###In this project we used two different machine learning techniques in a supervised machine learning approach to separate Cytochrome c oxidase 1 (COI) and Cytochrome b (cytb) sequences within the family Muridae. The objective was to investigate whether one classifier outperforms the other when differentiating the two genes. The Muridae family was chosen so sufficient IDs can be obtained for our classification, they are the largest family of rodents and mammals, and they also contain one of the most studied genus Mus (Aghová et al., 2018). COI and cytB are protein coding genes, encoded by the mitochondria, in the past they have been used for phylogenetic analysis at various taxonomic levels as well as successfully estimating divergence (Kartavtsev, 2011). However, because they are both protein coding and encoded by the mitochondria, they are likely to have similar sequences. Therefore, it is interesting to explore whether their sequences have sufficient variability within the Muridae family to be considered distinct. The two classifiers used in our project are Random Forest and Naive Bayes classifier. The Random Forest classifier uses an ensemble algorithm, which was first introduced Breiman (Latief et al.,2019). It creates multiple decision trees through regression models and each tree predicts an outcome, the one with the most "votes" is used for classification (Yiu, 2021; Latief et al.,2019). Some advantages of the random forest classifier include: efficiency, only few parameters are needed in comparison to other models, and it is not sensitive to over fitting (Latief et al.,2019). The Naive Bayes is based on Bayes theorem and uses a probabilistic prediction approach to create a model (Latief et al.,2019). The term Naive is reflects that the predictors used are independent of each other(Gandhi, 2018). Some advantages of the Naive Bayes model include accuracy and speed.

###calling all packages required:

#Packages from CRAN:

#install tidyverse package, if needed, then load.

#install.packages("tidyverse")

library(tidyverse)

#To install randomForest package, if needed and then load

#install.packages("randomForest")

library(randomForest)

#install.packages("rentrez")

library(rentrez)

#install.packages("seqinr")

library(seqinr)

#install.packages("naivebayes")

library(naivebayes)

#install.packages("psych")

library(psych)

#install.packages("ggplot")

library(ggplot2)

#Package from Bioconductor:

#To install Bioconductor packages

#install.packages("BiocManager")

#library(BiocManager)

#BiocManager::install("Biostrings")

library(Biostrings)

####Data Acquisition

#Look for searchable fields in nuccore database to determine which ones will be used for analysis

entrez\_db\_searchable(db = "nuccore")

#Create function to fetch fasta files required for analysis from Entrez\_fuction script used in class

FetchFastaFiles <- function(searchTerm, seqsPerFile = 100, fastaFileName) {

# This function will fetch FASTA files from NCBI nuccore based on a provided search term.

# searchTerm = character vector containing Entrez search term

# seqsPerFile = number of sequences to write to each FASTA file

# fastaFileName = character vector containing name you want to give to the FASTA files you are fetching

# Initial search for finding maximum number of hits

search1 <- entrez\_search(db = "nuccore", term = searchTerm)

# Second search for obtaining max number of hits and their IDs

search2 <- entrez\_search(db = "nuccore", term = searchTerm, retmax = search1$count, use\_history = T)

# Fetch the sequences in FASTA format using the web\_history object.

for (start\_rec in seq(0, search2$retmax, seqsPerFile)) {

fname <- paste(fastaFileName, start\_rec, ".fasta", sep = "")

recs <- entrez\_fetch(db = "nuccore", web\_history = search2$web\_history, rettype = "fasta", retstart = start\_rec, retmax = seqsPerFile)

write(recs, fname)

print(paste("Wrote records to ", fname, sep = ""))

}

return(search2)

}

##Using the function to fetch fasta files

#Searching for Muridae (common and scientific names using ORGN] and gene CytB in nuccore database (nucleotide database). The length of cytB gene sequence length is restricted to 600-1000bp to prevent whole mitochondrial genomes from being incorporated. Each fasta file will contain 1000 sequences

Muridae\_cytB<-FetchFastaFiles("Muridae[ORGN] AND CytB[Gene] AND 600:1000[SLEN]", 1000, "Muridae\_cytB")

#Searching for Muridae (common and scientific names using ORGN] and gene COI in nuccore database (nucleotide database). The length of COI gene sequence length is restricted to 600-700bp to prevent whole mitochondrial genomes from being incorporated. Each fasta file will contain 1000 sequences

Muridae\_COI<-FetchFastaFiles("Muridae[ORGN] AND COI[Gene] AND 400:700[SLEN]", 1000, "Muridae\_COI")

###Creating a function merge the all the fasta files together using Entrez\_fuction script used in class ito one dataframe

MergeFastaFiles <- function(filePattern) {

# This function merges multiple FASTA files into one dataframe.

# filePattern = Character vector containing common pattern in FASTA file names

# Read the FASTA files in.

fastaFiles <- list.files(pattern = filePattern)

l\_fastaFiles <- lapply(fastaFiles, readDNAStringSet)

# Convert them into dataframes.

l\_dfFastaFiles <- lapply(l\_fastaFiles, function(x) data.frame(Title = names(x), Sequence = paste(x) ))

# Combine the list of dataframes into one dataframe.

Muridae\_COIs <- do.call("rbind", l\_dfFastaFiles)

return(Muridae\_COIs)

}

#Creating dataframes

Muridae\_COI<-MergeFastaFiles("Muridae\_COI\*")

Muridae\_cytB<-MergeFastaFiles("Muridae\_cytB\*")

#cleaning up species names so they are more easily readable using stringr package and the function word(). This constricts the title to only species name, which are specified by 2L and 3L and a new column called species name is created

Muridae\_COI$Species\_name <- word(Muridae\_COI$Title, 2L, 3L)

#Rearranging columns so species name comes after title, followed by sequence

Muridae\_COI <- Muridae\_COI[, c("Title", "Species\_name", "Sequence")]

#Same for cytB sequences

Muridae\_cytB$Species\_name <- word(Muridae\_cytB$Title, 2L, 3L)

Muridae\_cytB <- Muridae\_cytB[, c("Title", "Species\_name", "Sequence")]

####Data Exploration

#Viewing data frames

View(Muridae\_COI)

View(Muridae\_cytB)

#summary

summary(Muridae\_COI)

summary(Muridae\_cytB)

#check column names

names(Muridae\_COI)

names(Muridae\_cytB)

#To see wide range of sequence lengths for COI we will create a histogram with sequence length on the x-axis and frequency on y =-axis

hist(nchar(Muridae\_COI$Sequence), xlab="Sequence length", ylab= "Frequency", main="Frequency histogram for COI sequence lengths")

Chart, histogram

Description automatically generated

Figure 1: Frequency of COI sequence lengths within the Muridae family

#same for cytB

hist(nchar(Muridae\_cytB$Sequence), xlab="Sequence length", ylab= "Frequency", main="Frequency histogram for cytB sequence lengths")

#cytB sequence length is more spread out compared to COI indicating variability

Chart, histogram

Description automatically generated

Figure 1: Frequency of cytB sequence lengths within the Muridae family

####Data Filtering

#Creating a new nucleotides column to clean up the sequences to have biological sequences rather than alignments.

#trimming ends, removing gaps and N's (may still have up to 5% Ns)

Muridae\_COI <- Muridae\_COI %>%

mutate(Sequence2 = str\_remove(Sequence, "^[-N]+")) %>%

mutate(Sequence2 = str\_remove(Sequence2, "[-N]+$")) %>%

mutate(Sequence2 = str\_remove\_all(Sequence2, "-+")) %>%

filter(str\_count(Sequence2, "N") <= (0.05 \* str\_count(Sequence)))

#Same for cytB

Muridae\_cytB <- Muridae\_cytB %>%

mutate(Sequence2 = str\_remove(Sequence, "^[-N]+")) %>%

mutate(Sequence2 = str\_remove(Sequence2, "[-N]+$")) %>%

mutate(Sequence2 = str\_remove\_all(Sequence2, "-+")) %>%

filter(str\_count(Sequence2, "N") <= (0.05 \* str\_count(Sequence)))

#Because the sequence range was so wide as can be seen in the histogram we will constrain seqeunces to above first quartile but below 3rd

#Determining first and third quartile sequence lengths for COI and cytB. This will tell us at what value of sequence length 25% of the observations fall below that sequence length and for 75% of sequence lengths fall above that length. This ensures that we aren't keeping lengths that are either too short or too long as they could have different properties

q1\_COI <- quantile(nchar(Muridae\_COI$Sequence2), probs = 0.25, na.rm = TRUE)

q1\_COI

q3\_COI <- quantile(nchar(Muridae\_COI$Sequence2), probs = 0.75, na.rm = TRUE)

q3\_COI

q1\_cytB <- quantile(nchar(Muridae\_cytB$Sequence2), probs = 0.25, na.rm = TRUE)

q1\_cytB

q3\_cytB <- quantile(nchar(Muridae\_cytB$Sequence2), probs = 0.75, na.rm = TRUE)

q3\_cytB

#We are now constraining sequence lengths to those that are greater than the first quartile but lesser than the third quartile.

Muridae\_COI <- Muridae\_COI %>%

filter((str\_count(Sequence2) >= q1\_COI & str\_count(Sequence2) <= q3\_COI))

#same for Muridae

Muridae\_cytB <- Muridae\_cytB %>%

filter((str\_count(Sequence2) >= q1\_cytB & str\_count(Sequence2) <= q3\_cytB))

#View to ensure our queries worked

View(Muridae\_COI)

View(Muridae\_cytB)

class(Muridae\_COI)

class(Muridae\_cytB)

##Caluculating Sequence features:

#converting sequence column to a DNAStringSet

Muridae\_COI$Sequence2 <- DNAStringSet(Muridae\_COI$Sequence2)

#same for cytB

Muridae\_cytB$Sequence2 <- DNAStringSet(Muridae\_cytB$Sequence2)

#to ensure it has been converted to a DNA String Set

class(Muridae\_COI$Sequence2)

class(Muridae\_cytB$Sequence2)

##Calculating Nucleotide frequency and binding the column to our dataframe. Alphabet frequency cannot be used here as it may lead to the incorporation of Ns (more than required, 5% may still be included because of how our data is filtered)

Muridae\_COI <- cbind(Muridae\_COI, as.data.frame(letterFrequency(Muridae\_COI$Sequence2, letters = c("A", "C","G", "T"))))

#same by cytB

Muridae\_cytB <- cbind(Muridae\_cytB, as.data.frame(letterFrequency(Muridae\_cytB$Sequence2, letters = c("A", "C","G", "T"))))

#view to ensure they have been incorporated as a column

View(Muridae\_COI)

View(Muridae\_cytB)

##Calculating proprtions of A, T C and G for COI.Cs not included as rest of the proportion is C

Muridae\_COI$Aprop <- (Muridae\_COI$A) / (Muridae\_COI$A + Muridae\_COI$T + Muridae\_COI$C + Muridae\_COI$G)

Muridae\_COI$Tprop <- (Muridae\_COI$T) / (Muridae\_COI$A + Muridae\_COI$T + Muridae\_COI$C + Muridae\_COI$G)

Muridae\_COI$Gprop <- (Muridae\_COI$G) / (Muridae\_COI$A + Muridae\_COI$T + Muridae\_COI$C + Muridae\_COI$G)

#same for CytB

Muridae\_cytB$Aprop <- (Muridae\_cytB$A) / (Muridae\_cytB$A + Muridae\_cytB$T + Muridae\_cytB$C + Muridae\_cytB$G)

Muridae\_cytB$Tprop <- (Muridae\_cytB$T) / (Muridae\_cytB$A + Muridae\_cytB$T + Muridae\_cytB$C + Muridae\_cytB$G)

Muridae\_cytB$Gprop <- (Muridae\_cytB$G) / (Muridae\_cytB$A + Muridae\_cytB$T + Muridae\_cytB$C + Muridae\_cytB$G)

##View to ensure proportions have been added

View(Muridae\_COI)

View(Muridae\_cytB)

##Adding dinucleotide k-mers of length 2 to account for sequence variability

Muridae\_COI <- cbind(Muridae\_COI, as.data.frame(dinucleotideFrequency(Muridae\_COI$Sequence2, as.prob = TRUE)))

#same for cytB

Muridae\_cytB <- cbind(Muridae\_cytB, as.data.frame(dinucleotideFrequency(Muridae\_cytB$Sequence2, as.prob = TRUE)))

##Adding dinucleotide k-mers of length 3 to account for sequence variability

Muridae\_COI <- cbind(Muridae\_COI, as.data.frame(trinucleotideFrequency(Muridae\_COI$Sequence2, as.prob = TRUE)))

Muridae\_cytB <- cbind(Muridae\_cytB, as.data.frame(trinucleotideFrequency(Muridae\_cytB$Sequence2, as.prob = TRUE)))

#add new column called 'Code' to COI sequences so they can easily be differentiated when dataframe is merged

Muridae\_COI$Code <- 'COI'

#same for cytB

Muridae\_cytB$Code <- 'cytB'

#Merging cytB and COI dataframes into one data frame

Muridae\_all<- rbind(Muridae\_COI, Muridae\_cytB)

#to enssure they have been merged correctly and have 5433 observations, 3632 from cyt\_B and 1801 from COI

table(Muridae\_all$Code)

View(Muridae\_all)

str(Muridae\_all)

head(Muridae\_all)

#to see most occuring species name in the data frame

sort(table(Muridae\_all$Species\_name))

#Since the data collected is either part of COI genes or cytB there are no missing cases, but if there were, only complete cases of genes and species name would have been taken and the rest would have been filtered out using complete.cases. The Muridae family is large enough that filtering out incomplete data would not impact our results.

###Main Analysis

###Creating Validating data set

#We want to see if COI and cytB have enough variation in their sequences to be classified separately

#Change sequence data back to character so it is easier to apply tidyverse functions

Muridae\_all$Sequence2<- as.character(Muridae\_all$Sequence2)

#The maximum sample size is 1801 as there are only 1801 sequences for COI. We will take 25% of the sample to be the validation dataset. This will separate from our training dataset.

.25\*1801

#We will take 450 samples from each data set which will be used as our validation dataset later on. To make our script reproducible we will be setting seed.

set.seed(221)

Muridae\_Validation <- Muridae\_all %>%group\_by(Code) %>% sample\_n(450)

#Checking sample size for each marker is same

table(Muridae\_Validation$Code)

###Creating Training data set

#Picking data that is not part of the validating dataset

set.seed(192)

Muridae\_Training<- Muridae\_all %>% filter (!Title %in% Muridae\_Validation) %>% group\_by(Code) %>% sample\_n(1351)

#ensuring we have 1351 samples of each for our training dataset

table(Muridae\_Training$Code)

#checking column numbers and names to determine which columns will be used in the next part of the code

names(Muridae\_Training)

#Building a gene classifier to separate COI and cytB genes using A, T, G proportions, followed by more complex k-mers if needed

random\_classifier <- randomForest::randomForest(x = Muridae\_Training[, 9:11], y = as.factor(Muridae\_Training$Code), ntree = 50, importance = TRUE)

#looking to see if gene classifier worked

random\_classifier

#increasing the number of trees to 500 to see if it more accurate

random\_classifier\_ <- randomForest::randomForest(x = Muridae\_Training[, 9:11], y = as.factor(Muridae\_Training$Code), ntree = 500, importance = TRUE)

random\_classifier\_

#It wasn't able to classify all of them accurately using ATCG proportions, error rate of 0.07% so k-mers of length 2 will be used to see if it classifies it better

random\_classifier\_1 <- randomForest::randomForest(x = Muridae\_Training[, 12:27], y = as.factor(Muridae\_Training$Code), ntree = 50, importance = TRUE)

random\_classifier\_1

#The error rate remains at 0.07% so increasing the number of trees to 500

random\_classifier\_1\_1 <- randomForest::randomForest(x = Muridae\_Training[, 12:27], y = as.factor(Muridae\_Training$Code), ntree = 500, importance = TRUE)

random\_classifier\_1\_1

#Error rate increases to 0.11% at 500 so k-mers of length 3 will be used to see if it classifies it better

random\_classifier\_2 <- randomForest::randomForest(x = Muridae\_Training[, 28:91], y = as.factor(Muridae\_Training$Code), ntree = 50, importance = TRUE)

random\_classifier\_2

#this classified them perfectly

#testing on unseen data

predictValidation <- predict(random\_classifier\_2, Muridae\_Validation[, c(70, 28:91)])

#These are the predictions that were made

predictValidation

class(predictValidation)

length(predictValidation)

table(predictValidation)

#Viewing confusion matrix to determine true positives, negatives as well as false positives and negatives in the dataset

random\_classifier\_2$confusion

#relative importance of each feature

random\_classifier\_2$importance

#This shows the fraction of votes each gene got

random\_classifier\_2$votes

#Trying to see if classifier works on unseen data

table(observed = Muridae\_Validation$Code, predicted = predictValidation)

#Our classifer works well with unseen data as well

##Adding error values to be used later

random\_error<-0.0007

random\_error\_1<-0.0007

random\_error\_2<-0

##Next we will try a different classifier known as Naive Bayes classifer to see if COI and cytB can be distinguished. Creating classifier with ATG proportions as training

Bayes\_Classifier<-naive\_bayes(x = Muridae\_Training[, 9:11], y = as.factor(Muridae\_Training$Code))

Bayes\_Classifier

#Predicting

Bayes\_predict<- predict(Bayes\_Classifier, Muridae\_Training, type='prob')

(head(cbind(Bayes\_predict, Muridae\_Training)))

### the first few rows are COI sequences and and it is able to predict it with a 80-90% probability

#Confusion matrix-train data

Bayes\_train<-predict(Bayes\_Classifier, Muridae\_Training)

(tab1<-table(Bayes\_train, Muridae\_Training$Code))

#there were 1333 correct predictions for COI and 1351 for cytB

#Confusion matrix-test data

Bayes\_test<-predict(Bayes\_Classifier, Muridae\_Validation)

(tab2<-table(Bayes\_test, Muridae\_Validation$Code))

#there were 455 correct COI and 450 correct cytB genes classified

#the error rate is

Bayes\_error<-1-sum(diag(tab1))/sum(tab1)

#Trying to see if the classifier can be made more accurate by incorporating k-mers of length 2

Bayes\_Classifier\_1<-naive\_bayes(x = Muridae\_Training[, 12:27], y = as.factor(Muridae\_Training$Code))

Bayes\_Classifier\_1

#Predicting

Bayes\_predict\_1<- predict(Bayes\_Classifier\_1, Muridae\_Training, type='prob')

head(cbind(Bayes\_predict\_1, Muridae\_Training))

### the first few rows are COI sequences and and it is able to predict it with almost 100% probability

#Confusion matrix-train data

Bayes\_train\_1<-predict(Bayes\_Classifier\_1, Muridae\_Training)

(tab\_1\_1<-table(Bayes\_train\_1, Muridae\_Training$Code))

#there were 1348 correct predictions for COI and 1351 for cytB

#Confusion matrix-test data

Bayes\_test\_1<-predict(Bayes\_Classifier\_1, Muridae\_Validation)

(tab2\_1<-table(Bayes\_test\_1, Muridae\_Validation$Code))

#there were 449 correct COI and 450 correct cytB genes classified

#the error rate for training dataset is

Bayes\_error\_1<-1-sum(diag(tab\_1\_1))/sum(tab\_1\_1)

#Trying to see if the classifier can be made more accurate by incorporating k-mers of length 3

Bayes\_Classifier\_2<-naive\_bayes(x = Muridae\_Training[, 28:91], y = as.factor(Muridae\_Training$Code))

Bayes\_Classifier\_2

#Predicting

Bayes\_predict\_2<- predict(Bayes\_Classifier\_2, Muridae\_Training, type='prob')

head(cbind(Bayes\_predict\_2, Muridae\_Training))

### the first few rows are COI sequences and and it is able to predict it with almost 100% probability

#Confusion matrix-train data

Bayes\_train\_2<-predict(Bayes\_Classifier\_2, Muridae\_Training)

(tab\_1\_2<-table(Bayes\_train\_2, Muridae\_Training$Code))

#there were 1351 correct predictions for COI and 1351 for cytB

#Confusion matrix-test data

Bayes\_test\_2<-predict(Bayes\_Classifier\_2, Muridae\_Validation)

(tab2\_2<-table(Bayes\_test\_2, Muridae\_Validation$Code))

#there were 450 correct COI and 450 correct cytB genes classified

#the error rate for training dataset is

(Bayes\_error\_2<-1-sum(diag(tab\_1\_2))/sum(tab\_1\_2))

#This led to perfect classification as that seen in random forest classifier

##plotting accuracy rates of each classifier from random forest as well as Naive Bayes. Random forest classifiers with increased number of trees(500) were omitted as they had similar error rates or higher to those with 50 trees

error\_rates<-rbind(Bayes\_error, Bayes\_error\_1, Bayes\_error\_2, random\_error, random\_error\_1, random\_error\_2)

#convert to dataframe

error\_rates<-as.data.frame(error\_rates)

#convert classfier name so it is considered a column and not individual rows

error\_rates <- error\_rates %>%

rownames\_to\_column('classifier\_name')

#Viewing to see if it worked

error\_rates

#replacing column name to error in data frame

colnames(error\_rates)<-c("classifier\_names", "error")

#Viewing to see if it worked

error\_rates

#Creating a barplot for accuracy of each classifier created

bar\_plot<-ggplot(error\_rates, aes(x=classifier\_names, y=error, title="Error frequency in classfiers")) +

geom\_bar(stat="identity")+theme\_minimal()

#Creating a barplot for accuracy of each classifier created

theme\_update(plot.title = element\_text(hjust = 0.5))

bar\_plot<-ggplot(error\_rates, aes(x=classifier\_names, y=error)) + ggtitle("Error frequency in classfiers") + theme(plot.title = element\_text(hjust = 0.5)) + geom\_bar(stat="identity") +theme\_minimal()

bar\_plot

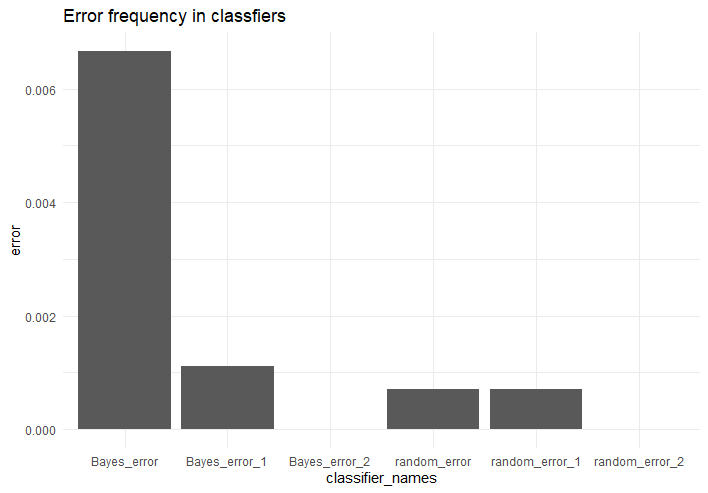


Figure 3: Various classifiers on x axis created in main analysis of our code with error frequencies on y axis

###Results and Discussion:

#Our study was intended to investigate whether Random Forest or Naive Bayes classifier would be better at predicting COI and cytB within the Muridae family. We predicted that Random forest would be able to classify COI and cytB genes far more efficiently and accurately based on our literature search. In one example from Lemons et al., 2020, the random forest classifier was outperformed Naive Bayes and had an accuracy of 97.82% when diagnosing breast cancer. From our results we can see that using both classifiers, Random Forest and Naive Bayes were not able to accurately classify all COI and cytB sequences from AGT proportions alone within the Muridae family. In Random Forest the first classifier generated had an error rate of 0.07, and although negligible we wanted to see if it could be classified more accurately. We decided to increase the number of trees to 500, however, the error rate remained at 0.07%. Next k-mers of length 2 were used for our training dataset however the error rate remained at 0.07%, increasing the number of trees to 500 did not help with accuracy, so we used k-mers of length 3. Using k-mer length of 3 we were able to classify both our training dataset as well as validation dataset accurately with 0 error rate. The error rate of each of the three classifiers can be seen in figure 3 labelled as “random\_error, random\_error\_1 and random\_error\_2. This is analysis was followed up by Naïve Bayes prediction algorithm. Naive Bayes followed a similar pattern, where AGT proportions and k-mer of length 2 were not sufficient to accurately differentiate all COI and cytB, however using k-mer of length 3 was able to differentiate both our training and validation dataset accurately. The error rate of each of these classifiers can also be seen in figure three labelled as Bayes\_error, Bayes\_error\_1 and Bayes\_error\_2. Despite what is found in the literature, our results show that Random Forest and Naive Bayes performed somewhat equally as both required additional information i.e k-mer length of 3 to accurately classify all genes. However, it is noteworthy that random forest classifiers had a lower error rate for AGT and k-mer length of 2 for training sets when compared with Bayes.

#In terms of biases, our dataset had a large amount of data from the species from Mus spertus, 2nd largest from Apodemus draco and 3rd largest from Apodemus sylvaticus. This may impact our classifier and it may be more accurate for those species however, the degree to which it is impacted was not tested here. In the future it will be beneficial to explore statistical tests to see if the observed differences are significant between the two models. In addition, testing various other classifiers could be beneficial in determining which one works best for gene classification of COI and cytB this can include the regression model. We could also test this classifier on other taxonomic species to see if there is common sequence variation between cytB and COI that is present in other species. In conclusion our results show that Random forest and Naive Bayes perform quite equally when classifying COI and cytB genes within the Muridae family.

#References:

#Yiu, T. (2021, September 29). Understanding random forest. Medium. Retrieved October 27, 2022, from https://towardsdatascience.com/understanding-random-forest-58381e0602d2

#Latief, M. A., Siswantining, T., Bustamam, A., &amp; Sarwinda, D. (2019). A comparative performance evaluation of Random Forest feature selection on classification of hepatocellular carcinoma gene expression data. 2019 3rd International Conference on Informatics and Computational Sciences (ICICoS). https://doi.org/10.1109/icicos48119.2019.8982435

#Gandhi, R. (2018, May 17). Naive Bayes classifier. Medium. Retrieved October 27, 2022, from https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

#Introduction to the rodentia. Rodentia. (n.d.). Retrieved October 27, 2022, from https://ucmp.berkeley.edu/mammal/rodentia/rodentia.html

#Aghová, T., Kimura, Y., Bryja, J., Dobigny, G., Granjon, L., &amp; Kergoat, G. J. (2018). Fossils know it best: Using a new set of fossil calibrations to improve the temporal phylogenetic framework of Murid rodents (rodentia: Muridae). Molecular Phylogenetics and Evolution, 128, 98-111. https://doi.org/10.1016/j.ympev.2018.07.017

#Kartavtsev, Y. P. (2011). Divergence at cyt-b and co-1 mtdna genes on different taxonomic levels and genetics of speciation in Animals. Mitochondrial DNA, 22(3), 55-65. https://doi.org/10.3109/19401736.2011.588215

#Lemons, K. (2020). A comparison between naïve Bayes and Random Forest to predict breast cancer. International Journal of Undergraduate Research and Creative Activities, 12(1), 1. https://doi.org/10.7710/2168-0620.0287

#westlandindia. (2018, February 14). Naive Bayes classification with R | example with steps. YouTube. Retrieved October 27, 2022, from <https://www.youtube.com/watch?v=RLjSQdcg8AM>

#Column not being recognised as variable in R. Stack Overflow. (1967, December 1). Retrieved October 28, 2022, from https://stackoverflow.com/questions/63714762/column-not-being-recognised-as-variable-in-r