BINF6210: Assignment 2
Author: Lisa Hoeg

INTRODUCTION

For this assignment I have chosen a project based on option one, Supervised Machine Learning, with the intent to Classify by Taxonomy. The classifier built will be able to distinguish at the genus level between two genera of the Muridae family of the Rodentia order. Within Muridae, the genera of interest were selected to be Apodemus and Niviventer by browsing the BOLD taxonomy database for genera with a reasonable number of species sampled. The expectation was that if a genus has a significant number of represented species on BOLD, which has a user-friendly browsing interface, there would likely be a comparable amount of data available through NCBI as well. Although COI is emerging as a popular gene for identifying species, as demonstrated through the Barcode of Life Project (Ratnasingham & Herbert, 2007), I have chosen CytB for this project because of the abundance of data on NCBI, and because it has been verified as an effective subgenus level classifier for the related Muridae genus, Apomys (Heaney et al., 2011).

This script will use the rentrez package to access NCBI's nuccore database and acquire sequences of the mitochondrial gene cytochrome b (CytB) for Apodemus and Niviventer, which will be analysed with a variety of packages, including randomForest to generate the supervised machine learning model. After refining the model to a suitable level of accuracy (target accuracy greater than 99%), sequences that are misclassified will be explored to see what might be causing the discrepancy between the label and the prediction

CODE SECTION 1 – DATA ACQUISITION, EXPLORATION, FILTERING, AND QUALITY CONTROL

1. Introduction: Supervised Machine Learning to Classify by Taxonomy ----
#
For Assignment 2 of BINF6210 I have chosen project option 1: Supervised Machine Learning.
This script will use the rentrez package to access NCBI's nuccore database and acquire sequences of the mitochondrial gene cytochrome b (CytB) for Apodemus and Niviventer, two genera within the Muridae family of the Rodentia order. These genera were selected by browsing the BOLD taxonomy database for genera with a reasonable number of species sampled, as an indicator that sufficient data would also be available through NCBI.
After refining the model to a suitable level of accuracy, sequences that are misclassified will be explored to see what might be causing the discrepancy between the label and the prediction
Although COI is emerging as a popular gene for identifying species, as demonstrated through the Barcode of Life Project (Ratnasingham & Herbert, 2007), I have chosen CytB for this project because of the abundance of data on NCBI, and because it has been verified as an effective species level classifier for the related Muridae genus, Apomys (Heaney et al., 2011).

Note: On trying to select and run all commands for error troubleshooting, the fourth basic search (niviventer_search_cytb) on line 47 exceeded API rate limit. This is not an issue when running a single line at a time. As a precaution, I have added the api key to all server requests, although it may not be necessary.

Note 2: The line of code calling on the large fetch function is commented out for simplicity, and to run through the script as is, please have the provided file "Apodemus_Niviventer_CytB.fasta" in the working directory. Alternatively, uncomment lines 147 & 149 as directed in the script to download directly.

2. Install & Load Packages ----

# Optional: Set working directory to desired location for saving and accessing downloaded files
setwd("/Users/lisa/Documents/Bioinformatics/6210/A2/Final")

# Set apikey to send excessive requests to NCBI
apikey <- "3454c77fdcc045aa2a1dd70c3b303abb5409"

Install and load required packages
#install.packages("rentrez")

```
library(rentrez)
#BiocManager::install("Biostrings")
library(Biostrings)
#install.packages("tidyverse")
library(tidyverse)
#install.packages("caret")
library(caret)
#install.packages("randomForest")
library(randomForest)
#install.packages("gridExtra")
library(gridExtra)


#### 3. Get IDs and Explore Summaries of Interest ----

# What fields are available to narrow my search?
entrez_db_searchable("nuccore")

# Compare available data for COI and CytB genes to confirm preference for CytB
# How many records are available for Apodemus with COI sequence?
apodemus_search_coi <- entrez_search(db = "nuccore", term = "Apodemus[ORGN]
AND COI[GENE] NOT genome[TITL]", api_key = apikey)
apodemus_search_coi$count  #433
# How many records are available for Apodemus with CytB sequence?
apodemus_search_cytb <- entrez_search(db = "nuccore", term = "Apodemus[ORGN]
AND CytB[GENE] NOT genome[TITL]", api_key = apikey)
apodemus_search_cytb$count #3072
# How many records are available for Niviventer with COI sequence?
niviventer_search_coi <- entrez_search(db = "nuccore", term = "Niviventer[ORGN] AND
COI[GENE] NOT genome[TITL]", api_key = apikey)
niviventer_search_coi$count #934
# How many records are available for Niviventer with CytB sequence?
niviventer_search_cytb <- entrez_search(db = "nuccore", term = "Niviventer[ORGN]
AND CytB[GENE] NOT genome[TITL]", api_key = apikey)
niviventer_search_cytb$count #1428

# These results show that the CytB gene has more records available, and will generate
a data set of 4500 hits which is within the appropriate range recommended for
supervised machine learning project choice

# Get IDs for full set
# First find the total count
apo_niv_search <- entrez_search(db = "nuccore", term = "Apodemus[ORGN] OR
Niviventer[ORGN] AND CytB[GENE] NOT genome[TITL]", api_key = apikey)
```

```
apo_niv_search$count
length(apo_niv_search$ids)
# The total hits are 4500, as expected, but the number of IDs downloaded is only the
default retmax of 20
# Apply count of initial search to retmax in order to receive all relevant IDs
apo_niv_search2 <- entrez_search(db = "nuccore", term = "Apodemus[ORGN] OR
Niviventer[ORGN] AND CytB[GENE] NOT genome[TITL]", retmax =
apo_niv_search$count, use_history = T, api_key = apikey)
apo_niv_search2$count
length(apo_niv_search2$ids)
# All 4500 IDs have been received

# Get summary information from IDs of interest
# There is a limit to the request size for receiving summary data. Start with a sample of
200 for a preliminary exploration of the data.
# Set seed for reproducibility
set.seed(5555)
# Select sample of IDs for first exploration
apo_niv_200id <- sample(apo_niv_search2$ids, 200, replace = F)
# Get summary data for sample of 200
apo_niv_summ <- entrez_summary(db = "nuccore", id = apo_niv_200id, api_key =
apikey)
# What are the fields of information available?
apo_niv_summ

# Explore characteristics of sequence length data
summary(extract_from_esummary(apo_niv_summ, "slen"))
# The range is from 319bp to 1144bp with a median of 1123.5bp

# A histogram will help to understand the distribution of sequence length
# Need a dataframe to work with, so extract the slen data
slen_summ <- data.frame(slen = extract_from_esummary(apo_niv_summ, "slen"))
# Check
head(slen_summ)
# Plot & save
ggplot(slen_summ, aes(x = slen, xmin = 250, xmax = 1200)) +
  geom_histogram(binwidth = 25, color = "black", fill = "lightblue") +
  geom_vline(aes(xintercept = median(slen)),
             color="blue", linetype="dashed", size=1) +
  labs(title = "Sample CytB Sequence Lengths", x = "Sequence Length", y =
"Frequency") +
  annotate(geom = "text", x = c(620,950), y = c(23,43), label = c("Range = 319:1144",
"Median = 1123.5")) +
  ggsave("A2_fig1.pdf", width = 6, height = 6)
```

```
# Most of the sequences are above 700bp, with the greatest amount between 1100 and 1200. There are a few sequences that are much shorter than the rest at 300-500bp. It is worth considering discarding sequences shorter than 600 for the final analysis.

# If you don't wish to keep the file, you can use this line
#file.remove("A2_fig1.pdf")

# What variety of organism names are present in this sample?
unique(extract_from_esummary(apo_niv_summ, "organism"))
length(unique(extract_from_esummary(apo_niv_summ, "organism")))
# The organism field includes common names, not scientific names. There are 31 unique names in the summary, which suggests a similar number of scientific names in the title.
# Have a look at a few titles to verify the full scientific names that are present
sample(extract_from_esummary(apo_niv_summ, "title"), 10)
# This shows a variety of specimens named within the Apodemus and Niviventer genera as expected, and also verifies the selection of the cytb gene.
# Check genome of information
unique(extract_from_esummary(apo_niv_summ, "genome"))
# This shows that all genetic sequences are mitochondrial, which is true for the gene of interest CytB


#### 4. Fetch Fasta File for Complete Dataset ----

# Define the IDs for fetch
apo_niv_ids <- apo_niv_search2$ids

# Define function for overcoming request maximum and retrieving large data set

LargeFastaFetch <- function(ids_for_fetch, savefasta, numberOfSeqs = 200) {
  # ids_for_fetch is defined above from the search
  # savefasta is the name desired for the resulting saved file of fetched data
  # numberOfSeqs has a default of 200, but can be changed. It identifies how many sequences to fetch per request
  full_fasta <- "" # create empty object for receiving the fetched data
  for (start_rec in seq(0, length(apo_niv_ids), numberOfSeqs)) {
    small_fetch <- entrez_fetch(db = "nuccore", web_history = apo_niv_search2$web_history, rettype = "fasta", retstart = start_rec, retmax = numberOfSeqs, api_key = apikey)
    # each iteration of the loop will fetch the next 200 (or numberOfSeqs if defined) items from the id list provided
    full_fasta <- paste0(full_fasta, small_fetch)
    # the new fetch will paste to the end of the existing file
```

```
    print(paste0(round(start_rec/length(apo_niv_ids)*100), '%'))
  }

  message("All Files Downloaded\nSaving File")
  # Save file for future use
  write(full_fasta, savefasta, sep = "\n")
  # Read data back into R environment
  full_fasta <- Biostrings::readDNAStringSet(savefasta)
  # Set data as a dataframe for ease of use
  full_fasta.df <- data.frame(Title = names(full_fasta), Sequence = paste(full_fasta))
  message('Done.')
  return(full_fasta.df)
}

# To download for yourself, follow along with these lines
# Choose name for saved file of fetched data
#savefasta_as = "Apodemus_Niviventer_CytB.fasta"
# Call function to download all files
#apo_niv_fetch <- LargeFastaFetch(apo_niv_ids, savefasta_as)

# To read existing file from current directory
apo_niv_fetch <- Biostrings::readDNAStringSet("Apodemus_Niviventer_CytB.fasta")
# Set as dataframe
apo_niv_fetch <- data.frame(Title = names(apo_niv_fetch), Sequence =
paste(apo_niv_fetch))

# Verify the data returned is what I was expecting
class(apo_niv_fetch)
head(apo_niv_fetch)
dim(apo_niv_fetch)

# Looks good!

# If you don't want to keep the saved file, you can run the following lines
#file.remove(savefasta_as)

# Clean up environment before continuing data analysis.
rm(apo_niv_search, apo_niv_search2, apo_niv_summ, apodemus_search_coi,
apodemus_search_cytb, niviventer_search_coi, niviventer_search_cytb, apo_niv_200id,
savefasta_as, apo_niv_ids, LargeFastaFetch, slen_summ, apikey)

#### 5. Explore and Process Data Set ----

# Locate organism names from within the title, and put into separate columns
```

```
# Start with new dataframe to avoid making irreversible errors in original data
dfApoNiv <- apo_niv_fetch

# Make column for genus names
dfApoNiv$Genus <- word(dfApoNiv$Title, 2L)
# Check what genera are in the set, and if there are any unexpected words extracted
unique(dfApoNiv$Genus)
# I have Niviventer and Apodemus as expected, but also Sylvaemus.
# Let's have a closer look at the items labelled Sylvaemus
dfApoNiv[grep("Sylvaemus", dfApoNiv$Genus),1]
length(grep("Sylvaemus", dfApoNiv$Genus))
# They are all within the same species, Sylvaemus uralensis, and there are 33 samples.
# According to Juškaitis et al., Sylvaemus uralensis is an alternative name for
Apodemus uralensis (2016). They indicate that Apodemus uralensis is the commonly
used global name, but Sylvaemus uralensis is popular in Russian publications.
# Check if samples named Apodemus uralensis also exist within this data set
length(grep("Apodemus uralensis", dfApoNiv$Title))
# 87 samples are named as Apodemus uralensis.

# To simplify the data, rename Sylvaemus uralensis by its more common equivalent,
Apodemus uralensis.
# At the same time, create a separate column for full species, following G.Species
convention
# Start with empty column for species information
dfApoNiv$Species <- ""
# For each row in the dataframe, check which of the 3 genera in the set is present, then
abbreviate the full species name accordingly
for (row_ct in seq(1, nrow(dfApoNiv))) {
  if (dfApoNiv[row_ct,"Genus"] == "Apodemus") {
    dfApoNiv[row_ct,"Species"] <- paste("A.", word(dfApoNiv[row_ct, "Title"], 3L))
  } else if (dfApoNiv[row_ct, "Genus"] == "Niviventer") {
    dfApoNiv[row_ct, "Species"] <- paste("N.", word(dfApoNiv[row_ct, "Title"], 3L))
  } else { # Only 3 genera in the set, so third does not need a conditional, as it is all that
remains
    dfApoNiv[row_ct, "Genus"] <- "Apodemus"
    dfApoNiv[row_ct, "Species"] <- paste("A.", word(dfApoNiv[row_ct, "Title"], 3L))
  }

}

# Check Genus column
unique(dfApoNiv$Genus)
# Only Niviventer and Apodemus remain, as expected
# Check number of samples in A.uralensis species
```

```
length(grep("A. uralensis", dfApoNiv$Species))
# 120 samples are within A.uralensis, which is the sum total of the 33 that were formerly identified as S. uralensis and the existing 87 A. uralensis.
# Check Species column
unique(dfApoNiv$Species)
length(unique(dfApoNiv$Species))
# There are 44 unique species. Lots to work with.

# Retain the unique id for each sequence
dfApoNiv$ID <- word(dfApoNiv$Title, 1L)
# Verify that the IDs are unique
length(unique(dfApoNiv$ID)) # 4500, just like total sample number

# Organize columns of relevant information
dfApoNiv <- dfApoNiv[, c("ID", "Genus", "Species", "Sequence")]

# Check composition of sequences
# Do any sequences have unidentified nucleotides?
grep("N", dfApoNiv$Sequence)

# Seems like a lot! Let's see how many sequences have N's
length(grep("N", dfApoNiv$Sequence))

# 322 have at least one "N" in the sequence. How many have more than 5% N's?
dfApoNiv %>%
  filter(str_count(Sequence, "N") > (0.05 * str_count(Sequence))) %>%
  count()

# Only 9 have greater than 5% unknown nucleotides.

# How many more get filtered if the threshold is tighter, like 1%? And which genera would be lost?
dfApoNiv %>%
  filter(str_count(Sequence, "N") > (0.01 * str_count(Sequence))) %>%
  group_by(Genus) %>%
  count()

# 37 get filtered with more than 1% N's, 36 of which are Apodemus. But there are more Apodemus in the full set to begin with, and the number of sequences lost is still quite low, so I will go with a 1% N threshold.

# I don't think NCBI data will have gap markers, but I should check
grep("-", dfApoNiv$Sequence)
# Answer is none. good.
```

```
# What about total sequence length?
summary(nchar(dfApoNiv$Sequence))
# As with the sample set explored earlier, the relatively high median and mean indicate
that most sequences are greater than 1000 nucleotides in length, but there are a
number of shorter sequences that can be removed for analysis.

# Refine set to only include sequences with less than 1% unidentified nucleotides and
longer than 600bp
dfApoNiv <- dfApoNiv %>%
  filter(str_count(Sequence, "N") <= (0.01 * str_count(Sequence)) &
str_count(Sequence) > 600)

# Check out sequence length properties of remaining sequences
class(dfApoNiv$Sequence)
mean(nchar(dfApoNiv$Sequence))
summary(nchar(dfApoNiv$Sequence))
# This shows the range of sequence length remaining is 604:1177, with a median of
1131

# How do sequence length properties vary by Genus?
dfApoNiv %>%
  group_by(Genus) %>%
  summarise(min = min(nchar(Sequence)), median = median(nchar(Sequence)), max =
max(nchar(Sequence)))
# The range and medians are similar for both groups: Apodemus has min=627,
median=1113, and max=1177; Niviventer has min=604, median=1139, and max=1143.

# A histogram overlaying the two genera might help visualize the distribution of
sequence lengths

ggplot(dfApoNiv, aes(x = nchar(Sequence), fill = Genus, color = Genus)) +
  geom_histogram(binwidth = 25, position = "identity", alpha = 0.5) +
  geom_vline(aes(xintercept = median(nchar(Sequence)), color = Genus),
             linetype = "dashed") +
  scale_color_brewer(palette = "Dark2") +
  labs(title = "Sequence Lengths by Genus", x = "Sequence Length", y = "Frequency") +
  annotate(geom = "text", x = 800, y = c(923, 823, 523, 423), label = c("Apodemus
Range = 627:1177", "Apodemus Median = 1113", "Niviventer Range = 604:1143",
"Niviventer Median = 1139")) +
  ggsave("A2_fig2.pdf", width = 6, height = 6)

# If you don't wish to keep the file, you can use this line
#file.remove("A2_fig2.pdf")
```

CODE SECTION 2 – MAIN ANALYSIS

6. Build Training and Testing Set ----

```
# Set sequences to DNAStringSet to be able to use specific functions on them
dfApoNiv$Sequence <- DNAStringSet(dfApoNiv$Sequence)
# Check
class(dfApoNiv$Sequence)

# Calculate letter frequencies
dfApoNiv <- cbind(dfApoNiv, as.data.frame(letterFrequency(dfApoNiv$Sequence, letters
= c("A", "C","G", "T"))))

# Calculate independent letter frequencies, that is for 3 of the 4 letters, because the 4th
is dependent on the other three. I will choose to calculate A, C, and G. These are like k-
mers of length 1.
dfApoNiv$Aprop <- (dfApoNiv$A) / (dfApoNiv$A + dfApoNiv$T + dfApoNiv$C +
dfApoNiv$G)
dfApoNiv$Cprop <- (dfApoNiv$C) / (dfApoNiv$A + dfApoNiv$T + dfApoNiv$C +
dfApoNiv$G)
dfApoNiv$Gprop <- (dfApoNiv$G) / (dfApoNiv$A + dfApoNiv$T + dfApoNiv$C +
dfApoNiv$G)

# Next, add dinucleotide frequencies (k-mers of length 2)
dfApoNiv <- cbind(dfApoNiv,
as.data.frame(dinucleotideFrequency(dfApoNiv$Sequence, as.prob = TRUE)))

# And finally, although this level of specificity may not be needed, calculate trinucleotide
frequencies (k-mers of length 3)
dfApoNiv <- cbind(dfApoNiv,
as.data.frame(trinucleotideFrequency(dfApoNiv$Sequence, as.prob = TRUE)))

# With the kmers prepared, it is time to build the training and testing sets

# To create training and testing sets, and continue analysis, convert the DNAStringSet
data type of sequences back to character data type.
dfApoNiv$Sequence <- as.character(dfApoNiv$Sequence)
# Check
class(dfApoNiv$Sequence)

# Determine how to allocate sequences to training or test set

# How many of each genus are there?
```

```
dfApoNiv %>%
  group_by(Genus) %>%
  count()
# There are 2490 Apodemus, 1423 Niviventer

# Because there are different numbers of sequences in each genus, I will use a data partition to proportionally draw 75% of each genus for the training set, and reserve 25% for the testing set.

# Set seed for reproducibiliy
set.seed(1254)
# Create index to split based on labels
train_index <- createDataPartition(dfApoNiv$Genus, p = 0.75, list = FALSE)

# Subset training set with training index
genus.training <- dfApoNiv[train_index,]

# Subset test set as everything not in the training index
genus.test <- dfApoNiv[-train_index,]

# Check numbers of each genus in training and test set
genus.training %>%
  group_by(Genus) %>%
  count()
genus.test %>%
  group_by(Genus) %>%
  count()

# Portions seem good!

# Clean environment
rm(row_ct, train_index)


#### 7. Build & Test Genus Classifiers ----

# Random Forest builds many decision trees and aggregates decisions.
# Random Forest will be used first as done in class, directly through the randomForest package, but then within caret for comparison because caret is a useful package to learn that includes a wide variety of models to explore and compare.

# Locate columns of interest
names(genus.training)
```

```
# Start with RandomForest, as described in class, using k-mers of length 1 to start, which are Aprop, Cprop, and Gprop in columns 9-11
set.seed(5329)
model_rf_genus <- randomForest(x = genus.training[, 9:11], y = as.factor(genus.training$Genus), ntree = 500, importance = TRUE)
model_rf_genus
# This showed an OOB estimate of error rate at 0.65%. This is excellent! The confusion matrix shows only 4 misclassified Apodemus specimens (0.2%), and 15 misclassified Niviventer specimens (1.4%).

# Using same parameters, make a random forest model using caret - commented out the code, because the processing time is too long.
#set.seed(5329)
#model_rfc_genus <- train(genus.training[, 9:11], genus.training[, 2], method = "rf", ntree = 500, importance = TRUE)
#model_rfc_genus$finalModel

# To verify processing times, the model training commands were passed to the system.time function, which indicated that the elapsed time for the randomForest training was 1.127s, and for the caret random forest training was 78.150s. Commands are below if you wish to check processing speed on your machine.
#system.time(model_rf_genus <- randomForest(x = genus.training[, 9:11], y = as.factor(genus.training$Genus), ntree = 500, importance = TRUE))
#system.time(model_rfc_genus <- train(genus.training[, 9:11], genus.training[, 2], method = "rf", ntree = 500, importance = TRUE))

# Troubleshooting the processing time through GitHub Issues, default caret models are produced with resampling that is not being done in default randomForest training. This can be adjusted to make the processes (and corresponding processing time) more comparable by changing the train controls

# Define train control method as "none" to eliminate resampling
fitcontrols <- trainControl(method = "none")
# Run same model again with new control
set.seed(5329)
model_rfc_genus2 <- train(genus.training[, 9:11], genus.training[, 2], method = "rf", trControl = fitcontrols, ntree = 500, importance = TRUE)
system.time(model_rfc_genus2 <- train(genus.training[, 9:11], genus.training[, 2], method = "rf", trControl = fitcontrols, ntree = 500, importance = TRUE))
# Elapsed time is now a more comparable 1.748s
model_rfc_genus2$finalModel
# This shows an OOB estimate of error rate at 0.75%. The confusion matrix shows only 5 misclassified Apodemus specimens (0.3%), and 17 misclassified Niviventer specimens (1.6%).
```

```
# This adjusted caret model will be used to compare to the randomForest prediction
results

# Using two models created, generate predictions from testing set
pred_rf_genus <- predict(model_rf_genus, genus.test[, 9:11])
pred_rfc_genus <-predict(model_rfc_genus2, genus.test[, 9:11])

# Evaluate the predictions
# Confusion matrix for basic package randomForest model
table(observed = genus.test$Genus, predicted = pred_rf_genus)
# 4 Apodemus were misclassified as Niviventer, and 13 Niviventer were misclassified as
Apodemus
# Confusion matrix for caret package randomForest model
table(observed = genus.test$Genus, predicted = pred_rfc_genus)
# Nearly the same as the basic randomForest model, 3 Apodemus were misclassified
as Niviventer, and 13 Niviventer were misclassified as Apodemus

# Did the same sequences get misclassified in both models?
# Generate dataframe of data for sequences misidentified in randomForest model
wrong_genus_rf <- slice(genus.test, which(genus.test$Genus != pred_rf_genus))
# Generate dataframe of data for sequences misidentified in caret's random forest
model
wrong_genus_rfc <- slice(genus.test, which(genus.test$Genus != pred_rfc_genus))
# Check object features
class(wrong_genus_rf)
length(wrong_genus_rf$ID)
length(wrong_genus_rfc$ID)
# As expected, there are 17 samples in the misclassified set from randomForest, and 16
from caret

# Check the expected number of each misclassified genera are in the new dataframes
wrong_genus_rf %>%
  group_by(Genus) %>%
  dplyr::count()

wrong_genus_rfc %>%
  group_by(Genus) %>%
  dplyr::count()

# The same genus profiles are shown as was in the confusion matrices

# Were the same samples misclassified in both models?
```

```
wrong_genus_same <- wrong_genus_rf[wrong_genus_rf$ID %in%
wrong_genus_rfc$ID,]
length(wrong_genus_same$ID)

# There are 16 results that overlap between the two sets, which is the entire size of the
smaller set, indicating that the same sequences are being misidentified in both models

# Verify that they are the same
wrong_genus_rf[!(wrong_genus_rfc$ID %in% wrong_genus_rf$ID),"ID"]
# Zero answers returned, as expected

# Nearly the same from both methods.

# I will carry on using only the randomForest package for simplicity.

#Although less than 1% error is pretty decent, let's see if using k-mer of length 2 will
improve the accuracy

# Locate columns of dinucleotides
names(dfApoNiv)
# Dinucleotide frequencies are columns 12:27. Use the same training and testing sets
set.seed(331)
model_rf_genus_di <- randomForest(x = genus.training[, 12:27], y =
as.factor(genus.training$Genus), ntree = 500, importance = TRUE)
model_rf_genus_di

# The OOB estimate of error rate is now as low as 0.37%. Computing time was still
quite quick.

# Let's test the new 2mer model with the testing set to see how it performs
pred_rf_genus_di <-predict(model_rf_genus_di, genus.test[, 12:27])
# Confusion matrix to see where the errors are
table(observed = genus.test$Genus, predicted = pred_rf_genus_di)
# Looks great! This time all Apodemus were classified correctly, and only 6 Niviventer
were identified as Apodemus.

# Check which sequences were misidentified
wrong_genus_rfdi <- slice(genus.test, which(genus.test$Genus != pred_rf_genus_di))
# Check object features
class(wrong_genus_rfdi)
length(wrong_genus_rfdi$ID)
# Looks as expected. What is the overlap with the previous set of misclassified
sequences?
```

```
wrong_genus_same2 <- wrong_genus_rf[wrong_genus_rf$ID %in%
wrong_genus_rfdi$ID,]
wrong_genus_same2$ID
length(wrong_genus_same2$ID)
# All 6 misclassified are in the same set as with the kmers of length 1, indicating that
these are less typical than the rest of the sequences for their genus, but the model is
getting more specific.

# Which species are being misclassified?
unique(wrong_genus_same2$Species)
# Three species: "N. confucianus" "N. fulvescens"  "N. huang"

# One last refinement with kmers of length 3 to see if the model can trained to be more
accurate

# Locate columns of trinucleotides
names(dfApoNiv)
# Trinucleotide frequencies are columns 28:91. Use the same training and testing sets
set.seed(202020)
model_rf_genus_tri <- randomForest(x = genus.training[, 28:91], y =
as.factor(genus.training$Genus), ntree = 500, importance = TRUE)
# Computation time was still pretty quick
model_rf_genus_tri
# This model misclassifies 1 Apodemus and 2 Niviventer during the build, with an OOB
error of 0.1%

# Let's see the model performs with the test set.
pred_rf_genus_tri <-predict(model_rf_genus_tri, genus.test[, 28:91])
# Confusion matrix to see where the errors are
table(observed = genus.test$Genus, predicted = pred_rf_genus_tri)
# Looks great! This time all Apodemus were classified correctly, and only 3 Niviventer
were identified as Apodemus.

# Check which sequences were misidentified
wrong_genus_rftri <- slice(genus.test, which(genus.test$Genus != pred_rf_genus_tri))
# Check object features
length(wrong_genus_rftri$ID)
# Looks as expected. What is the overlap with the previous set of misclassified
sequences?
wrong_genus_same3 <- wrong_genus_rfdi[wrong_genus_rfdi$ID %in%
wrong_genus_rftri$ID,]
wrong_genus_same3$ID
length(wrong_genus_same3$ID)
# Yes all three are the same.
```

```
# Which species are misclassified with this model?
unique(wrong_genus_same3$Species)
# All three are "N. confucianus" this time.

# Clean environment
rm(model_rf_genus, model_rf_genus_di, model_rfc_genus, model_rfc_genus2,
pred_rf_genus, train_index, wrong_genus_rf, wrong_genus_rfc, wrong_genus_rfdi,
wrong_genus_same, wrong_genus_same2, pred_rf_genus, pred_rfc_genus,
pred_rf_genus_di, fitcontrols)


#### 8. Test of Classifier at Species Level ----

# What are the 3 most populous species in the existing genus.training set? Retaining
same training and testing sets for consistency when comparing accuracy of models.
genus.training %>%
  group_by(Species) %>%
  dplyr::count(sort = T)
# Top three species are A. sylvaticus (611), A. agrarius (309), N. confucianus (306).

# Are the same species most populous in the test set?
genus.test %>%
  group_by(Species) %>%
  dplyr::count(sort = T)
# Yes, the same three species are most populous.

# Subset genus training set to only include top three species
species.training <- genus.training[genus.training$Species %in% c("A. sylvaticus", "A.
agrarius", "N. confucianus"),]
# Check
unique(species.training$Species)
# As expected

# Subset genus test set to only include top three species
species.test <- genus.test[genus.test$Species %in% c("A. sylvaticus", "A. agrarius", "N.
confucianus"),]
# Check
unique(species.test$Species)
# As expected

# Pass species training set to same randomForest parameters as used for genus
classifier
set.seed(80082)
```

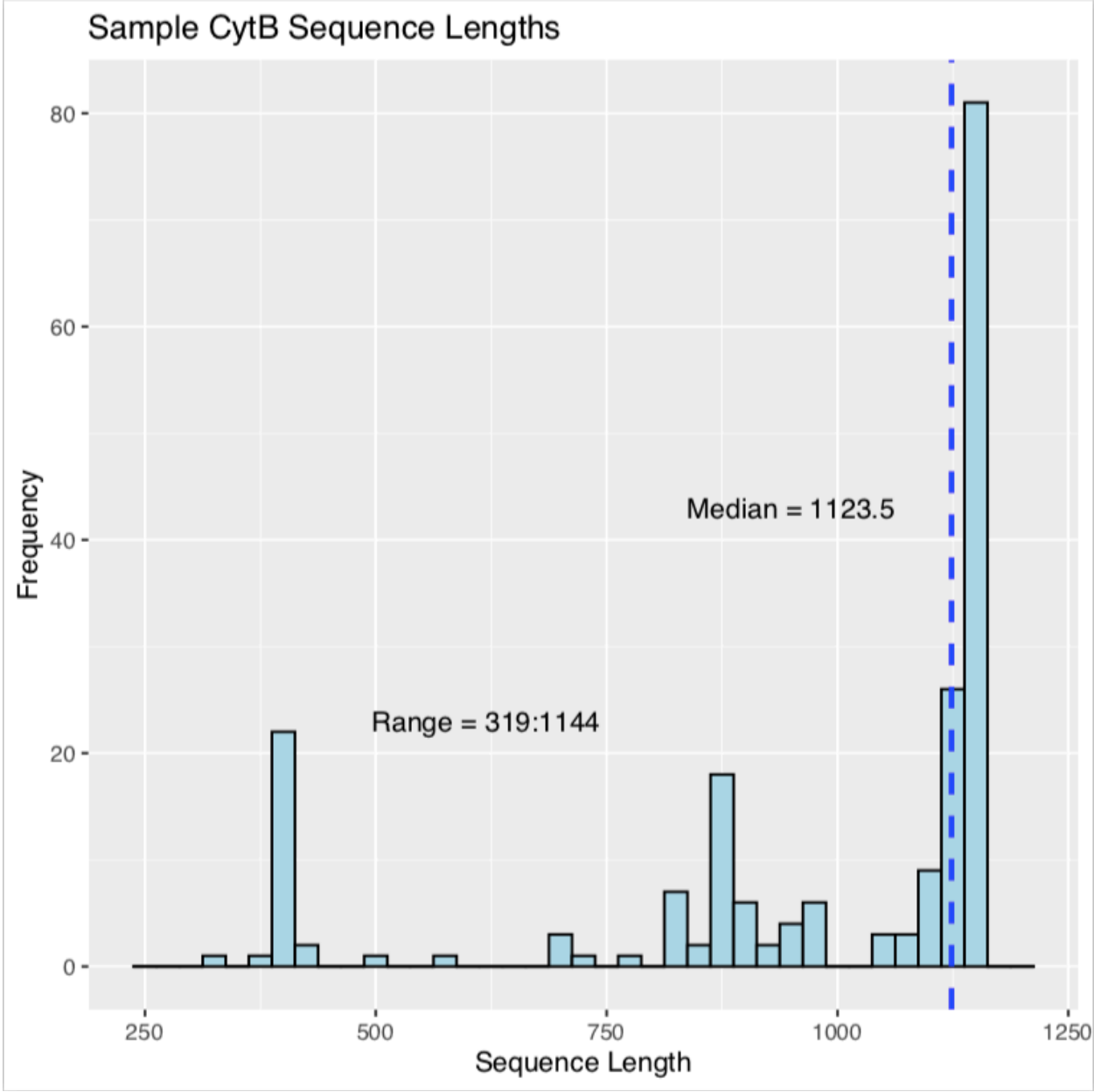
```
model_rf_species_tri <- randomForest(x = species.training[, 28:91], y =
as.factor(species.training$Species), ntree = 500, importance = TRUE)
# Computation time was still pretty quick
model_rf_species_tri
# This model is 100% successful within the training group. Very nice! The three species
must be reasonably distinct at their CytB gene.

# Use test set to predict species of data not used in training set.
pred_rf_species_tri <-predict(model_rf_species_tri, species.test[, 28:91])

# Confusion matrix to see where the errors are
table(observed = species.test$Species, predicted = pred_rf_species_tri)
# Looks great! All A.agrarius and A.sylvaticus were classified correctly, and only 4
N.confucianus were identified as misidentified.

# Check which sequences were misidentified
wrong_species_rftri <- slice(species.test, which(species.test$Species !=
pred_rf_species_tri))
# Check object features
length(wrong_species_rftri$ID)
# Looks as expected. What is the overlap with the previous set of misclassified
sequences?
wrong_species_same <- wrong_genus_rftri[wrong_genus_rftri$ID %in%
wrong_species_rftri$ID,]
wrong_species_same$ID
length(wrong_species_same$ID)
# Yes all three that overlap are the same.
# Verify species are N. confucianus, as expected from genus classifier
unique(wrong_species_same$Species)
# As expected.


#### 9. Explore Sequences Consistently Misidentified ----

# I will compare importance of variables to classification at genus level.

# Let's take a look at the importance of the trimers in the model.
# Mean Decrease Accuracy (MDA) indicates the loss in predictive ability if a particular
variable is removed from training. Mean Decrease Gini (MDG) describes uniformity of
terminal nodes in the forest (Coding Resource: Oxford Protein Informatics Group
(OPIG), “A very basic introduction to Random Forests using R”)

# What are the most influential parts of the classification variables?
model_rf_genus_tri$importance
```

```
# The importance for the trimers varies considerably

# What are the top four trimers in MDA?
impMDA <- as.data.frame(model_rf_genus_tri$importance) %>%
  arrange(desc(MeanDecreaseAccuracy))
impMDA[1:4,]

# What are the top four trimers in MDG?
impMDG <- as.data.frame(model_rf_genus_tri$importance) %>%
  arrange(desc(MeanDecreaseGini))
impMDG[1:4,]

# The top four impact variables in terms of both Mean Decrease Accuracy and Mean
Decrease Gini are ATA, AAT, TAA, and CTT.

# Let's see how different the trimer proportions are in the correctly identified Apodemus
and Nivivneter species, vs. the three misidentified samples

# Generate column of Classification, which will describe the genus of the sample if
identified correctly in the final model, or Wrong #1, Wrong #2, and Wrong #3 for the
three samples that were labelled as N. confucianus but classified as Apodemus.
dfApoNiv$Classification <- ""
num.wrong <- 1
for (row_ct in seq(1:length(dfApoNiv$ID))) {
  if (dfApoNiv[row_ct, "ID"] %in% wrong_genus_same3$ID) {
    dfApoNiv[row_ct, "Classification"] <- paste("Wrong #", num.wrong, sep = "")
    num.wrong <- num.wrong + 1
  } else {
    dfApoNiv[row_ct, "Classification"] <- dfApoNiv[row_ct, "Genus"]
  }
}

# Generate boxplots to show frequencies and outliers for each of the model's high
importance trimers

pATA <- ggplot(dfApoNiv, aes(x = ATA, y = as.factor(Classification))) +
  geom_boxplot(outlier.color = "blue") +
  labs(x = "ATA Frequency", y = "Classification")

pAAT <- ggplot(dfApoNiv, aes(x = AAT, y = as.factor(Classification))) +
  geom_boxplot(outlier.color = "yellow") +
  labs(x = "AAT Frequency", y = "Classification")
```

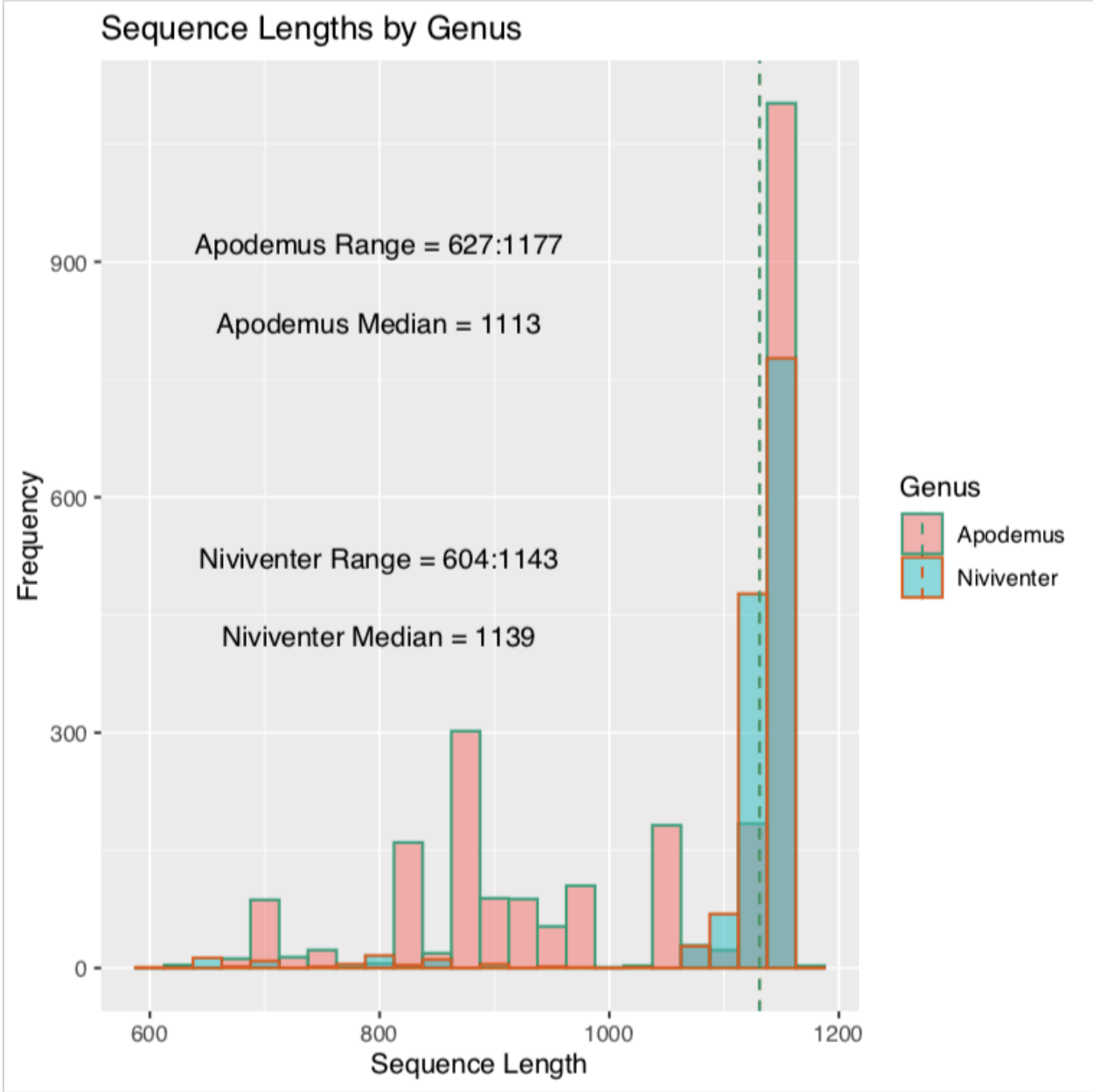
```
pTAA <- ggplot(dfApoNiv, aes(x = TAA, y = as.factor(Classification))) +
  geom_boxplot(outlier.color = "green") +
  labs(x = "TAA Frequency", y = "Classification")

pCTT <- ggplot(dfApoNiv, aes(x = CTT, y = as.factor(Classification))) +
  geom_boxplot(outlier.color = "red") +
  labs(x = "CTT Frequency", y = "Classification")

# Arrange the boxplots into a single, easy to view figure
grid.arrange(pATA, pAAT, pTAA, pCTT, nrow = 2, top = "Frequency of High Importance
Trimers")

# Save figure for submitting with assignment
pdf("A2_fig3.pdf", width = 6, height = 6)
grid.arrange(pATA, pAAT, pTAA, pCTT, nrow = 2, top = "Frequency of High Importance
Trimers")
dev.off()

# If you don't wish to keep the file, you can use this line
#file.remove("A2_fig3.pdf")

# It's no wonder they keep getting misclassified! The samples of N. confucianus that are
being identified as Apodemus genus are outliers to the frequency of each the top four
importance trimers, and similar to the Apodemus frequencies.
```

VISUALIZATIONS

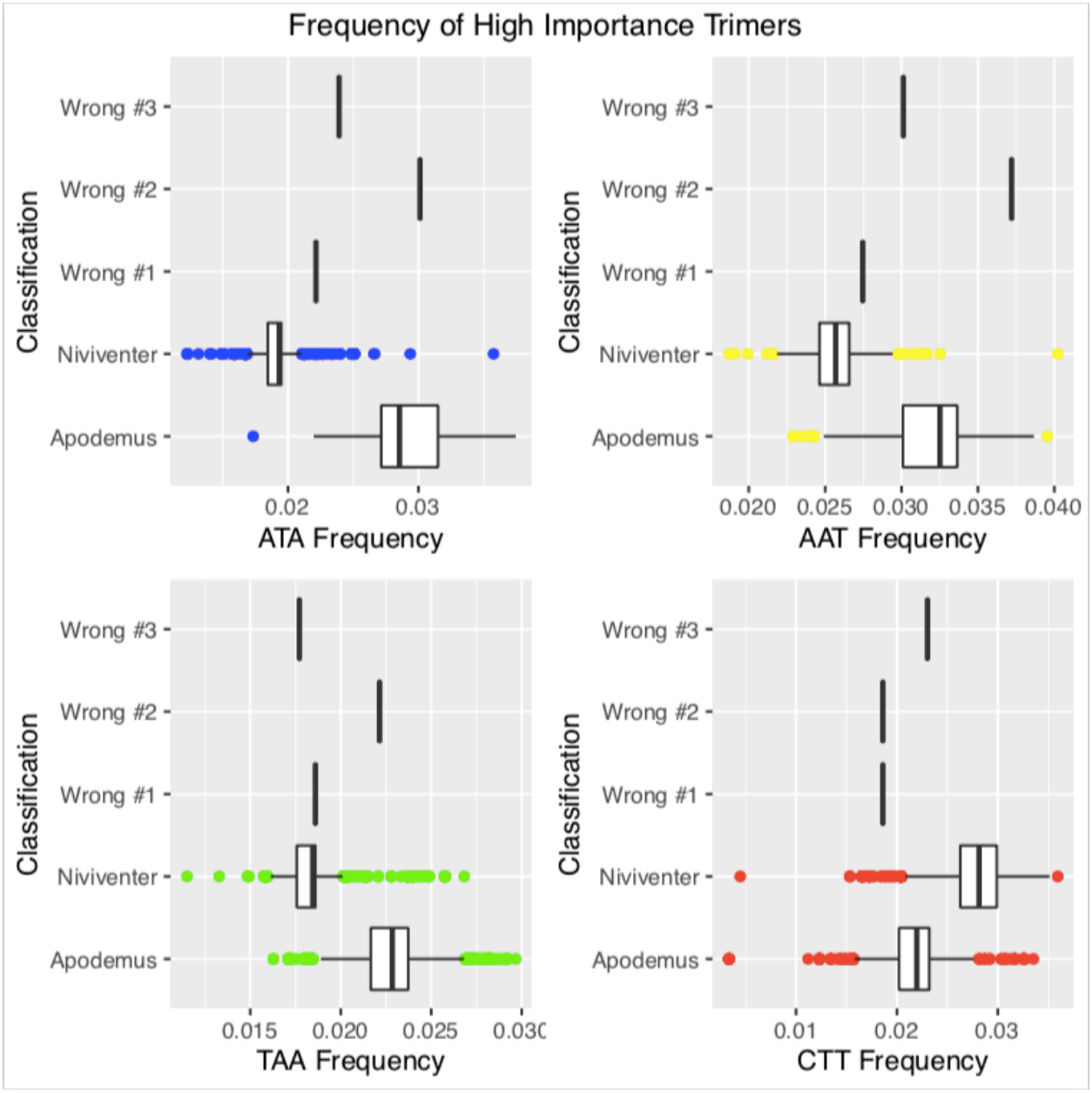
Visualization 1 from Coding Section 1. Lines 90-96 of code. Saved as A2_fig1.pdf.



Visualization 2 from Coding Section 1. Lines 273-280 of code. Saved as A2_fig2.pdf.



Visualization 1 from Coding Section 2. Lines 594-611 of code. Saved as A2_fig3.pdf



RESULTS AND DISCUSSION

The presence of samples labelled by an alternate name for Apodemus uranalsis, Sylvaemus uranalsis (Juškaitis et al., 2016), demonstrated the importance of quality control and checking the data for unexpected features. The randomForest classifier built with trimers from the mitochondrial gene cytochrome B was more than 99.5% successful at classifying the sequences to genus resolution for Apodemus and Niviventer rodents, with only three specimens labelled N. confucianus that were misclassified as Apodemus. Taking the same classification parameters to species level identification, the same three specimens were mislabelled, but the algorithm was otherwise a success.

Preliminary exploration of the importance of variables to the model showed that the four most influential trimers in the classifier were outliers for the three misclassified samples relative to their labelled identity. If this project was to be run again at a larger scale, it would be helpful to do further analysis on the sequences that the algorithm is classifying differently than the label. The three misclassified sequences could be aligned with a sample of Niviventer sequences and with Apodemus sequences, for example through DECHIPHER in R, or run through BLAST to see if they were likely misidentified in the original data. An expansion of this project could also include further explorations of the different models being offered within the caret package.

ACKNOWLEDGEMENTS

Help from others was requested on the course Discussion Board regarding fetching datasets larger than the maximum request allowed by a single permutation of the entrez_fetch() function. Sample scripts provided by Jason Moggridge and Jacqueline May were invaluable to coding my version of the large data fetch.

REFERENCES

ACADEMIC PAPERS

Heaney, Lawrence R.; Balete, Danilo S.; Rickart, Eric A.; Alviola, Phillip A.; Duya, Mariano Roy M.; Duya, Melizar V.; Veluz, M. Josefa; Vandevrede, Lawren; Steppan, Scott J. (2011). "Chapter 1: Seven New Species and a New Subgenus of Forest Mice (Rodentia: Muridae: Apomys) from Luzon Island". Fieldiana Life and Earth Sciences. 2 (2): 1–60. doi:10.3158/2158-5520-2.1.1.

Juškaitis, R., Balčiauskas, L., & Alejūnas, P. (2016). Distribution, habitats and abundance of the herb field mouse (Apodemus uralensis) in Lithuania. Biologia, 71(8), 960–965. https://doi.org/10.1515/biolog-2016-0116

Ratnasingham, S., & Herbert, P. (2007). bold: The Barcode of Life Data System (http://www.barcodinglife.org). Molecular Ecology Notes, 7(3), 355–364. https://doi.org/10.1111/j.1471-8286.2007.01678.x

CODING RESOURCES

#Help with ggplot2 (general)
ggplot2(part of the tidyverse), “Reference”
https://ggplot2.tidyverse.org/reference/

#Help with ggplot2 (histograms)
Statistical tools for high-throughput data analysis, “ggplot2 histogram plot : Quick start guide - R software and data visualization”
http://www.sthda.com/english/wiki/ggplot2-histogram-plot-quick-start-guide-r-software-and-data-visualization

#Help with randomForest Package
(2018-03-25) Package ‘randomForest’
https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

#Help with randomForest function
Liaw, A. “Classification And Regression With Random Forest”

https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest

#Troubleshooting caret package processing slow
GitHub, “caret seems to be really slow #108”
https://github.com/topepo/caret/issues/108

#Help with caret functions, arguments, and features, specifically trainControl()
Kuhn, M. (2019-03-27), “The caret Package”
http://topepo.github.io/caret/index.html

#Interpreting importance in randomForest model
Oxford Protein Informatics Group (OPIG), “A very basic introduction to Random Forests using R”
https://www.blopig.com/blog/2017/04/a-very-basic-introduction-to-random-forests-using-r/

#Arranging subfigures into a single grid figure
Auguié, B (2019-07-13). “Laying out multiple plots on a page”
https://cran.r-project.org/web/packages/egg/vignettes/Ecosystem.html