
#
# #(1) Classify true negatives
# data$type=ifelse(data$pcr==0 & data$competence==0,"true negative","other")
#
# #(2) Which species is competent but no PCR record?
# set=data
# set$treename[set$pcr==0 & set$competence==1]
#
# #(3) 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)

#(4) Make binary variables for each taxonomic family; remove any duplicates
dums=dummy_cols(data["fam"])
dums=dums[!duplicated(dums$fam),]

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

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

#(7) Drop unnecessary columns and clean environment
data$traitname=NULL
rm(dums, i, PC_columns)
```

Assess variation and availability of data

```
#(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','source','sequence','link','virus','genus','ord','gtip','treenam
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$variables=rownames(mval)
names(mval)=c("comp","column")
#
#(9) visualize distribution of NA
# png("Figure S1.png", width=4,height=4,units="in",res=600)
# ggplot(mval[!mval$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")
mval$keep=ifelse(mval$comp>=0.60,"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))

set <- subset(data,select=-c(fam, source, sequence, virus, genus, ord, gtip, treename))
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
write.csv(ts1, "Output/brt_var_t1.csv")

#(17) Check that binary variables are numeric and not factor (except for fam vars)
str(set)

#(18) Remove vars not needed for BRT analysis
data$source=NULL
data$sequence=NULL
# save(data, file='Output/data_LinkBRT.RData')
```

Alternative dataset excluding vaccinia virus sequences

```
# #(1) Subset data to exclude host-vaccinia virus links
# data <- data[data$virus!="vaccinia virus",]
# set <- subset(data,select=-c(fam, virus, genus, ord, gtip, treename))
```

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

```
#(1) Hyperparameter tuning ifelse
#hok="ok"
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))
  # hgrid=expand.grid(n.trees=500,                                  #creates df from all combinations of fa
  #                   interaction.depth=c(2,3,4),
  #                   shrinkage=c(0.1,0.01,0.005),
  #                   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
    ndata$link=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 link
hpars=lapply(1:nrow(hgrid),function(x) hfit(x,response="link"))

## 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"))

search=hsearch
```

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

}else{

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

}
```

Model tuning results: Figure S2

```
search=read.csv("/Users/katietseng/Fernandez Lab Dropbox/Katie Tseng/Mac/Downloads/HPC_24Nov2022_Model1/

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

#(10) Visualize - Figure S2
png("Output/brt_tuning_results_f1.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")+
  facet_grid(search2$measure,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 shrinkage==

#(12) Plot best.iter by type (pcr/competence) to see max number of trees to include
plotA <- search_nt5000 %>%
  # ggplot( aes(x=best, fill=type)) +
  ggplot( aes(x=best)) +
```

```
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  ggtitle("(A) ntrees=5000 and shrinkage=0.001")
plotB <- search_nt15000 %>%
  # ggplot( aes(x=best, fill=type)) +
  ggplot( aes(x=best)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080"))  +
  ggtitle("(B) ntrees=15000 and shrinkage=0.001")
plotC <- search_nt5000_sh0.01 %>%
  # ggplot( aes(x=best, fill=type)) +
  ggplot( aes(x=best)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity') +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  ggtitle("(C) ntrees=5000 and shrinkage=0.01")

png("Output/brt_tuning_results_f2.png",width=12,height=4,units="in",res=600)
plot_grid(plotA,plotB,plotC) +
  ggtitle("Histogram of best iteration")
dev.off()

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

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
    ndata$link=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=="link",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=data[c("virus","gtip",'treename',"fam","ord","link")]
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"))
brts=lapply(1:smax,function(x) brt_part(seed=x,response="link"))

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

#(3) save results to wd
save(brts,pm_brts,file="Output/LinkData_results.RData")
```

4. BRT Figures

Note: Model tuning and model predictions were run twice on HPC. Once with vaccinia virus links included (Model 1) and a second time excluding vaccinia virus links (Model 2).

Load required packages and set system

Evaluate distribution of MPXV sequences in train and test datasets from Model 1 results using 'cites' variable as proxy

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

# load("Output/LinkData_results.RData")

#(1) Get data from Model 1
load("~/Fernandez Lab Dropbox/Katie Tseng/Mac/Downloads/HPC_24Nov2022_Model1/Output/brts_Model1.RData")
brts1=brts
rm(brts)

#(2) Get data from Model 2
load("~/Fernandez Lab Dropbox/Katie Tseng/Mac/Downloads/HPC_24Nov2022_Model2/Output/brts_Model2.RData")
brts2=brts
rm(brts)
```

```
# ####KATIE: Predictions with 'feline poxvirus ita2_bc' as known associations (i.e., with Homo) should
# roc_temp <- roc(brts1[[1]]$predict$link,brts1[[1]]$predict$cpred, auc=TRUE, ci=TRUE)
# print(roc_temp$auc)
# print(brts1[[1]]$testAUC)

#(3) Row-bind predicted values of all iterations for Model 2
model2=lapply(brts2,function(x) x$predict)
model2=do.call(rbind,model2)

#(4) Build an ROC curve from the known links and the predicted probs for Model 1 and Model 2
roc1 <- roc(model1$link,model1$cpred, auc=TRUE, ci=TRUE)
roc2 <- roc(model2$link,model2$cpred, auc=TRUE, ci=TRUE)

# extract cites from brts1 train data
ctrain=lapply(brts1,function(x) x$traindata$cites)
ctrain=do.call(rbind, ctrain)

# convert to matrix
cmat_train = as.matrix(ctrain)
cmat_train = t(cmat_train)

# change colnames
colnames(cmat_train) = paste("iter", seq(ncol(cmat_train)), sep="")

# convert to dataframe
dat_train = as.data.frame(cmat_train)
dat_train$obs = rownames(dat_train)

# reshape wide to long
mdat_train = reshape2::melt(dat_train, id.vars="obs")
# mdat_train$obs = as.numeric(gsub("obs", "", mdat_train$variable))
mdat_train$iter = as.factor(mdat_train$variable)
mdat_train$log10_cites = as.numeric(log10(mdat_train$value))
mdat_train$obs = as.numeric(mdat_train$obs)

# plot as line graph by iteration
p_train = ggplot(mdat_train, aes(x=obs, y=log10_cites, color=iter)) +
    theme_bw() + theme(panel.grid=element_blank()) +
    theme(legend.position="none") +
    geom_line(linewidth=0.2, alpha=0.1) +
    stat_summary(aes(y = log10_cites), fun=mean, color="black", geom="line", linewidth=0.2, alpha=0.4)

# save plot
png("Output/perf_mpxv_f1.png",width=20,height=15,units="in",res=300)
p_train
dev.off()

# extract cites from brts1 train data
ctest=lapply(brts1,function(x) x$testdata$cites)
ctest=do.call(rbind, ctest)

# convert to matrix
cmat_test = as.matrix(ctest)
```

```
cmat_test = t(cmat_test)

# change colnames
colnames(cmat_test) = paste("iter", seq(ncol(cmat_test)), sep="")

# convert to dataframe
dat_test = as.data.frame(cmat_test)
dat_test$obs = rownames(dat_test)

# reshape wide to long
mdat_test = reshape2::melt(dat_test, id.vars="obs")
# mdat_train$obs = as.numeric(gsub("obs", "", mdat_train$variable))
mdat_test$iter = as.factor(mdat_test$variable)
mdat_test$log10_cites = as.numeric(log10(mdat_test$value))
mdat_test$obs = as.numeric(mdat_test$obs)

# plot as line graph by iteration
p_test = ggplot(mdat_test, aes(x=obs, y=log10_cites, color=iter)) +
    theme_bw() + theme(panel.grid=element_blank()) +
    theme(legend.position="none") +
    geom_line(linewidth=0.2, alpha=0.1) +
    stat_summary(aes(y = log10_cites), fun=mean, colour="black", geom="line", linewidth=0.2, alpha=0.4)

# save plot
png("Output/perf_mpxv_f2.png",width=20,height=15,units="in",res=300)
p_test
dev.off()

#() replot for iter 3 for train and test data
png("Output/perf_mpxv_f3.png",width=20,height=15,units="in",res=300)
ggplot(mdat_train[mdat_train$iter=="iter1" | mdat_train$iter=="iter2" | mdat_train$iter=="iter3" | mdat_
    theme_bw() + theme(panel.grid=element_blank()) +
    geom_line(linewidth=0.2, alpha=1)
dev.off()
png("Output/perf_mpxv_f4.png",width=20,height=15,units="in",res=300)
ggplot(mdat_test[mdat_test$iter=="iter1" | mdat_test$iter=="iter2" | mdat_test$iter=="iter3" | mdat_tes
    theme_bw() + theme(panel.grid=element_blank()) +
    geom_line(linewidth=0.2, alpha=1)
dev.off()

# clean environment
rm(cmat_test, cmat_train, ctest,ctrain, dat_test, dat_train, mdat_test, mdat_train, p_test, p_train)
```

Evaluate performance measures for Model 1 (all links) and Model 2 (excluding Vaccinia)

How accurately did Model 1 and Model 2 distinguish host-virus links vs. non-links?

```
#(3) Index non-missing
keep1=which(!is.na(sapply(brts1,function(x) x$testAUC)))
keep2=which(!is.na(sapply(brts2,function(x) x$testAUC)))

#(4) Trim, keeping only those that are non-missing
```

```
brts1=brts1[keep1]
brts2=brts2[keep2]

### Get model performance measures from BRT results, whereby threshold value (THV) = 0.5
### Save values for Supp Table 4

#(5) Get AUC for Model 1
auc1 <- paste0(round(mean(sapply(brts1,function(x) x$testAUC)),3), " (", round(std.error(sapply(brts1,fu

#(6) Get sensitivity for Model 1
sens1 <- paste0(round(mean(sapply(brts1,function(x) x$sen)),3)," (", round(std.error(sapply(brts1,functi

#(7) Get specificity for Model 1
spec1 <- paste0(round(mean(sapply(brts1,function(x) x$spec)),3), " (", round(std.error(sapply(brts1,func

#(8) Get AUC for Model 2
auc2 <- paste0(round(mean(sapply(brts2,function(x) x$testAUC)),3), " (", round(std.error(sapply(brts2,fu

#(9) Get sensitivity for Model 2
sens2 <- paste0(round(mean(sapply(brts2,function(x) x$sen)),3), " (", round(std.error(sapply(brts2,funct

#(10) Get specificity for Model 2
spec2 <- paste0(round(mean(sapply(brts2,function(x) x$spec)),3), " (", round(std.error(sapply(brts2,func

# #(11) Get AUC for Cites Model 1
# paste0(round(mean(sapply(pm_brts1,function(x) x$testAUC)),3), " (", round(std.error(sapply(pm_brts1,f

# #(12) Get AUC for Cites Model 2
# paste0(round(mean(sapply(pm_brts2,function(x) x$testAUC)),3), " (", round(std.error(sapply(pm_brts2,f

rm(keep1, keep2)

### How do the model performances of BRTs trained on all links (Model 1) vs. links excluding Vaccinia (

#(1) Function to compare significant difference between two means (t-test) and the measured difference
tfun=function(measure){

  ## format data
  n=length(sapply(brts1,function(x) x$measure))
  adata=data.frame(y=c(sapply(brts1,function(x) x[measure][[1]]),
                       sapply(brts2,function(x) x[measure][[1]])),
                   response=c(rep('model1',n),rep('model2',n)),
                   seed=c(sapply(brts1,function(x) x$seed),
                          sapply(brts2,function(x) x$seed)))
  rm(n)

  ## factor
  adata$model=factor(adata$response,levels=c('model1','model2'))

  ## 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("model1"="Full Model",
                                                 "model2"="Partial Model"))

  ## 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 effect size of AUC between models; extract t-stat and Cohen's d
adata=tfun("testAUC")
adata$tsum$statistic
adata$csum$effsize

#(3) Compare effect size of sensitivity between models; extract t-stat and Cohen's d
sedata=tfun("sen")
sedata$tsum$statistic
sedata$csum$effsize

#(4) Compare effect size of specificity between models; extract t-stat and Cohen's d
spdata=tfun("spec")
spdata$tsum$statistic
spdata$csum$effsize

#(5) Adjust p values
ps=c(adata$tsum$p.value, sedata$tsum$p.value, spdata$tsum$p.value)
ps_adjust <- round(p.adjust(ps,method="BH"),4)
###"BH" (aka "fdr") = Benjamini & Hochberg (1995) method controls the false discovery rate, the expecte

#(6) Create and save table of performance measures
ts4 <- data.frame(matrix(c(auc1, sens1, spec1,
                auc2, sens2, spec2,
                round(adata$tsum$statistic,2), round(sedata$tsum$statistic,2), round(spdata$tsum$statis
                ps_adjust,
                round(adata$csum$effsize,2), round(sedata$csum$effsize,2), round(spdata$csum$effsize,2)
            ncol=3, byrow=TRUE))
colnames(ts4) <- c('mean_AUC (SE)', 'mean_sens (SE)', 'mean_spec (SE)')
rownames(ts4) <- c('Model 1', 'Model 2', 't-test', 'p-val', "Cohen's d")
write.csv(ts4, "Output/perf_auc_t1.csv")

#(9) Clean environment
# rm(pm_brts, keep1, keep2)
rm(tfun, ts4, auc1, sens1, spec1, auc2, sens2, spec2, ps, ps_adjust)
```

What if we optimize the threshold level based on max(sensitivity + specificity)? How do our performance measures change?

```
### Get optimized THV from predicted probabilities when combined across all 100 iterations

#(1) load library
library(pROC)

#(2) Row-bind predicted values of all iterations for Model 1
model1=lapply(brts1,function(x) x$predict)
model1=do.call(rbind,model1)

#(3) Row-bind predicted values of all iterations for Model 2
model2=lapply(brts2,function(x) x$predict)
model2=do.call(rbind,model2)

#(4) Build an ROC curve from the known links and the predicted probs for Model 1 and Model 2
roc1 <- roc(model1$link,model1$cpred, auc=TRUE, ci=TRUE)
roc2 <- roc(model2$link,model2$cpred, auc=TRUE, ci=TRUE)

#(5) Plot ROC curve for Model 1
png("Output/perf_thv_f1.png",width=8,height=6,units="in",res=300)
plot(roc1, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),
  grid.col=c("gray", "gray"), max.auc.polygon=TRUE,
  auc.polygon.col="lightblue", print.thres=TRUE) #print.thres based on max(sens+spec)
dev.off()

#(6) Plot ROC curve for Model 2
plot(roc2, print.auc=TRUE, auc.polygon=TRUE, grid=c(0.1, 0.2),
  grid.col=c("gray", "gray"), max.auc.polygon=TRUE,
  auc.polygon.col="lightblue", print.thres=TRUE)

#(7) Get optimum THV by finding THV when sens+spec are maximized for Model 1
tss <- cbind(roc1$thresholds, roc1$sensitivities + roc1$specificities)
thv1 <- subset(tss, tss[,2]==max(tss[,2]))[,1]

#(8) Get optimum THV by finding THV when sens+spec are maximized for Model 2
tss <- cbind(roc2$thresholds, roc2$sensitivities + roc2$specificities)
thv2 <- subset(tss, tss[,2]==max(tss[,2]))[,1]

### Get optimized THV averaged across all 100 model iterations based on predicted probs

#(9) Build an ROC curve for each iteration of Model 1 and Model 2
roc1=lapply(brts1,function(x) roc(x$predict$link,x$predict$cpred, auc=TRUE,ci=TRUE))
roc2=lapply(brts2,function(x) roc(x$predict$link,x$predict$cpred, auc=TRUE,ci=TRUE))

#(10) Get mean(SE) optimum THV by finding THV when sens+spec are max for Model 1
tss=lapply(roc1, function(x) cbind(x$thresholds, x$sensitivities + x$specificities))
thv_mean = lapply(tss, function(x) subset(x, x[,2]==max(x[,2]))[,1])
thv_mean1 = mean(sapply(thv_mean, function(x) x))
thv_se1 =se(sapply(thv_mean, function(x) x))

#(11) Get mean(SE) optimum THV by finding THV when sens+spec are max for Model 2
```

```r
tss=lapply(roc2, function(x) cbind(x$thresholds, x$sensitivities + x$specificities))
thv_mean = lapply(tss, function(x) subset(x, x[,2]==max(x[,2]))[,1])
thv_mean2 = mean(sapply(thv_mean, function(x) x))
thv_se2 =se(sapply(thv_mean, function(x) x))

#(12) Plot ROC curves for Model 1
png("Output/perf_thv_f2.png",width=8,height=6,units="in",res=300)
plot(roc1[[1]])
for (i in 2:length(roc1)) {
   plot(roc1[[i]], add=TRUE, grid=c(0.1, 0.2),
  grid.col=c("gray", "gray"))
}
dev.off()

#(13) Plot ROC curves for Model 2
plot(roc2[[1]])
for (i in 2:length(roc1)) {
   plot(roc2[[i]], add=TRUE, grid=c(0.1, 0.2),
  grid.col=c("gray", "gray"))
}

#(14) View coordinates of opt-THVs and extract sens and spec values for Model 1
coords1 = lapply(roc1, function(x) coords(x, thv_mean1, input="threshold"))
coords_sens1 = unlist(lapply(coords1, function(x) x$sensitivity))
coords_spec1 = unlist(lapply(coords1, function(x) x$specificity))

#(15) Calculate corresponding mean sens and spec for Model 1
mean(coords_sens1)
std.error(coords_sens1)

#(16) View coordinates of opt-THVs and extract sens and spec values for Model 2
coords2 = lapply(roc2, function(x) coords(x, thv_mean2, input="threshold"))
coords_sens2 = unlist(lapply(coords2, function(x) x$sensitivity))
coords_spec2 = unlist(lapply(coords2, function(x) x$specificity))

#(17) Calculate corresponding mean sens and spec for Model 2
mean(coords_spec2)
std.error(coords_spec2)

# #(18) Alternative approach for calculating mean sens and spec at opt-THV
# model1=lapply(brts1,function(x) x$predict)
# sens=unlist(lapply(model1, function(x) InformationValue::sensitivity(x$link,x$cpred,threshold=opt_mea
# spec=unlist(lapply(model1, function(x) InformationValue::specificity(x$link,x$cpred,threshold=opt_mea
# mean(sens)
# mean(spec)

#(19) Calculate mean AUC of Model 1
auc=unlist(lapply(roc1, function(x) x$auc))
mean(auc)
std.error(auc)

#(19) Calculate mean AUC of Model 2
auc=unlist(lapply(roc2, function(x) x$auc))
```

```
mean(auc)
std.error(auc)

### Get optimum THV based on the ROC curve: maximize TRP and FPR (true/false positive rate)
### by minimizing |FPR + TPR - 1|, which is equal to minimizing |sensitivity - specificity|

# Unlist and extract sensitivity, specificity and threshold values
sensitivities <- unlist(lapply(roc1, function(x) x$sensitivities))
specificities <- unlist(lapply(roc1, function(x) x$specificities))
thresholds <- unlist(lapply(roc1, function(x) x$thresholds))

# Combine values into df
df <- as.data.frame(cbind(sensitivities, specificities, thresholds))
colnames(df) <- c("sens", "spec", "thresh")
df <- subset(df, is.finite(df$thresh))

# create variable = |sensitivity - specificity|
df$sens_spec <- abs(df$sens - df$spec)

#plot sens_spec to see threshold level at minimum
ggplot(df, aes(thresh, sens_spec)) + geom_line()

#get values
df[which.min(df$sens_spec),]
thv3 <- df[which.min(df$sens_spec),3]

#(20) Clean environment
rm(tss, roc1, roc2, coords1, coords_sens1, coords_spec1, coords2, coords_sens2, coords_spec2)
```

###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,sedata,spdata)

#(3) Figure S3 - boxplot of model performance for supplement
png("Output/perf_boxplot_f1.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$response1),
                     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=c("Full Model","Partial Model"),
                     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 and rank features by relative importance: Table S5 (Model 1) and Table S6 (Model 2)

```
### Identify relative feature importance ###

#(1) Relative importance for Model 1
vinf=lapply(brts1,function(x) x$rinf)
vinf1=do.call(rbind,vinf)

#(2) Relative importance for Model 2
vinf=lapply(brts2,function(x) x$rinf)
vinf2=do.call(rbind,vinf)

#() Create function for calculating standard error
se <- function(x) {
  sqrt(var(x) / length(x))
}

#(3) Aggregate mean, SE, var for Model 1
```

```
vdata1=data.frame(aggregate(rel.inf~var,data=vinf1,mean),
                  aggregate(rel.inf~var,data=vinf1,se)["rel.inf"],
                  aggregate(rel.inf~var,data=vinf1,var)["rel.inf"])
names(vdata1)=c("var","rel.inf","rse","rvar")
vdata1=vdata1[order(vdata1$rel.inf,decreasing=T),]

#(4) Aggregate mean, SE, var for Model 2
vdata2=data.frame(aggregate(rel.inf~var,data=vinf2,mean),
                      aggregate(rel.inf~var,data=vinf2,se)["rel.inf"],
                      aggregate(rel.inf~var,data=vinf2,var)["rel.inf"])
names(vdata2)=c("var","rel.inf","rse","rvar")
vdata2=vdata2[order(vdata2$rel.inf,decreasing=T),]

#(5) Compare mean var
paste0("mean Relative Importance for Model 1: ", mean(vdata1$rvar))
paste0("mean Relative Importance for Model 2: ", mean(vdata2$rvar))

#(6) Compare variance in mean var
paste0("variance in mean Relative Importance for Model 1: ", var(vdata1$rel.inf))
paste0("variance in mean Relative Importance for Model 2: ", var(vdata2$rel.inf))

### Rank features by relative importance: Table S5 (Model 1) and Table S6 (Model 2) ###

#(1) Rank for Model 1 and Model 2
vdata1$rank1=1:nrow(vdata1)
vdata2$rank2=1:nrow(vdata2)

#(2) Relative infuence
vdata1$imp1=vdata1$rel.inf/100
vdata2$imp2=vdata2$rel.inf/100

#(3) Combine ranks
ranks=merge(vdata1[c("var","rank1","imp1")],
            vdata2[c("var","rank2","imp2")],
            by="var")

#(4) Table S5 - Ranks for Model 1
ts=ranks
ts$feature=ts$var
ts5=ts[c("feature","imp1","rank1")]
ts5=ts5[order(ts5$rank1),]
write.csv(ts5,"Output/trait_rank_t1.csv")
ts5

#(5) Table S6 - Ranks for Model 2
ts6=ts[c("feature","imp2","rank2")]
ts6=ts6[order(ts6$rank2),]
write.csv(ts6,"Output/trait_rank_t2.csv")
ts6

#(6) Extract list of top 10 variables
keep1 <- ranks$var[which(ranks$rank1<=10)]
keep2 <- ranks$var[which(ranks$rank2<=10)]
```

```
#(7) Subset df of relative importance values associated with top 10 vars
vinf1 <- vinf1[which(vinf1$var%in%keep1),]
vinf1$rel.inf <- vinf1$rel.inf/100

vinf2 <- vinf2[which(vinf2$var%in%keep2),]
vinf2$rel.inf <- vinf2$rel.inf/100

#(8) Boxplot of Relative Feature Importance - Model 1
fs2A <- ggplot(vinf1) + ggtitle("(C) Link prediction 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))

#(8) Boxplot of Relative Feature Importance - Model 2
fs2B <- ggplot(vinf2) + ggtitle("(D) Link prediction model (no vaccinia virus)") +
        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 Figure S2A and S2B
png("Output/trait_rank_f1.png",width=10,height=10,units="in",res=300)
ggarrange(fs2A,fs2B,ncol=2,nrow=1,
          font.label=list(face="plain",size=12))
dev.off()

#Note: Island dwelling = 20% or more of the breeding range occurs on an island
# https://stats.stackexchange.com/questions/422458/gbm-how-to-interpret-relative-variable-influence

rm(vinf, vinf1, vinf2, keep1, keep2, vdata1, vdata2, ts, ts5, ts6)
```

Identify consistently important/unimportant host traits: Figure 2B

```
#(1) Correlate: Were the rankings of relative feature importance sig. correlated?
cor.test(ranks$rank1,ranks$rank2,method="spearman")

#(2) Trim to non-zero and rerank
ranks2=ranks[-which(ranks$imp1==0 & ranks$imp2==0),]
ranks2=ranks2[order(ranks2$imp1,decreasing=T),]
```

```
ranks2$rank1=1:nrow(ranks2)
ranks2=ranks2[order(ranks2$imp2,decreasing=T),]
ranks2$rank2=1:nrow(ranks2)

#(3) Correlate: Were the rankings still correlated after removing traits with zero/no relative importan
cor.test(ranks2$rank1,ranks2$rank2,method="spearman")

#(4) Identify features with high residuals and plot
ranks2$resid=abs(resid(lm(rank2~rank1,data=ranks2)))  # extract residuals from linear regression as abs

#(5) Plot residuals
plot(ranks2$rank1,ranks2$resid,
     ylab="Residuals",xlab="Traits by ranking for Model 1",
     main="Model 1 with variables with zero/no relative importance removed")

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

#(7) Flag if consistently low or consistently high
n=7
ranks2$select=ifelse(ranks2$rank2<n & ranks2$rank1<n,"yes",ranks2$select)
ranks2$select=ifelse(ranks2$rank2%in%tail(1:nrow(ranks2),n) & ranks2$rank1%in%tail(1:nrow(ranks2),n),"y

#(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(rank1,rank2))+
  #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$rank2,ranks2$rank1))+4,0))+
  scale_x_reverse(limits=c(max(c(ranks2$rank2,ranks2$rank1))+4,0))+
```

```
  #geom_abline(slope=1,linetype=2,size=0.5)+
  theme_bw()+
  labs(x="Feature rank for Model 1 (Full) ",
       y="Feature rank for Model 2 (Partial)")+
  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/trait_consistent_f1.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
library(fastDummies)
load("Output/LinkData_clean.RData")
data <- linkdata

#(5) Make binary columns for family
dums=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 ranking - Model 1
ranks2=ranks2[order(ranks2$rank1),]
a1=pdp_plot(brts1,ranks2$var[1]) #118
a2=pdp_plot(brts1,ranks2$var[2])
a3=pdp_plot(brts1,ranks2$var[3])
a4=pdp_plot(brts1,ranks2$var[4])
a5=pdp_plot(brts1,ranks2$var[5])
a6=pdp_plot(brts1,ranks2$var[6])
a7=pdp_plot(brts1,ranks2$var[7])
a8=pdp_plot(brts1,ranks2$var[8])
a9=pdp_plot(brts1,ranks2$var[9])
a10=pdp_plot(brts1,ranks2$var[10])

#(10) Top ranking - Model 2
ranks2=ranks2[order(ranks2$rank2),]
b1=pdp_plot(brts2,ranks2$var[1])
b2=pdp_plot(brts2,ranks2$var[2])
b3=pdp_plot(brts2,ranks2$var[3])
b4=pdp_plot(brts2,ranks2$var[4])
```

```
b5=pdp_plot(brts2,ranks2$var[5])
b6=pdp_plot(brts2,ranks2$var[6])
b7=pdp_plot(brts2,ranks2$var[7])
b8=pdp_plot(brts2,ranks2$var[8])
b9=pdp_plot(brts2,ranks2$var[9])
b10=pdp_plot(brts2,ranks2$var[10])

#(11) Figure S4 - Compile top ranked features by PCR and competence models
library(patchwork)
m1_pdp_plots <- a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+plot_layout(nrow=10,ncol=1,byrow=F)
m2_pdp_plots <- b1+b2+b3+b4+b5+b6+b7+b8+b9+b10+plot_layout(nrow=10,ncol=1,byrow=F)
png("Output/trait_effect_f1.png",width=4,height=10,units="in",res=300)
ggarrange(m1_pdp_plots,m2_pdp_plots,ncol=2,nrow=2,widths=c(4,4),heights=c(22,1),
          labels=c("(A) Full Model","(B) Partial Model"),
          label.x=c(0,-0.1), label.y=0.001,
          font.label=list(face="plain",size=12))
# dev.off()

#(12) Clean environment
rm(m1_pdp_plots,m2_pdp_plots,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,
   b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,
   ranks,ranks2,ts5,f2A,f2B,adata,
   pdp_agg,pdp_plot)
```

Model predictions: Extract and save predicted probabilities

```
#(1) Average predictions: Model 1
# pcr_apreds=lapply(pcr_brts,function(x) x$predict)
# pcr_apreds=do.call(rbind,pcr_apreds)
model1_apreds=lapply(brts1,function(x) x$predict)
model1_apreds=do.call(rbind,model1_apreds)

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

model1_apreds=data.frame(aggregate(pred~treename+virus,data=model1_apreds,mean),
                      aggregate(cpred~treename+virus,data=model1_apreds,mean)['cpred'], ## holding wos
                      aggregate(link~treename+virus,data=model1_apreds,prod)['link'])

#(3) Generate type variable
model1_apreds$type='Model1'

#(4) Average predictions: Model 2
model2_apreds=lapply(brts2,function(x) x$predict)
model2_apreds=do.call(rbind,model2_apreds)

#(5) Aggregate
model2_apreds=data.frame(aggregate(pred~treename+virus,data=model2_apreds,mean),
```

```
                        aggregate(cpred~treename+virus,data=model2_apreds,mean)['cpred'], ## holding wos
                        aggregate(link~treename+virus,data=model2_apreds,prod)['link'])


#(6) Generate type variable
model2_apreds$type='Model2'

#(7) Combine apreds
apreds=rbind.data.frame(model1_apreds,model2_apreds)

#(11) Generate type variable
library(plyr)
apreds$type=factor(apreds$type,levels=c("Model1","Model2"))
apreds$type2=revalue(apreds$type,c("Model1"="FullModel","Model2"="PartialModel"))

#(12) Transform apreds long to wide
apreds2=spread(apreds[c('virus','treename','type','cpred')],type,cpred)
model1_apreds$link1=model1_apreds$link
model2_apreds$link2=model2_apreds$link

#(13) Merge with apreds2
apreds2=merge(apreds2,model1_apreds[c("virus","treename","link1")],by=c("virus","treename"),all=TRUE)
apreds2=merge(apreds2,model2_apreds[c("virus","treename","link2")],by=c("virus","treename"),all=TRUE)

#() Check that datasets merged correctly, links for vaccinia virus should be NA for 'Model2' and 'link2
# View(apreds2[apreds2$virus=="vaccinia virus",])

#(14) Fix names
names(apreds2)=c("virus","treename","pred_Model1","pred_Model2","link1","link2")

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

#(16) Merge with data
apreds2=merge(apreds2,data[c("virus","treename","source","ord","fam","genus")],by=c('virus','treename')
# View(apreds2[duplicated(apreds2),]) #check
apreds2 <- apreds2[!duplicated(apreds2),]
apreds=merge(apreds,data[c("virus","treename","source")],by=c('virus','treename'),all.x=TRUE)
# View(apreds[duplicated(apreds),]) #check
apreds <- apreds[!duplicated(apreds),]

# #(17) Fix source labels
apreds2$cat=ifelse(apreds2$source=="NA","Pseudoabsence",apreds2$source)
apreds2$cat=ifelse(apreds2$source=="both","Virion & WGS",apreds2$cat)
apreds2$cat=ifelse(apreds2$source=="virion","Virion", apreds2$cat)
apreds2$cat=ifelse(apreds2$source=="wgs","WGS", apreds2$cat)

apreds$cat=ifelse(apreds$source=="NA","Pseudoabsence",apreds$source)
apreds$cat=ifelse(apreds$source=="both","Virion & WGS",apreds$cat)
apreds$cat=ifelse(apreds$source=="virion","Virion", apreds$cat)
apreds$cat=ifelse(apreds$source=="wgs","WGS", apreds$cat)

#(18) Fix source type
```

```
apreds2$cat=factor(apreds2$cat,c("Pseudoabsence","Virion","Virion & WGS", "WGS"))
apreds$cat=factor(apreds$cat,levels=levels(apreds2$cat))

#(19) Fix type2
apreds$type2=revalue(apreds$type2,
                     c("FullModel"="Full Model",
                       "PartialModel"="Partial Model"))

#(20) Reformat for saving
preds=apreds2
preds$source=preds$cat
library(stringr)
preds$fam=str_to_title(preds$fam)
preds$ord=str_to_title(preds$ord)

#####KATIE, YOU STOPPED HERE. YOU NEED TO REFORMAT CATPOX AS COWPOX.

#(21) Subset predictions for each model and save
preds1=preds[c("source","virus","treename","fam","ord","pred_Model1","link1")]
preds2=preds[c("source","virus","treename","fam","ord","pred_Model2","link2")]
# write.csv(preds1,"Output/PoxHost_predictions_Model1.csv")
# write.csv(preds2,"Output/PoxHost_predictions_Model2.csv")

#(22) Print predicted links organized by virus and source for discussion
preds1=preds1[order(preds1$virus,preds1$source,-preds1$pred_Model1),]
paste0("Predicted links for Model 1 by virus and source")
preds1[preds1$pred_Model1>0.5,]
length(which(preds1$pred_Model1>0.5))
length(which(preds1$pred_Model1>thv_mean1))
length(which(preds1$pred_Model1>thv3))

preds2=preds2[order(preds2$virus,preds2$source,-preds2$pred_Model2),]
paste0("Predicted links for Model 2 by virus and source")
preds2[preds2$pred_Model2>0.5,]
length(which(preds2$pred_Model2>0.5))
length(which(preds2$pred_Model2>thv_mean2))
```

Model predictions: Figures of predicted probabilities distribution - Figure 3

```
#(1) Test correlation between the predicted probabilities of the two models
cor(apreds2$pred_Model1,apreds2$pred_Model2,method='spearman') #computes Spearment correlation coeffici
cor.test(apreds2$pred_Model1,apreds2$pred_Model2,method='spearman') #tests for correlation b/w paires s

#(1) Figure 3a - Density plot of predicted probabilities
remotes::install_github("awhstin/awtools")
library(awtools)
cc=mpalette[2:5]
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 link")))+
  # xlim(0,1)+
  xlim(thv_mean1,1)+
  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) Host-OPV positivity"),
         fill=guide_legend(title="(a) Host-OPV positivity"))

#(2) Figure 3b - Scatterplot of predicted probabilities
f3B=ggplot(apreds2,aes(pred_Model1,pred_Model2))+
  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 Full Model 1')),
       y=expression(paste(italic(P),' from Partial Model 2')))+
  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) Host-OPV positivity"),
         fill=guide_legend(title="(a) Host-OPV positivity"))

#(3) Figure 3 - Combine figure 3a and 3b
png("Output/pp_figure_f1.png",width=6.5,height=4,units="in",res=300)
f3=ggarrange(f3A,f3B,common.legend=T)
f3
dev.off()

rm(f3,f3A,f3B,fs2A,fs2B,cc)
```

Threshold the results and export predictions of known and undiscovered hosts

```
#load libraries
library(openxlsx)
library(PresenceAbsence) #for thresholding results

#() Set seed
set.seed(12345)

##############################################################################
### Extract Binary Predictions for Model 1 ###
##############################################################################

### CALCULATE THRESHOLD VALUES ###

#() Load file
pred1 <- read.csv("Output/PoxHost_predictions_Model1.csv")
pred1$virus_treename=paste0(pred1$virus, "%",pred1$treename)

#() drop out 'feline poxvirus ita2_bc'
pred1 <- pred1[!(pred1$virus=="feline poxvirus ita2_bc"),]

#() Create table of optimal threshold values for 5 different methods
ts1 <- optimal.thresholds(data.frame(pred1[,c('virus_treename','link1','pred_Model1')]),
                          threshold = 10001,
                          opt.methods = c(2,3,4,5,10),
                          req.sens = 0.80,
                          na.rm = TRUE)
  #opt.methods=2: Sens=Spec = 0.0385000
  #opt.methods=3: MaxSens+Spec = 0.0597000
  #opt.methods=4: MaxKappa = 0.4949000 (Th that gives max value of Kappa - assess improvement over chan
  #opt.methods=5: MaxPCC = 0.5670222 (Th that maximizes total accuracy or Percent Correctly Classified)
  #opt.methods=10: ReqSens = 0.0365000 (set required sensitivity - req.sens)

#() Add method applying 85% required sensitivity
ts1[nrow(ts1) + 1,] <- optimal.thresholds(data.frame(pred1[,c('virus_treename','link1','pred_Model1')])
                          threshold = 10001,
                          opt.methods = c(10),
                          req.sens = 0.85,
                          na.rm = TRUE)

#() Add method applying 90% required sensitivity
ts1[nrow(ts1) + 1,] <- optimal.thresholds(data.frame(pred1[,c('virus_treename','link1','pred_Model1')])
                          threshold = 10001,
                          opt.methods = c(10),
                          req.sens = 0.9,
                          na.rm = TRUE)

#() Add method applying 95% required sensitivity
ts1[nrow(ts1) + 1,] <- optimal.thresholds(data.frame(pred1[,c('virus_treename','link1','pred_Model1')])
                          threshold = 10001,
                          opt.methods = c(10),
                          req.sens = 0.95,
```

```
                          na.rm = TRUE)

# Create function that sums the number of predicted obs (known and unknown) and just unknown obs predic
cut.1 <- function(x) {sum(pred1$pred_Model1 > x)}
cut.2 <- function(x) {sum(pred1$pred_Model1[pred1$link1==0] > x)}

# Apply cut function using THVs from ts.1 and from prior calculated THVs for Model 1 to view number of
sapply(unlist(ts1[2]), cut.1)
sapply(unlist(ts1[2]), cut.2)
sum(pred1$link1)
# cut.1(thv1)
# cut.1(thv_mean1)

# save threshold values
write.csv(ts1, file='Output/ts_model1.csv')

### APPLY THRESHOLDS TO PREDICTIONS ###

# Extract selected optimum threshold values from ts.1
ts1_rs0.8 <- as.data.frame(ts1[5,])
ts1_rs0.85 <- as.data.frame(ts1[6,])
ts1_rs0.9 <- as.data.frame(ts1[7,])
ts1_rs0.95 <- as.data.frame(ts1[8,])
ts1_mss3 <- as.data.frame(ts1[2,])

# Extract lists of known and unknown predicted hosts applying different thresholds
  pred1$virus_treename=paste0(pred1$virus, "%",pred1$treename)

#(2) Threshold the results to binary outputs
  pred1 %>%
    mutate(bin_rs0.8 = ifelse(pred_Model1 > ts1_rs0.8$pred_Model1, 1, 0),
           bin_rs0.85 = ifelse(pred_Model1 > ts1_rs0.85$pred_Model1, 1, 0),
           bin_rs0.9 = ifelse(pred_Model1 > ts1_rs0.9$pred_Model1, 1, 0),
           bin_mss3 = ifelse(pred_Model1 > ts1_mss3$pred_Model1,1,0)) -> pred1

#(4) Pull out the relevant lists
pred1 %>% filter(link1==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> known
pred1 %>% filter(bin_rs0.8==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_rs0.8
pred1 %>% filter(bin_rs0.85==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_rs0.85
pred1 %>% filter(bin_rs0.9==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_rs0.9
pred1 %>% filter(bin_mss3==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_mss3

#(5) Sort and create table of known and unknown hosts (hosts that do not overlap)
pred_kn <- as.data.frame(sort(known))
pred_unk_rs0.8 <- as.data.frame(sort(pred_rs0.8[!(pred_rs0.8 %in% known)])) #n=421
pred_unk_rs0.85 <- as.data.frame(sort(pred_rs0.8[!(pred_rs0.85 %in% known)])) #n=421
pred_unk_rs0.9 <- as.data.frame(sort(pred_rs0.9[!(pred_rs0.9 %in% known)])) #n=2174
pred_unk_mss3 <- as.data.frame(sort(pred_mss3[!(pred_mss3 %in% known)])) #n=1309

  #(3) How many predicted hosts are undiscovered by Model 1?
  nrow(pred_unk_rs0.8) #n=389; n=31 for mpox
  nrow(pred_unk_rs0.85) #n=389; n=31 for mpox
  nrow(pred_unk_rs0.9) #n=1571; n=31 for mpox
```

```
  nrow(pred_unk_mss3) #n=1550; n=31 for mpox

#(6) Rename variables
colnames(pred_kn) <- c("V1")
colnames(pred_unk_rs0.8) <- c("V1")
colnames(pred_unk_rs0.85) <- c("V1")
colnames(pred_unk_rs0.9) <- c("V1")
colnames(pred_unk_mss3) <- c("V1")

#(7) Create list of dataframes/tables
list_df <- list(pred_kn, pred_unk_rs0.8, pred_unk_rs0.85, pred_unk_rs0.9, pred_unk_mss3)

#(8) Extract virus as a separate column
list_df <- lapply(list_df, function(x) {x$virus <- sapply(strsplit(x$V1,'%'),function(x) paste(x[1],sep=

#(9) Extract genus as a separate column
list_df <- lapply(list_df, function(x) {x$genus <- sapply(strsplit(x$V1,'%'),function(x) paste(x[2],sep=

#(10) Clean up
list_df <- lapply(list_df, function(x) {x$V1=NULL; x})

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

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

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

#(14) Reformat to proper
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_kn <- as.data.frame(list_df[[1]])
pred_unk_rs0.8 <- as.data.frame(list_df[[2]])
pred_unk_rs0.85 <- as.data.frame(list_df[[3]])
pred_unk_rs0.9 <- as.data.frame(list_df[[4]])
pred_unk_mss3 <- as.data.frame(list_df[[5]])

#(16) Save for known hosts and unknown hosts where req.sens=0.8, req.sens=0.85, req.sens=0.9, and opt.m
pred_model1 <- list('known'=pred_kn,
                    'unknown_rs0.8'=pred_unk_rs0.8,
                    'unknown_rs0.85'=pred_unk_rs0.85,
                    'unknown_rs0.9'=pred_unk_rs0.9,
                    'unknown_mss3'=pred_unk_mss3)
write.xlsx(pred_model1, file='Output/Predicted_Links_Model1.xlsx')

#(17) Let's analyze the ratio of unknown predicted hosts to known hosts
list_virus <- list_df
```

```
list_virus <- lapply(list_virus, function(x) {x$link <- 1; x})
list_virus <- lapply(list_virus, function(x) {x <- aggregate(link ~ virus, data=x, FUN=sum); x})
known <- list_virus[[1]]$link
list_virus <- lapply(list_virus, function(x) {x <- cbind(x, known); x})
list_virus <- lapply(list_virus, function(x) {x$sum <- x$link+x$known; x})
list_virus <- lapply(list_virus, function(x) {x$ratio_sum_to_known <- x$sum/x$known; x})
list_virus <- lapply(list_virus, function(x) {x$opv_ratio <- sum(x$sum)/sum(x$known); x})

virus_ratio_rs0.8 <- as.data.frame(list_virus[[2]])
virus_ratio_rs0.85 <- as.data.frame(list_virus[[3]])
virus_ratio_rs0.9 <- as.data.frame(list_virus[[4]])
virus_ratio_mss0.3 <- as.data.frame(list_virus[[5]])

#() save as excel sheets
library(openxlsx)
virus_ratio_model1 <- list('rs0.8'=virus_ratio_rs0.8,
                           'rs0.85'=virus_ratio_rs0.85,
                           'rs0.9'=virus_ratio_rs0.9,
                           'mss0.3' = virus_ratio_mss0.3)
write.xlsx(virus_ratio_model1, file = 'Output/virus_ratio_model1.xlsx')

################################################################################
### Extract Binary Predictions for Model 2 ###
################################################################################

### CALCULATE THRESHOLD VALUES ###

#() Load file
pred2 <- read.csv("Output/PoxHost_predictions_Model2.csv")
pred2$virus_treename=paste0(pred2$virus, "%",pred2$treename)

#() remove NA vaccinia values
pred2 <- pred2[which(!is.na(pred2$link2)),]

#() drop out 'feline poxvirus ita2_bc'
pred2 <- pred2[!(pred2$virus=="feline poxvirus ita2_bc"),]

ts2 <- optimal.thresholds(data.frame(pred2[,c('virus_treename','link2','pred_Model2')]),
                          threshold = 10001,
                          opt.methods = c(2,3,4,5,10),
                          req.sens = 0.90,
                          na.rm = TRUE)

#() Add method applying 85% required sensitivity
ts2[nrow(ts2) + 1,] <- optimal.thresholds(data.frame(pred2[,c('virus_treename','link2','pred_Model2')]),
                          threshold = 10001,
                          opt.methods = c(10),
                          req.sens = 0.85,
                          na.rm = TRUE)

#() Add method applying 90% required sensitivity
ts2[nrow(ts2) + 1,] <- optimal.thresholds(data.frame(pred2[,c('virus_treename','link2','pred_Model2')]),
                          threshold = 10001,
```

```
                            opt.methods = c(10),
                            req.sens = 0.9,
                            na.rm = TRUE)

#() Add method applying 95% required sensitivity
ts2[nrow(ts2) + 1,] <- optimal.thresholds(data.frame(pred2[,c('virus_treename','link2','pred_Model2')])
                            threshold = 10001,
                            opt.methods = c(10),
                            req.sens = 0.95,
                            na.rm = TRUE)

# Create function that sums the number of unknown obs predicted as links based on calculated THVs for M
cut.2 <- function(x) {sum(pred2$pred_Model2[pred2$link2==0] > x)}

# Apply cut function using THVs from ts.2 and from prior calculated THVs for Model 2 to view number of
sapply(unlist(ts2[2]), cut.2)
# cut.2(thv2)
# cut.2(thv_mean2)

# save threshold values
write.csv(ts2, file='Output/ts_model2.csv')

### APPLY THRESHOLDS TO PREDICTIONS ###

# Extract selected optimum threshold values from ts.2
ts2_rs0.8 <- as.data.frame(ts2[5,])
ts2_rs0.85 <- as.data.frame(ts2[6,])
ts2_rs0.9 <- as.data.frame(ts2[7,])
ts2_rs0.95 <- as.data.frame(ts2)
ts2_ses2 <- as.data.frame(ts2[1,])
ts2_mss3 <- as.data.frame(ts2[2,])

# Extract lists of known and unknown predicted hosts applying different thresholds
pred2$virus_treename=paste0(pred2$virus, "%",pred2$treename)

  #(2) Threshold the results to binary outputs
  pred2 %>%
    mutate(bin_rs0.8 = ifelse(pred_Model2 > ts2_rs0.8$pred_Model2, 1, 0),
           bin_rs0.85 = ifelse(pred_Model2 > ts2_rs0.85$pred_Model2, 1, 0),
           bin_rs0.9 = ifelse(pred_Model2 > ts2_rs0.9$pred_Model2, 1, 0),
           bin_mss3 = ifelse(pred_Model2 > ts2_mss3$pred_Model2,1,0)) -> pred2

#(4) Pull out the relevant lists
pred2 %>% filter(link2==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> known
pred2 %>% filter(bin_rs0.8==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_rs0.8
pred2 %>% filter(bin_rs0.85==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_rs0.85
pred2 %>% filter(bin_rs0.9==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_rs0.9
pred2 %>% filter(bin_mss3==1) %>% dplyr::pull(virus_treename) %>% gsub("_"," ",.) -> pred_mss3

#(5) Sort and create table of known and unknown hosts (hosts that do not overlap)
pred_kn <- as.data.frame(sort(known))
pred_unk_rs0.8 <- as.data.frame(sort(pred_rs0.8[!(pred_rs0.8 %in% known)])) #n=421
pred_unk_rs0.85 <- as.data.frame(sort(pred_rs0.85[!(pred_rs0.8 %in% known)])) #n=421
```

```r
pred_unk_rs0.9 <- as.data.frame(sort(pred_rs0.9[!(pred_rs0.9 %in% known)])) #n=2174
pred_unk_mss3 <- as.data.frame(sort(pred_mss3[!(pred_mss3 %in% known)])) #n=1309

  #(3) How many predicted hosts are undiscovered by Model 1?
  nrow(pred_unk_rs0.8) #n=399; n=31 for mpox
  nrow(pred_unk_rs0.85) #n=897; n=31 for mpox
  nrow(pred_unk_rs0.9) #n=1582; n=31 for mpox
  nrow(pred_unk_mss3) #n=1316; n=31 for mpox

#(6) Rename variables
colnames(pred_kn) <- c("V1")
colnames(pred_unk_rs0.8) <- c("V1")
colnames(pred_unk_rs0.85) <- c("V1")
colnames(pred_unk_rs0.9) <- c("V1")
colnames(pred_unk_mss3) <- c("V1")

#(7) Create list of dataframes/tables
list_df <- list(pred_kn, pred_unk_rs0.8, pred_unk_rs0.85, pred_unk_rs0.9, pred_unk_mss3)

#(8) Extract virus as a separate column
list_df <- lapply(list_df, function(x) {x$virus <- sapply(strsplit(x$V1,'%'),function(x) paste(x[1],sep=

#(9) Extract genus as a separate column
list_df <- lapply(list_df, function(x) {x$genus <- sapply(strsplit(x$V1,'%'),function(x) paste(x[2],sep=

#(10) Clean up
list_df <- lapply(list_df, function(x) {x$V1=NULL; x})

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

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

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

#(14) Reformat to proper
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_kn <- as.data.frame(list_df[[1]])
pred_unk_rs0.8 <- as.data.frame(list_df[[2]])
pred_unk_rs0.85 <- as.data.frame(list_df[[3]])
pred_unk_rs0.9 <- as.data.frame(list_df[[4]])
pred_unk_mss3 <- as.data.frame(list_df[[5]])

#(16) Save for known hosts and unknown hosts where req.sens=0.8, req.sens=0.85, req.sens=0.9, and opt.m
pred_model2 <- list('known'=pred_kn,
                    'unknown_rs0.8'=pred_unk_rs0.8,
```

```
                    'unknown_rs0.85'=pred_unk_rs0.85,
                    'unknown_rs0.9'=pred_unk_rs0.9,
                    'unknown_mss3'=pred_unk_mss3)
write.xlsx(pred_model2, file='Output/Predicted_Links_Model2.xlsx')

#(17) Let's analyze the ratio of unknown predicted hosts to known hosts
list_virus <- list_df
list_virus <- lapply(list_virus, function(x) {x$link <- 1; x})
list_virus <- lapply(list_virus, function(x) {x <- aggregate(link ~ virus, data=x, FUN=sum); x})
known <- list_virus[[1]]$link
list_virus <- lapply(list_virus, function(x) {x <- cbind(x, known); x})
list_virus <- lapply(list_virus, function(x) {x$sum <- x$link+x$known; x})
list_virus <- lapply(list_virus, function(x) {x$ratio_sum_to_known <- x$sum/x$known; x})
list_virus <- lapply(list_virus, function(x) {x$opv_ratio <- sum(x$sum)/sum(x$known); x})

virus_ratio_rs0.8 <- as.data.frame(list_virus[[2]])
virus_ratio_rs0.85 <- as.data.frame(list_virus[[3]])
virus_ratio_rs0.9 <- as.data.frame(list_virus[[4]])
virus_ratio_mss0.3 <- as.data.frame(list_virus[[5]])

#() save as excel sheets
virus_ratio_model2 <- list('rs0.8' = virus_ratio_rs0.8,
                           'rs0.85' = virus_ratio_rs0.85,
                           'rs0.9' = virus_ratio_rs0.9,
                           'mss0.3' = virus_ratio_mss0.3)
write.xlsx(virus_ratio_model2, file = 'Output/virus_ratio_model2.xlsx')

####KATIE, YOU STOPPED HERE. We also need to go to the phylopart code and rerun the code dropping out f
```

Model predictions: Explore model correlation and phylogenetic signal

```
# load("/Users/katietseng/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernan

#Load library
library(ape)

#Load phylogeny
load("/Users/katietseng/Library/CloudStorage/OneDrive-WashingtonStateUniversity(email.wsu.edu)/Fernande
mtree=hostTree

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

#Trim
bdata=subset(apreds2,tree=='keep')

#Save new vars
bdata$label=bdata$treename
bdata$Species=bdata$treename
```

```
#Drop 'feline poxvirus ita2_bc': this species has a single known link to the genus 'homo' and can be re
bdata <- bdata[!(bdata$virus=="feline poxvirus ita2_bc"),]

#Generate variable of whether a link was predicted based on threshold value of __
bdata$link1_rs0.8 <- ifelse(bdata$pred_Model1 > ts1_rs0.8$pred_Model1, 1, 0)
bdata$link1_rs0.85 <- ifelse(bdata$pred_Model1 > ts1_rs0.85$pred_Model1, 1, 0)
bdata$link1_rs0.9 <- ifelse(bdata$pred_Model1 > ts1_rs0.9$pred_Model1, 1, 0)
bdata$link1_mss3 <- ifelse(bdata$pred_Model1 > ts1_mss3$pred_Model1, 1, 0)

#Aggregate at the host genera level assuming the mean b/c pgls() takes only one value per tip
bdata_mean <- aggregate(cbind(pred_Model1,pred_Model2) ~ treename + ord + fam + genus + label + Species

# For visualizing binary classification of links on a phylogenetic tree (where only one tip value is al
  # 'cetacean poxvirus 1'
  # 'cetacean poxvirus 2'
  # 'orthopoxvirus gcp2010'
  # 'orthopoxvirus gcp2013'
  # 'orthopoxvirus sp.'
  # 'orthopoxvirus tena dona'
  # 'steller sea lion poxvirus'

#Combine all rows of data for unclassified virus species into a single dataset
bdata_unclassified <- bdata[bdata$virus=="cetacean poxvirus 1",]
bdata_unclassified <- rbind(bdata_unclassified, bdata[bdata$virus=="cetacean poxvirus 2",])
bdata_unclassified <- rbind(bdata_unclassified, bdata[bdata$virus=="orthopoxvirus gcp2010",])
bdata_unclassified <- rbind(bdata_unclassified, bdata[bdata$virus=="orthopoxvirus gcp2013",])
bdata_unclassified <- rbind(bdata_unclassified, bdata[bdata$virus=="orthopoxvirus sp.",])
bdata_unclassified <- rbind(bdata_unclassified, bdata[bdata$virus=="orthopoxvirus tena dona",])
bdata_unclassified <- rbind(bdata_unclassified, bdata[bdata$virus=="steller sea lion poxvirus",])

#Aggregate unclassified virus species at host genera level and take max value of link
bdata_unclassified <- aggregate(cbind(link1,link2,link1_rs0.8,link1_rs0.85,link1_rs0.9,link1_mss3) ~ tr

#Add virus column for "unclassified orthopoxvirus"
bdata_unclassified$virus <- "unclassified orthopoxvirus"

#Create new df bdata_sum from bdata to which we will append bdata_unclassified: match column names to t
bdata_sum <- subset(bdata, select=c(colnames(bdata_unclassified)))
bdata_sum <- bdata_sum[!(bdata_sum$virus=="cetacean poxvirus 1"),]
bdata_sum <- bdata_sum[!(bdata_sum$virus=="cetacean poxvirus 2"),]
bdata_sum <- bdata_sum[!(bdata_sum$virus=="orthopoxvirus gcp2010"),]
bdata_sum <- bdata_sum[!(bdata_sum$virus=="orthopoxvirus gcp2013"),]
bdata_sum <- bdata_sum[!(bdata_sum$virus=="orthopoxvirus sp."),]
bdata_sum <- bdata_sum[!(bdata_sum$virus=="orthopoxvirus tena dona"),]
bdata_sum <- bdata_sum[!(bdata_sum$virus=="steller sea lion poxvirus"),]

#Append to bdata_sum new virus group of unclassified
bdata_sum <- rbind(bdata_sum, bdata_unclassified)

#() Aggregate at the host genera level assuming the sum
bdata_sum <- aggregate(cbind(link1,link2,link1_rs0.8,link1_rs0.85,link1_rs0.9,link1_mss3) ~ treename + 

#() Combine bdata_mean and bdata_sum
```

```
bdata_agg <- merge(bdata_mean, bdata_sum, by=c("treename", "ord", "fam", "genus", "label", "Species"))

### We take the mean of the predictions for visualizing the phylotree later b/c if the mean is >1 that
### link1 and link2 represent known hosts; unknown_link1_ and unknown_link2_ represent unknown predicte

#(6) Merge tree data with prediction data
cdata=comparative.data(phy=mtree,data=bdata_agg,names.col=treename,vcv=T,na.omit=F,warn.dropped=T)
  #cdata=comparative.data(phy=mtree,data=bdata,names.col=treename,vcv=T,na.omit=F,warn.dropped=T)
  #combines phylogenies w/ datasets ensuring consistent structure (needed for the next step -> caper::p
  #vcv=T indicates to include variance covariance array representing phylogeny w/in the comparative dat

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

#(8) Measure phylogenetic signal (Pagel's lambda) of model predictions using caper::pgls
Model1_lmod=pgls(pred_Model1~1,data=cdata,lambda="ML")   #pgls fits a linear model while taking into ac
Model2_lmod=pgls(pred_Model2~1,data=cdata,lambda="ML")   #lambda = value for lambda transformation; 'ML
###for more info: https://static1.squarespace.com/static/5459da8ae4b042d9849b7a7b/t/57ea64eae58c62718aa

#(9) Get Pagel's lambda (phylogenetic signal)
summary(Model1_lmod)
summary(Model2_lmod)
```

Model predictions: Identify taxonomic patterns

```
#load library
library(phylofactor)

#(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) Phylofactorization of Model 1 predictions
set.seed(1)
```

```
Model1pred_pf=gpf(Data=cdata$data,tree=cdata$phy,
                frmla.phylo=pred_Model1~phylo,
                family=gaussian,algorithm='phylo',nfactors=20,min.group.size=5)

#(8) Phylofactorization of Model 2 predictions
set.seed(1)
Model2pred_pf=gpf(Data=cdata$data,tree=cdata$phy,
                 frmla.phylo=pred_Model2~phylo,
                 family=gaussian,algorithm='phylo',nfactors=10,min.group.size=5)

#(9) Summarize
Model1pred_pf_results=pfsum(Model1pred_pf)$results
Model2pred_pf_results=pfsum(Model2pred_pf)$results

#(10) Add model
Model1pred_pf_results$model="Model 1 (Full)"
Model2pred_pf_results$model="Model 2 (Partial)"

#(11) Bind
predpfs=rbind.data.frame(Model1pred_pf_results,Model2pred_pf_results)

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

#(13) Write
write.csv(predpfs,"Output/pp_taxo_t1.csv")
```

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

```
#load library
library(treeio)
library(ggtree)
library(plotrix)

#(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

#() Let's look at the distribution of predicted probabilities and (1) cap those that are outliers at 0.
```

```
hist(cdata$data$pred_Model1)
cdata$data$cap0.15_pred_Model1 = ifelse(cdata$data$pred_Model1 > 0.15, 0.15, cdata$data$pred_Model1)
cdata$data$cap0.1_pred_Model1 = ifelse(cdata$data$pred_Model1 > 0.1, 0.1, cdata$data$pred_Model1)
cdata$data$quant0.95_pred_Model1 = ifelse(cdata$data$pred_Model1 > quantile(cdata$data$pred_Model1,0.95)
# cdata$data$circle0.1 = 0.1

#(6) Make data frame for Model 1
samp=data.frame(x=tdata$x,
                y=tdata$y,
                yend=tdata$y,
                # xend_Model1=rescale(cdata$data$pred_Model1,c(max(tdata$x),xmax)),
                xend_Model1=rescale(cdata$data$cap0.15_pred_Model1,c(max(tdata$x),xmax)), #rescale(x, n
                # xend_Model1=rescale(cdata$data$quant0.95_pred_Model1,c(max(tdata$x),xmax)),
                # xend_Model1_circle=rescale(cdata$data$circle0.1,c(max(tdata$x),xmax)),
                link1_rs0.8=cdata$data$link1_rs0.8,
                link1_rs0.85=cdata$data$link1_rs0.85,
                link1_rs0.9=cdata$data$link1_rs0.9,
                link1_mss3=cdata$data$link1_mss3,
                factor_link1_rs0.8=as.factor((cdata$data$link1_rs0.8)),
                factor_link1_rs0.85=as.factor((cdata$data$link1_rs0.85)),
                factor_link1_rs0.9=as.factor((cdata$data$link1_rs0.9)),
                factor_link1_mss3=as.factor((cdata$data$link1_mss3)),
                treename=tdata$label)

#() For plotting, separate df into predicted host genera and non-predicted host genera for different th
samp_nopred_rs0.8 <- samp[samp$link1_rs0.8==0,]
samp_pred_rs0.8 <- samp[samp$link1_rs0.8>0,]
samp_nopred_rs0.85 <- samp[samp$link1_rs0.85==0,]
samp_pred_rs0.85 <- samp[samp$link1_rs0.85>0,]
samp_nopred_rs0.9 <- samp[samp$link1_rs0.9==0,]
samp_pred_rs0.9 <- samp[samp$link1_rs0.9>0,]
samp_nopred_mss3 <- samp[samp$link1_mss3==0,]
samp_pred_mss3 <- samp[samp$link1_mss3>0,]

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

#(8) Plot base tree and highlight significant clades for Model 1
gg=pbase
for(i in 1:nrow(Model1pred_pf_results)){
  gg=gg+
    geom_hilight(node=Model1pred_pf_results$node[i],
                 alpha=ifelse(Model1pred_pf_results$tips[i]/Ntip(cdata$phy)<0.5,0.5,0.25), #alpha=0.5 i
                 fill="black")
}

# #(9) Add pred-probs and assign color based on whether host genera had any predicted host-virus links
p1_rs0.8=gg+
  geom_segment(
    data=samp_nopred_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_Model1,yend=yend,
                alpha=factor_link1_rs0.8),
    color="gray",linewidth=0.75)+
```

```
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(
    data=samp_pred_rs0.8,
    mapping=aes(x=x,y=y,xend=xend_Model1,yend=yend,
                colour=factor(link1_rs0.8)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      limits=seq(from=1,to=12),
                      name="Number of predicted links\nwith OPV species") +
 theme(legend.position=c(1.1,.5))

p1_rs0.85=gg+
  geom_segment(
    data=samp_nopred_rs0.85,
    mapping=aes(x=x,y=y,xend=xend_Model1,yend=yend,
                alpha=factor_link1_rs0.85),
    color="gray",linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(data=samp_pred_rs0.85,aes(x=x,y=y,xend=xend_Model1,yend=yend,
                                  colour=factor(link1_rs0.85)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      limits=seq(from=1,to=12),
                      name="Number of predicted links\nwith OPV species") +
 theme(legend.position=c(1.1,.5))

p1_rs0.9=gg+
  geom_segment(
    data=samp_nopred_rs0.9,
    mapping=aes(x=x,y=y,xend=xend_Model1,yend=yend,
                alpha=factor_link1_rs0.9),
    color="gray",linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(data=samp_pred_rs0.9,aes(x=x,y=y,xend=xend_Model1,yend=yend,
                                  colour=factor(link1_rs0.9)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
                      limits=seq(from=1,to=12),
                      name="Number of predicted links\nwith OPV species") +
 theme(legend.position=c(1.1,.5))

p1_mss3=gg+
  geom_segment(
    data=samp_nopred_mss3,
    mapping=aes(x=x,y=y,xend=xend_Model1,yend=yend,
                alpha=factor_link1_mss3),
    color="gray",
    linewidth=0.75)+
  scale_alpha_discrete(range=c(1,1),name="No predicted links")+
  geom_segment(data=samp_pred_mss3,aes(x=x,y=y,xend=xend_Model1,yend=yend,
                                  colour=factor(link1_mss3)),linewidth=0.75,)+
  scale_color_viridis(discrete=TRUE,option="D", direction=-1,
                      guide=guide_legend(reverse=TRUE),
```

```
                      limits=seq(from=1,to=12),
                      name="Number of predicted links\nwith OPV species") +
 theme(legend.position=c(1.1,.5))

#(12) Figure F3C - Combine figures for different threshold values
f3C_model1=ggarrange(p1_rs0.8,p1_rs0.85, p1_rs0.9,p1_mss3,
               labels=c("(a) ReqSens0.8; Th=0.14","(b) ReqSens0.85; Th=0.08","(c) ReqSens0.9; Th=0.05","
               label.x=c(-0.25,-0.25,-0.25,-0.25),
               label.y=0.2,
               font.label=list(face="plain",size=13),
               ncol=2,nrow=2,
               common.legend = TRUE, legend="right")

#(13) Visualize

#(14) Let's use ReqSens0.8 and MaxSensSpec to demonstrate the impact of threshold moving on binary clas
f3C_model1=ggarrange(p1_rs0.8,p1_mss3,
               labels=c("(C) ReqSens0.8; Th=0.14","(D) MaxSensSpec; Th=0.06"),
               label.x=c(0.27,0.27),
               label.y=c(1,1),
               font.label=list(face="plain",size=13),
               ncol=1,nrow=2,
               common.legend = TRUE, legend="left")

#(13) Revise Figure 3
png("Output/pp_tree_f1.png",width=12,height=8,units="in",res=300)
f3C_model1
dev.off()

#() Export tip labels for mammal orders
tiplabels <- ggtree(dtree,layout="fan",branch.length="none",size=0.25) + geom_tiplab(aes(label = dtree@
png("Output/tiplabels1",width=24,height=24,units="in", res=300)
tiplabels
dev.off()

#() Export tip labels again for mammal orders but dropping duplicate labels
temp_tree <- dtree
temp_tree@extraInfo$ord[duplicated(temp_tree@extraInfo$ord)] <- NA

tiplabels <- ggtree(dtree,layout="fan",branch.length="none",size=0.25) + geom_tiplab(aes(label = temp_t

png("Output/tiplabels2",width=24,height=24,units="in", res=300)
tiplabels
dev.off()

######## BEGIN INTERMISSION, TRYING TO ADD SCALE/CONCENTRIC CIRCLES ########

#() Install package ggtreeExtra
#https://github.com/YuLab-SMU/plotting-tree-with-data-using-ggtreeExtra
BiocManager::install("ggtreeExtra")
library(ggrteeExtra)

#() Add concentric circles
```

```
#https://yulab-smu.top/treedata-book/chapter10.html
BiocManager::install("phyloseq")
data("GlobalPatterns")

plot <- p1_rs0.8 +
  geom_segment(data=samp,aes(x=x,y=y,xend=xend_Model1_circle,yend=yend,
                                colour=factor(link1_rs0.8)),linewidth=0.75,)
gg +
     geom_fruit(
         data=samp,
         geom=geom_segment,
         mapping = aes(
                     y=0.5,
                     x=x,
                     xend=xend_Model1,
                     yend=yend,
                     colour=factor(samp$link1_rs0.8)),
         # axis.params=list(
         #                  axis       = "x",
         #                  text.size  = 1.8,
         #                  hjust      = 1,
         #                  vjust      = 0.5,
         #                  nbreak     = 3,
         #             ),
         grid.params=list()
     )

### SAMPLE SILHOUETTE IMAGE CODE
# Silhouettes image in the outermost ring
install.packages("ggtreeExtra")
library(ggtreeExtra)
library(ggimage)
phylopicda <- read.csv("https://raw.githubusercontent.com/YuLab-SMU/plotting-tree-with-data-using-ggtre

p1_rs0.8+geom_nodelab(aes(image=phylopicda), geom="phylopic", alpha=.5, color='steelblue')

fig4 <- fig3 +
      new_scale_colour() +
      geom_fruit(
          data=phylopicda,
          geom=geom_phylopic,
          mapping=aes(y=taxa, image=uid, color=class),
          size=0.035,
          offset=0.16,
          alpha=0.8,
          position=position_identityx()
      ) +
      scale_colour_manual(
          values=c("#b2df8a","#33a02c","#fb9a99",
                   "#EACB47","#6a3d9a"),
          guide="none"
      ) +
      theme(
```

```
          legend.background=element_rect(fill=NA),
          legend.title=element_text(size=7),
          legend.text=element_text(size=5),
          legend.spacing.y = unit(0.02, "cm")
      )

### ANNOTATE TREE WITH PHYLOPIC: https://yulab-smu.top/treedata-book/chapter8.html

######## END INTERMISSION, TRYING TO ADD SCALE/CONCENTRIC CIRCLES ########
```

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")
library(velox)
library(fasterize)
library(sf)
library(openxlsx)
library(PresenceAbsence) #for thresholding results

#(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

#(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
r <- 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=' '))

#(7) Pull out the relevant lists of known hosts and predicted hosts for all OPVs
pred1 %>% filter(link1==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> known.1 #n=102
pred1 %>% filter(bin_rs0.8==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred.1_rs0.8 #n=503
pred1 %>% filter(bin_rs0.85==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred.1_rs0.85 #n=503
pred1 %>% filter(bin_rs0.9==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred.1_rs0.9 #n=2266
pred1 %>% filter(bin_mss3==1) %>% dplyr::pull(treename) %>% gsub("_"," ",.) -> pred.1_mss3 #n=1399

#() Pull out the relevant lists of predicted unknown hosts for all OPVs
pred.1_unk_rs0.8 <- pred.1_rs0.8[!(pred.1_rs0.8 %in% known.1)] #n=311
pred.1_unk_rs0.85 <- pred.1_rs0.85[!(pred.1_rs0.85 %in% known.1)] #n=311
pred.1_unk_rs0.9 <- pred.1_rs0.9[!(pred.1_rs0.9 %in% known.1)] #n=1844
pred.1_unk_mss3 <- pred.1_mss3[!(pred.1_mss3 %in% known.1)] #n=1097

###iucn$treename is the genus of the iucn species
iucn.1_kn <- iucn[iucn$treename %in% known.1,] #n=1699
iucn.2_rs0.8 <- iucn[iucn$treename %in% pred.1_unk_rs0.8,] #n=1586
iucn.3_rs0.85 <- iucn[iucn$treename %in% pred.1_unk_rs0.85,] #n=1586
iucn.3_rs0.9 <- iucn[iucn$treename %in% pred.1_unk_rs0.9,] #n=5155
iucn.4_mss3 <- iucn[iucn$treename %in% pred.1_unk_mss3,] #n=4838

library(fasterize)
map.kn1 <- (fasterize(iucn.1_kn, r, fun="sum"))
map.pr1_rs0.8 <- (fasterize(iucn.2_rs0.8, r, fun="sum"))
map.pr1_rs0.85 <- (fasterize(iucn.3_rs0.85, r, fun="sum"))
map.pr1_rs0.9 <- (fasterize(iucn.3_rs0.9, r, fun="sum"))
map.pr1_mss3 <- (fasterize(iucn.4_mss3, r, fun="sum"))

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

map.kn1 <- fix(map.kn1)
map.pr1_rs0.8 <- fix(map.pr1_rs0.8)
map.pr1_rs0.85 <- fix(map.pr1_rs0.85)
map.pr1_rs0.9 <- fix(map.pr1_rs0.9)
map.pr1_mss3 <- fix(map.pr1_mss3)

#Maps 1 includes req.sens 85%
raster::stack(map.kn1, map.pr1_rs0.8, map.pr1_rs0.85, map.pr1_mss3) %>% raster::trim() -> maps1 #altern

#Maps 2 includes req.sens 90%
raster::stack(map.kn1, map.pr1_rs0.8, map.pr1_rs0.9, map.pr1_mss3) %>% raster::trim() -> maps2 #alterna

names(maps1) <- c('KnownModel1', 'PredModel1_ReqSens0.8', 'PredModel1_ReqSens0.85', 'PredModel1_MaxSens
names(maps2) <- c('KnownModel1', 'PredModel1_Unk_ReqSens0.8', 'PredModel1_Unk_ReqSens0.9', 'PredModel1_

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

mycolors <- colorRampPalette(rev(brewer.pal(10,"Spectral")))(21)
```

```
mycolors[1] <- "#C0C0C0"

png("Output/Map_f1.png",width=10,height=10,units="in",res=300)
rasterVis::levelplot(maps1,
                     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()

png("Output/Map_f2_maxsensspec.png",width=10,height=10,units="in",res=300)
rasterVis::levelplot(maps2,
                     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()

###Selected Map using 90% sensitivity threshold

#Maps final using rs0.9
raster::stack(map.kn1, map.pr1_rs0.9) %>% raster::trim() -> maps_rs0.9
names(maps_rs0.9) <- c('KnownModel1', 'PredModel1_Unk_ReqSens0.9')

#Generate the actual visualization
png("Output/Map_rs0.9.png",width=10,height=10,units="in",res=300)
rasterVis::levelplot(maps_rs0.9,
                     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()
```