Alisha Soares
Project 2

######Supervised Machine Learning of organisms in the Clupeiformes Order using Random Forest and Caret----

######Introduction----

Overfishing is a global concern. It can alter the fish populations and affect their maturity and reproducibility. Marine ecosystems may also be affected by the excessive overfishing of a fish species, possibly endangering other marine life including susceptible species such as sea turtles and coral reefs.

Sustainable fishing is vital in reducing the impact on the environment and securing future fish populations. One of the methodologies is the accurate labeling of seafood products. The information in the labels would include details such as where the fish were harvested, and how they were harvested and processed (Alfnes, Chen, and Rickertsen. 2017). One of the most important labels in recent times is the species label. The One Name, One Fish Report (2015) addressed the issue of species fraud, whereby endangered species of fish may be sold as another member of that family through general labelling.

Sardines, herring, anchovies and other ray-finned fish belong to the Clupeiformes order. Most of the species within this family are valuable for food production and manufacturing of "fish oil and fish meal" (BOLD Systems). Studies such as Kochzias et al (2010), researched the feasibility of using cytochrome b (CytB) gene and cytochrome oxidase subunit 1 (COI) gene found in the mitochondria as markers to identify fish species.

This project aims to build a classifier to identify sequence records as belonging to either CytB gene or COI gene. First, the "Random Forest" package will be used to build a classifier, then another classifier will be created using the "Caret" package and the "KNN" model, with the success of each classifier evaluated at the end. The goal is to build successful classifiers using supervised machine learning and to better understand how to use these methods in R.

######Packages and Functions----

Install packages and/or load libraries

#install.packages("rentrez")

library(rentrez)

#install.packages("seqinr")

library(seqinr)

#install.packages("BiocManager")

#library(BiocManager)

#BiocManager::install("Biostrings")

library(Biostrings)

#install.packages("tidyverse")

```
library(tidyverse)

library(dplyr)

library(tidyr)

#install.packages("randomForest")

library(randomForest)

#install.packages("caret")

library(caret)


## Load functions.

source('ez_rentrez.R.txt') #written by Jason Moggridge


######Obtaining Data from NCBI----

### There are 3000 records in the dataset for members of the Clupeiformes order with CytB and COI genes. Different methods were tried for data acquisition, but errors appeared when running the whole RScript. In the end, the function written by Jason Moggridge was selected as the best choice for obtaining data without errors. This methodology works as long as the internet connection is stable.


## Set parameter variables:

# Personal NCBI apikey

apikey <- '31c99e909c129df58c7eeceb510ce4d56009'


# Set search expression variables

searchexp1 <- '(Clupeiformes[ORGN] AND CytB[Gene]) NOT (genome[TITL])'

searchexp2 <- '(Clupeiformes[ORGN] AND COI[Gene]) NOT (genome[TITL])'


# Get search metadata list, and summary and fasta as a data frame for CytB gene

CytB.search <- get_ncbi_ids(searchexp1)

CytB.summary.df <- get_ESummary_df(searchexp1, apikey)

CytB.fasta.df <- get_Efasta(searchexp1, apikey)
```

```
# Get search metadata list, and summary and fasta as a data frame for COI gene

COI.search <- get_ncbi_ids(searchexp2)

COI.summary.df <- get_ESummary_df(searchexp2, apikey)

COI.fasta.df <- get_Efasta(searchexp2, apikey)


#######Organizing the data into manageable datasets----

## For CytB gene:

# Append CytB.fasta data frame with CytB.summary data frame. This will provide a dataset with features of the records as well as organism names that may be used to verify that data belongs to fish species within the Clupeiformes order.

dfCytB <- cbind(CytB.summary.df, as.data.frame(CytB.fasta.df$seq))


# Rename CytB.fasta.df$seq as sequence.

names(dfCytB)[34] <- "sequence"


# Convert dfCytB to tibble format so that Tidyverse functions may be utilized.

as_tibble(dfCytB)


#Split title column to remove extra information.

dfCytB <- separate(dfCytB,
                   col = title,
                   sep = ",",
                   into = c("title", "excess"),
                   extra = "drop")


# Add a new column for the CytB gene.

dfCytB$gene <- "CytB"


# Create subset of the CytB dataset to work with downstream.
```

```
dfCytBsub <- dfCytB[, c("id", "title", "organism", "sequence", "gene")]


## For COI gene:

# Append COI.fasta data frame with COI.summary data frame. This will provide a dataset with features of the records as well as organism names that may be used to verify that data belongs to fish species within the Clupeiformes order.

dfCOI <- cbind(COI.summary.df, as.data.frame(COI.fasta.df$seq))


# Rename COI.fasta.df$seq as sequence.

names(dfCOI)[34] <- "sequence"


# Convert dfCOI to tibble so that Tidyverse functions may be utilized.

as_tibble(dfCOI)


# Split title column to remove extra information.

dfCOI <- separate(dfCOI,
                  col = title,
                  sep = ",",
                  into = c("title", "excess"),
                  extra = "drop")


# Add a new column for the COI gene.

dfCOI$gene <- "COI"


# Create subset of the COI dataset to work with downstream.

dfCOIsub <- dfCOI[, c("id", "title", "organism", "sequence", "gene")]


# Check if there are any NA values in the sequence column for CytB and COI genes. Here we see that there are no NA values for either dataset.

sum(is.na(dfCytBsub$sequence))
```

```
sum(is.na(dfCOIsub$sequence))
```

Clean up sequences for CytB and COI genes by creating a new variable, sequence2. This will then be edited in place, to remove any Ns or gaps at the beginning and ends of the sequences. Also remove any gaps found within each sequence as the goal is to use k-mer frequencies to help create classifiers, instead of aligning sequences. Filter out any sequences with more than 5% of missing data, after trimming for Ns and gaps in.

```
# For CytB:
dfCytBseq <- dfCytBsub %>%
  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)))

# For COI:
dfCOIseq <- dfCOIsub %>%
  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)))

# Check histogram and summary of sequence lengths for CytB gene. See Figure 1a.
hist(nchar(dfCytBsub$sequence[dfCytBseq$gene == "CytB"]),xlab = "Sequence Length",ylab =
"Frequency",main = "Frequency Histogram of CytB Sequence Lengths", col = "lightblue")

summary(nchar(dfCytBsub$sequence[dfCytB$gene == "CytB"]))

# Check histogram and summary of sequence lengths for COI gene. See Figure 1b.
hist(nchar(dfCOIsub$sequence[dfCOIseq$gene == "COI"]), xlab = "Sequence Length", ylab =
"Frequency", main = "Frequency Histogram of COI Sequence Lengths", col = "lightblue")
```

```
summary(nchar(dfCOIsub$sequence[dfCOI$gene == "COI"]))


# Sort species (organism column) with highest number of records for each gene. The European anchovy (Engraulis encrasicolus) has 1105 records for CytB gene and 186 records for the COI gene.

sort(table(dfCytBseq$organism), decreasing = TRUE) [1:5]


sort(table(dfCOIseq$organism), decreasing = TRUE) [1:5]


## Create histograms to visualize the species with the highest records for each gene.See Figure 2.

barchart(sort(table(dfCytBseq$organism), decreasing = TRUE) [1:5], xlab = "Number of records", ylab = "Species names", main = "Species with highest number of records for CytB gene", col = "lightblue")


barchart(sort(table(dfCOIseq$organism), decreasing = TRUE) [1:5], xlab = "Number of records", ylab = "Species names", main = "Species with highest number of records for COI gene", col = "lightblue")


# Combine the CytB sequence and COI sequence datasets for further analysis.

dfSeq <- rbind(dfCytBseq, dfCOIseq)


## Assign variables for the first and third quartile sequence lengths. Want to raise third quartile level to ensure that all needed sequences are kept.

q1 <- quantile(nchar(dfSeq$sequence2[dfSeq$gene == "CytB"]), probs = 0.15, na.rm = TRUE)

q1


q3 <- quantile(nchar(dfSeq$sequence2[dfSeq$gene == "CytB"]), probs = 0.85, na.rm = TRUE)

q3


# Filter by constraining length variability using q1 and q3 variables set above.

dfSeq <- dfSeq %>%

  filter(((str_count(sequence2) >= q1 & str_count(sequence2) <= q3 & gene == "CytB" | gene == "COI")))
```

```
# Perform checks to ensure that everything is as expected.
dim(dfSeq)
unique(dfSeq$gene)
table(dfSeq$gene)
sum(is.na(dfSeq$sequence2))
summary(str_count(dfSeq$sequence2[dfSeq$gene == "CytB"]))
summary(str_count(dfSeq$sequence2[dfSeq$gene == "COI"]))

# Convert nucleotides to DNAStringSet.
dfSeq <- as.data.frame(dfSeq)
dfSeq$sequence2 <- DNAStringSet(dfSeq$sequence2)

# Calculate nucleotide frequencies and append them to the dataframe using cbind.
dfSeq <- cbind(dfSeq, as.data.frame(letterFrequency(dfSeq$sequence2, letters = c("A", "T", "G", "C"))))

# Add A, T, and G proportions in relation to the total number of nucleotides
dfSeq$Aprop <- (dfSeq$A) / (dfSeq$A + dfSeq$T + dfSeq$C + dfSeq$G)

dfSeq$Tprop <- (dfSeq$T) / (dfSeq$A + dfSeq$T + dfSeq$C + dfSeq$G)

dfSeq$Gprop <- (dfSeq$G) / (dfSeq$A + dfSeq$T + dfSeq$C + dfSeq$G)

## Add in dinucleotide and trinucleotide frequencies to account for variability in sequence lengths.
# Add dinucleotide frequency in proportions
dfSeq <- cbind(dfSeq, as.data.frame(dinucleotideFrequency(dfSeq$sequence2, as.prob = TRUE)))

# Add in trinucleotide frequencies in proportions.
dfSeq <- cbind(dfSeq, as.data.frame(trinucleotideFrequency(dfSeq$sequence2, as.prob = TRUE)))
```

```
# Remove datasets and information that is no longer needed

rm(COI.fasta.df, COI.search, COI.summary.df, dfCOI, dfCOIsub, dfCOIseq, CytB.fasta.df, CytB.search, CytB.summary.df, dfCytB, dfCytBsub, dfCytBseq)


#######Training the Classification Model with Random Forest----

### Determine if the CytB and COI genes can be accurately classified using simple sequence features.

## As seen above, European anchovies have both genes represented in sufficient numbers. Therefore, use the European anchovy species to test our classification model.

# Make a dataset with only European anchovy data

dfEuroAnchovy <- dfSeq[dfSeq$organism == "European anchovy", ]

unique(dfEuroAnchovy$organism)


## Convert the data from strings format to character data so that tidyverse functions may be used on the data.

dfEuroAnchovy$sequence2 <- as.character(dfEuroAnchovy$sequence2)


# Check the counts by gene.

table(dfEuroAnchovy$gene)


## The maximum sample size is 186 due to the COI gene. Sample 45 individuals (20% of COI dataset) with each gene to be used in the validation dataset, setting them aside completely during model training. Use set.seed to randomize the data.

set.seed(300)


# Sample 45 rows from each gene for validation dataset

dfValidation <- dfEuroAnchovy %>%

  group_by(gene) %>%

  sample_n(45)
```

```
# Check object

table(dfValidation$gene)
```

Next, create a training dataset that will not overlap with the records in the validation set. After removing the ids selected for the validation set, pick 135 individuals (80% of COI dataset) with each gene for the training set. Use set.seed to randomize the data.

```
set.seed (65)

# Sample 135 individuals of each gene not already in the validation set

dfTraining <- dfEuroAnchovy %>%
  filter(!id %in% dfValidation$id) %>%
  group_by(gene) %>%
  sample_n(135)

# Check object

table(dfTraining$gene)

# Check the variable names and their column numbers

names(dfTraining)
```

The next step is to build the actual classifier to separate out CytB and COI genes. Here, use the A, T and G proportion columns (11 to 13) as predictors. The response variable is the gene. The goal is to be able to predict which gene a European anchovy sequence belongs to based on simple sequence features alone.

```
# Build the first classifier. Based on A, T, G proportions

ATG.Classifier <- randomForest(x = dfTraining[, 11:13], y = as.factor(dfTraining$gene), ntree = 1000, importance = TRUE)

ATG.Classifier

# Check the importance and OOB times
```

```
ATG.Classifier$importance

ATG.Classifier$oob.times


## Try the classifier again but with just A and T proportions as predictors. Keep the response variable as gene.

AT.Classifier <- randomForest(x = dfTraining[, 11:12], y = as.factor(dfTraining$gene), ntree = 1000, importance = TRUE)

AT.Classifier


# Check the importance and OOB times

AT.Classifier$importance

AT.Classifier$oob.times


## There is a difference in the error rate when using all 3 nucleotide proportions instead of just 2. For the next step, use the A, T and G proportion classifier to predict the marker for the unseen data based on the nucleotide proportions.

ATG.Validation <- predict(ATG.Classifier, dfValidation[, c(5, 11:13)])


# Check the data format

ATG.Validation

class(ATG.Validation)

length(ATG.Validation)


# Create a confusion matrix for ATG.Validation.

table(observed = dfValidation$gene, predicted = ATG.Validation)


# Draw the confusion matrix for ATG.validation. See Figure 3a.

fourfoldplot(table(observed = dfValidation$gene, predicted = ATG.Validation), main = "Confusion matrix for A, T and G proportions")
```

Though the classification has worked, test further classifiers using dinucleotide and trinucleotides and see what the results are.

Using dinucleotides (columns 12 to 29 in dfTraining) build a classifier to separate CytB and CoI genes

Build the dinucleotide classifier

```
dinuc.Classifier <- randomForest(x = dfTraining[, 14:29], y = as.factor(dfTraining$gene), ntree = 1000, importance = TRUE)

dinuc.Classifier
```

Success! The error rate is now 0%. With dinucleotide or k-mer patterns it becomes easier for the

```
dinuc.Validation <- predict(dinuc.Classifier, dfValidation[, c(5, 14:29)])
```

Check the data format

```
dinuc.Validation

class(dinuc.Validation)

length(dinuc.Validation)
```

Create a confusion matrix for dinucleotide validation

```
table(observed = dfValidation$gene, predicted = dinuc.Validation)
```

Draw the confusion matrix for dinucleotide frequencies.

```
fourfoldplot(table(observed = dfValidation$gene, predicted = dinuc.Validation), main = "Confusion matrix for dinucleotides")
```

Now using trinucleotides (columns 30 to 93 in dfTraining) build a classifier to separate CytB and CoI genes

Build the trinucleotide classifier

```
trinuc.Classifier <- randomForest(x = dfTraining[, 30:93], y = as.factor(dfTraining$gene), ntree = 1000, importance = TRUE)

trinuc.Classifier
```

Success! The error rate is now 0%. With dinucleotide or k-mer patterns it becomes easier for the classifier to correctly identify the genes.

trinuc.Validation <- predict(trinuc.Classifier, dfValidation[, c(5, 30:93)])

Check the data format

trinuc.Validation

class(trinuc.Validation)

length(trinuc.Validation)

Create a confusion matrix for trinucleotide validation.

table(observed = dfValidation$gene, predicted = trinuc.Validation)

Draw the confusion matrix for trinucleotide frequencies. Note: As the resulting matrix is the same for both dinucleotides and trinucleotides, this matrix has a main title for both. See Figure 3b.

fourfoldplot(table(observed = dfValidation$gene, predicted = dinuc.Validation), main = "Confusion matrix for dinucleotides and trinucleotides")

#######Training the Classification Model with Caret----

Use caret and knn (k- Nearest Neighbors) for supervised machine learning and classification of CytB and COI genes

Create a new dataset for the Caret package to use. This will take data from the European Anchovy dataset so that both classification methods use the same data.

dfCaret <- dfEuroAnchovy[, c(5,7:13)]

Check the summary of the caret dataset

summary(dfCaret)

Use the createDataPartition function from caret to create balanced training and test datasets. Unlike the sample_n function, this function automatically sets aside 80% of the records for the training set and 20% for the test set.

Create the training and test datasets for European Anchovies in the caret dataset

```
set.seed(200)
train.Rows <- createDataPartition(dfCaret$gene, p = 0.8, list = FALSE)
train.Caret <- dfCaret[train.Rows,]
test.Caret <- dfCaret[-train.Rows,]

# Check the dimessions of the training and testing datasets
dim(train.Caret); dim(test.Caret)
str(train.Caret) ; str(test.Caret)

# Need to convert genes from characters to factor variables in the train.Caret dataset
train.Caret[["gene"]] = factor(train.Caret[["gene"]])

# Use the knn model for categorization of the genes
knn.train <- trainControl(method = "repeatedcv", number = 10, repeats = 2)
set.seed(200)
knn.fit <- train(gene ~., data = train.Caret, method = "knn",
                 trControl = knn.train,
                 # Transform the data into binary format or 1s and 0s
                 preProcess = c("center", "scale"),
                 # tuneLength is for tuning the algorithm
                         tuneLength = 20)
knn.fit

# Plot knn test accuracy to see the best k values for the test.See Figure 4a.
plot(knn.fit, main = "Test Accuracy for KNN Model - European anchovies subset")

## The model trained with a k value of 5. Next, predict genes for the test set using the predict function
knn.predict <- predict(knn.fit, newdata = test.Caret)
knn.predict
```

```
# Convert the genes from characters to factor variables in the res.Caret dataset

test.Caret[["gene"]] = factor(test.Caret[["gene"]])


# Use a confusion matrix to see the accuracy of the model above. Here the model claims 98% accuracy.

cm <- confusionMatrix(data = knn.predict, reference = test.Caret$gene)


cm$table


# Draw the confusion matrix for the knn classifier model. See Figure 3c.

fourfoldplot(cm$table, main = "Confusion matrix for KNN classifier model - European anchovies subset")


# Remove unneeded objects.

rm(dfEuroAnchovy, dfTraining, dfValidation, dinuc.Classifier, trinuc.Classifier, AT.Classifier,
ATG.Classifier, dfCaret, test.Caret, train.Caret, train.Rows, knn.fit)



### Try and see if the knn classification model works for the entire dataset. Will it function at the same
level of accuracy as the knn classification for the European anchovies subset?


# Create a new dataset for the Caret package to use. This will take data from the Sequence dataset of
that has the data for all the records of the CytB and CoI genes..

dfSeq.caret <- dfSeq[, c(5,7:13)]


# Check the summary of the dataset

summary(dfSeq.caret)


## Use the createDataPartition function from caret to create balanced training and test datasets.

# Create the training and test datasets.

set.seed(700)
```

```
train.Rows2 <- createDataPartition(dfSeq.caret$gene, p = 0.8, list = FALSE)

train.Caret2 <- dfSeq.caret[train.Rows2,]

test.Caret2 <- dfSeq.caret[-train.Rows2,]


# Check the dimessions of the training and testing datasets

dim(train.Caret2); dim(test.Caret2)

str(train.Caret2) ; str(test.Caret2)


# Need to convert genes from characters to factor variables in the train.Caret2 dataset

train.Caret2[["gene"]] = factor(train.Caret2[["gene"]])


# Use the knn model for categorization of the genes

knn.train2 <- trainControl(method = "repeatedcv", number = 20, repeats = 4)

set.seed(700)

knn.fit2 <- train(gene ~., data = train.Caret2, method = "knn",

          trControl = knn.train2,

          # Transform the data into binary format or 1s and 0s

          preProcess = c("center", "scale"),

          # tuneLength is for tuning the algorithm

          tuneLength = 20)

knn.fit2


# Plot knn test accuracy to see the best k values for the test. See Figure 4b.

plot(knn.fit2, main = "Test Accuracy for KNN Model - Clupeiformes data")


## The model also trained with a k value of 5. Next, predict genes for the test set using the predict function

knn.predict2 <- predict(knn.fit2, newdata = test.Caret2)

knn.predict2
```

```
# Convert the genes from characters to factor variables in the res.Caret dataset

test.Caret2[["gene"]] = factor(test.Caret2[["gene"]])


# Use a confusion matrix to see the accuracy of the model above. Here the model claims 97% accuracy.

cm2 <- confusionMatrix(data = knn.predict2, reference = test.Caret2$gene)


cm2$table


# Draw the confusion matrix for the knn classifier model.See Figure 3d.

fourfoldplot(cm2$table, main = "Confusion matrix for KNN classifier model - Clupeiformes data")
```

######Discussion and Conclusion----

The initial data acquired from the NCBI database, for species in the Clupeiformes order, had over 6000 records for both the CytB and COI genes. There were no missing values in the sequence data obtained, though outliers were found in the datasets for each gene (Figure 1). Further evaluation of the data showed that the European anchovy species had sufficient records for each of the genes (Figure 2) and was therefore chosen as the training dataset of our classifiers.

The project proved successful as both “Random Forest” and “Caret” packages were used to create good classifiers for the CytB and COI genes. When evaluating the performance of using A and T nucleotide proportions in the Random Forest classifier, the error rate was 3.33%. The error rate then decreased to 2.59% when G nucleotide proportions were added to the classifier. This suggests that more nucleotide information and proportions increase the efficiency of the Random Forest classifier. Despite similarity between the mitochondrial CytB and COI genes, the classifier was able to differentiate and identify each successfully. When k-mer frequencies of 2 and 3 were considered, the error rate of the classifier decreased to zero percent (Figure 3(a) and (b)).

This was further demonstrated with the “K- Nucleotide Neighbor” (KNN) classification method using the “Caret” package. Training of the classification model suggested an optimal k-mer length of 5 for both the European anchovy species subset, as well as the original Clupeiformes dataset (Figure 4). Classification was 98.8% accurate in the European anchovy species subset, decreasing to 97.2% accuracy in the Clupeiformes dataset with relatively few mislabeled sequences (Figure 3(c) and (d)). KNN and the Caret package were easy to use and apply to datasets of various sizes. However, as KNN stored a large amount of training data for the Clupeiformes dataset, this could be an issue in projects with even larger datasets.

The study by Kochzius et. al (2010), used CytB and COI genes in DNA barcoding methods to identify fish species, while another study used CytB gene to differentiate between different species of anchovies (Santaclara, Cabado and Vieites. 2006). Therefore, this project can be expanded to research whether CytB and COI genes could be used to build classification models to predict the species of the Clupeiformes order accurately. It would then be possible to use CytB and COI genes to create phylogenies in R, showing geographical and evolutionary diversification of species within the Clupeiformes order. This possible future project will improve understanding machine learning, sequence alignment and phylogenetics in R.

######Acknowledgments----

I would like to acknowledge the guidance and support provided during the completion of this project. Dr. Sally Adamowicz, guided me during office hours in the selection of my genus of interest. When issues arose, she advised me to focus on the big picture instead of getting overwhelmed. Jacqueline May's tutorial session and code were helpful in resolving issues with consistent data acquisition. The function and vignette written by Jason Moggridge was also very helpful in obtaining data that would load without errors occurring when the RScipt was run.

######References----

1. Alfnes, F., Chen, Xi., and Rickertsen, K. (2017). Labeling farmed seafood: A review. Aquaculture Economics & Management. 22. 1-26. 10.1080/13657305.2017.1356398.

2. Lowell, B., Mustain, P., Ortenzi, K., and Warner, K., Ph.D. (2015, July). One Name, One Fish: Why Seafood Names Matter. Retrieved October 20, 2020, https://usa.oceana.org/sites/default/files/onenameonefishreport.pdf

3. Clupeidae: Taxonomy Browser. (n.d.). Retrieved October 19, 2020, from http://www.boldsystems.org/index.php/Taxbrowser_Taxonpage?taxid=1252

4. Kochzius M, Seidel C, Antoniou A, et al. (2010). Identifying Fishes through DNA Barcodes and Microarrays. PLoS One. 5(9):e12620. Published 2010 Sep 7. doi:10.1371/journal.pone.0012620

5. Santaclara, F., Cabado, A., and Vieites, J. (2006). Development of a method for genetic identification of four species of anchovies: E. encrasicolus, E. anchoita, E. ringens and E. japonicus. European Food Research and Technology. 223. 609-614. 10.1007/s00217-005-0241-5.

6. Moggridge, J. (2020, October 20). Ez_rentrez. Retrieved October 26, 2020, https://github.com/jmoggridge/ez_rentrez

7. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

8. https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/fourfoldplot

9. https://www.analyticsvidhya.com/blog/2016/12/practical-guide-to-implement-machine-learning-with-caret-package-in-r-with-practice-problem/

10. https://rpubs.com/njvijay/16444

Figures

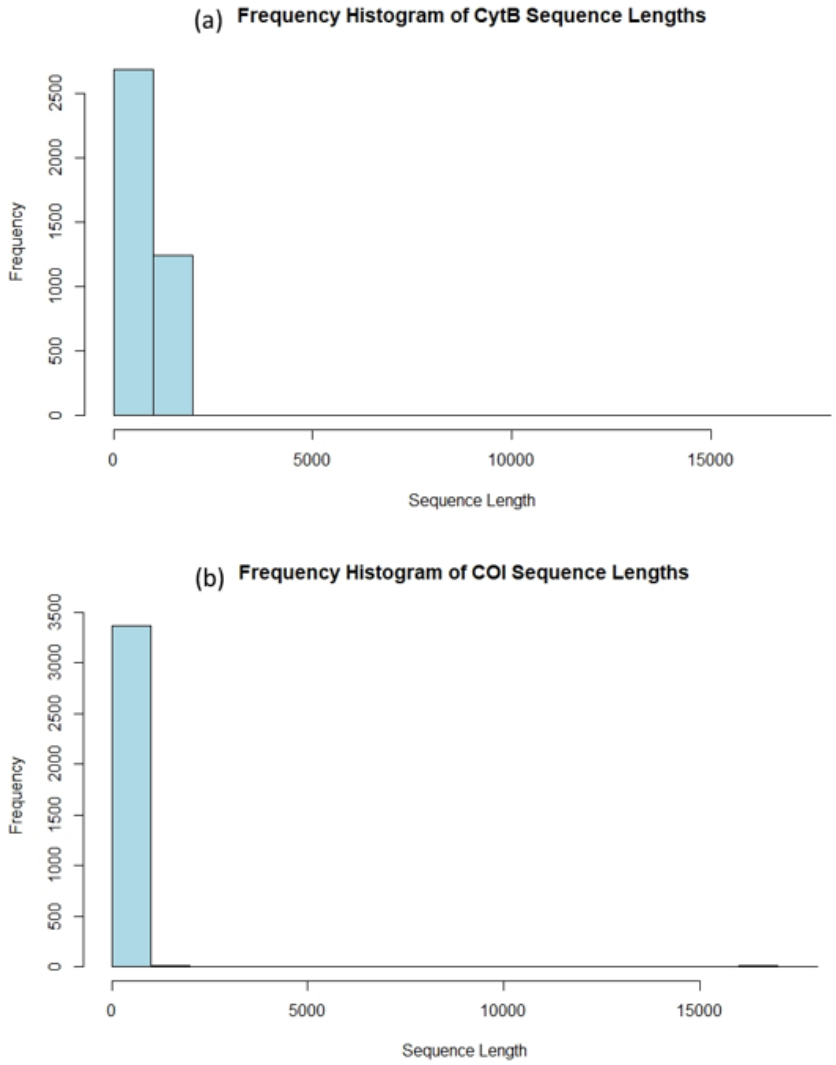

Figure 1: *Shows the frequency histograms of the sequence lengths for the CytB (a) and COI (b) genes before filtering the data for sequences within a certain length. The summary of the sequence lengths had shown that both genes had outlier sequences of 17049 base pairs.*

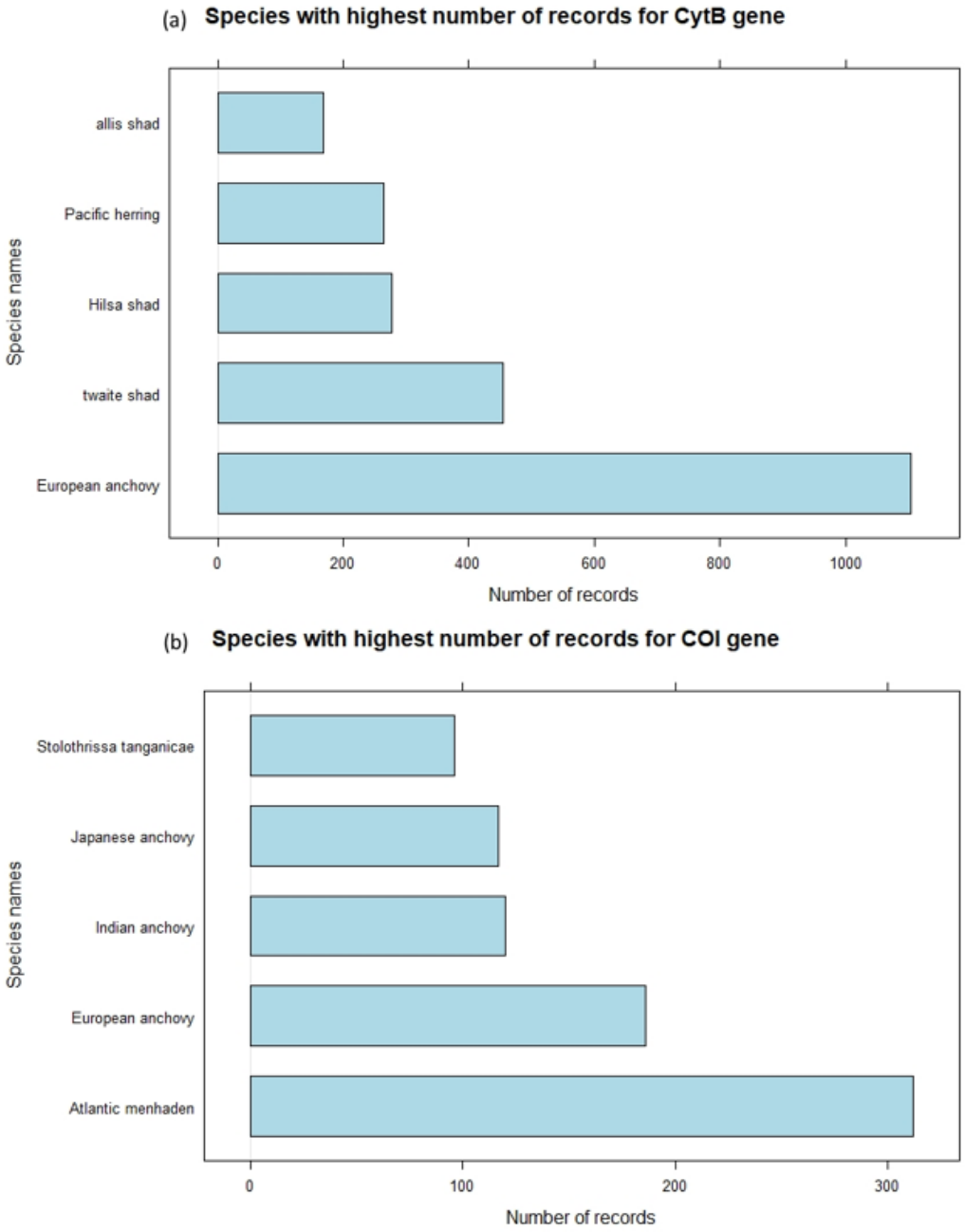

Figure 2: *The bar plots show the five species (by their common names) with the highest number of records for each gene. The Euopean anchovy (Engraulis encrasicolus) is present in both categories. The European anchovy species has 1105 records for CytB gene(a) and 186 records for COI gene(b). This species will be used in our classification models.*

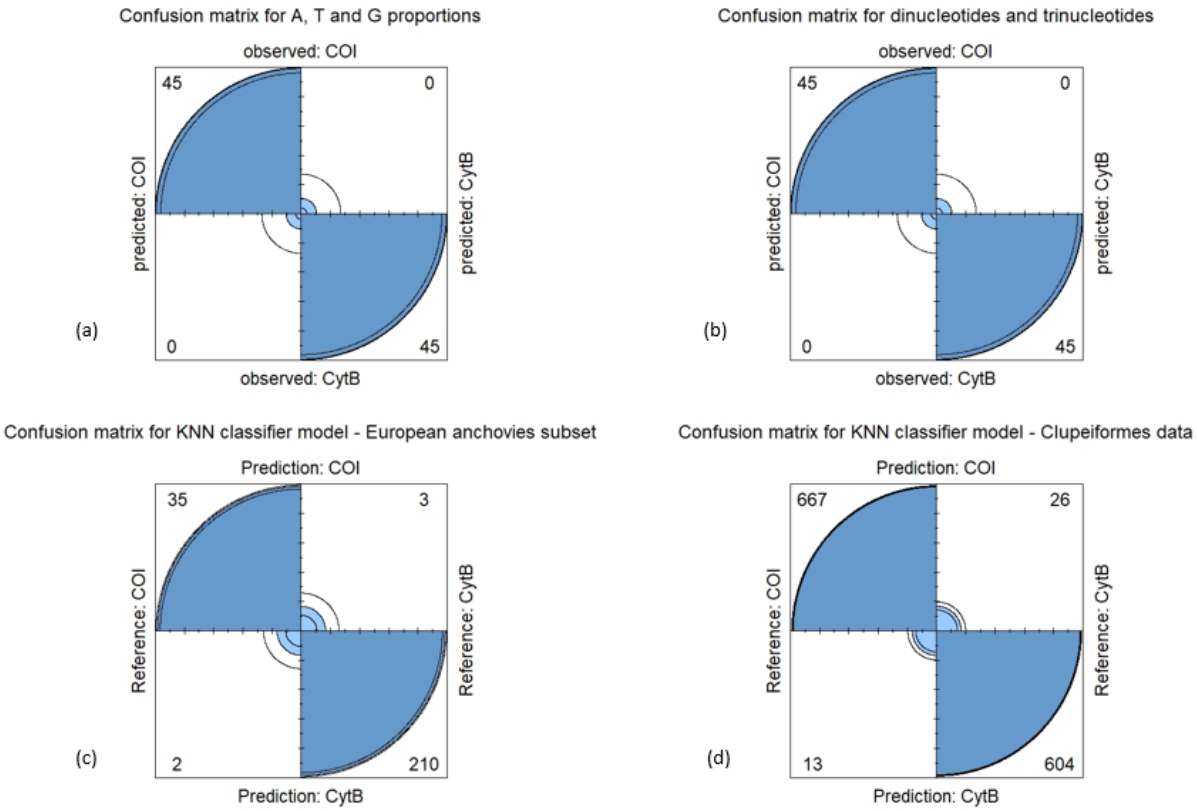

Figure 3: *The confusion matrices for each of the classification models and tests. Matrices (a) and (b) belong to the Random Forest classifier, show that there were no false negative and no false positives predicted when using single nucleotide proportions (a), or dinucleotide and trinucleotide classifiers (b). Matrices (c) and (d) belong to the KNN classifier model of the Caret package. The model was first used to classify CytB and COI genes in the European anchovy subset (c). This shows few false negative and false positive results with an overall test accuracy of 98.8%. The KNN model was then applied to the entire Clupeiformes data set (d). There are still relatively few false negative and false positive results, with a test accuracy of 97.2%.*

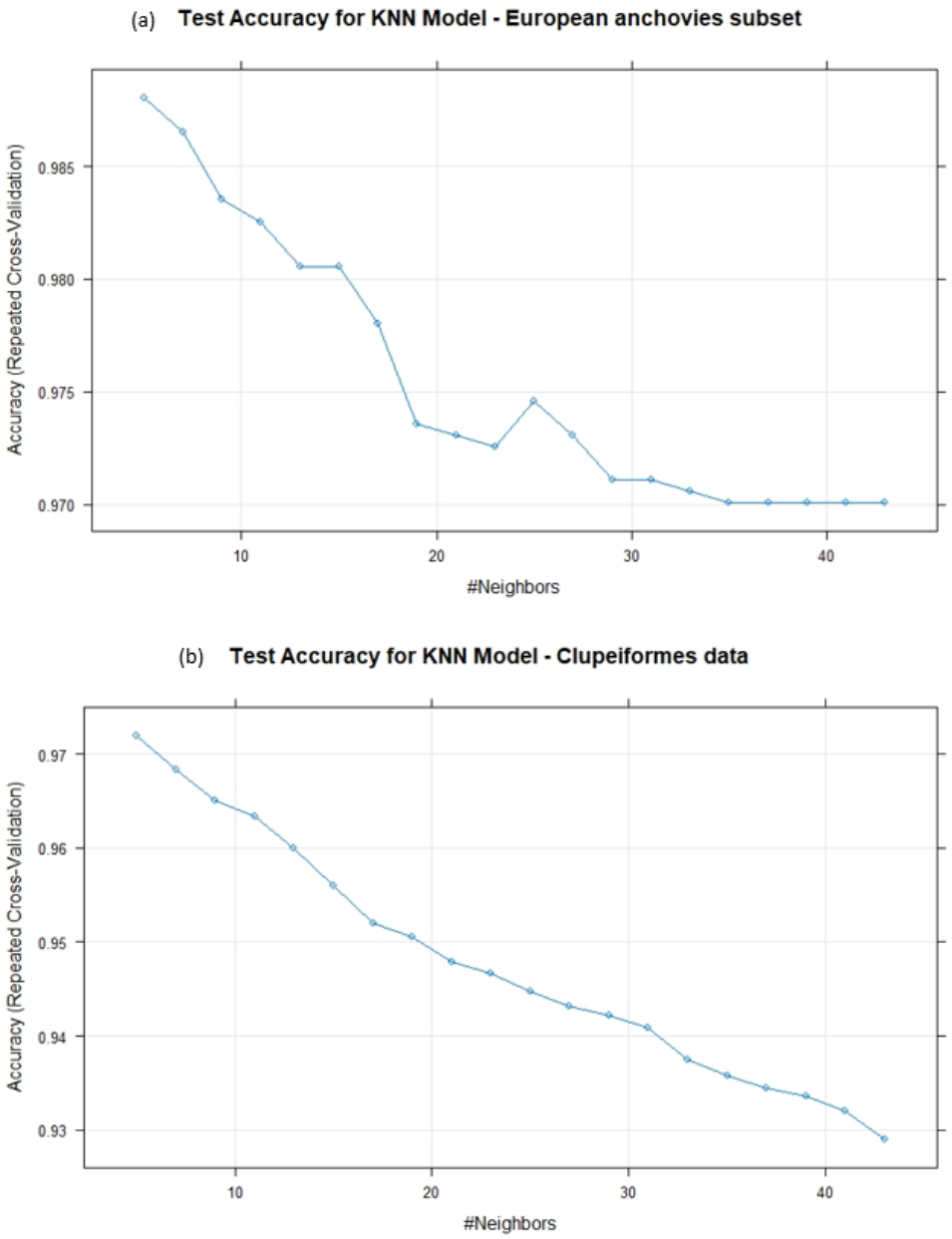

Figure 4: *The Test Accuracy plot for the K-Nearest Neighbor (KNN) Model depicts the optimal number of neighbors (k) for maximum test accuracy. The KNN model for European anchovies (a) has a 98.8% accuracy when k = 5. The KNN model for the Clupeiformes order (b) has a 97.2% accuracy when k = 5.*