Introduction

With the advent of next generation sequencing, there has been an abundance of sequence data uploaded to online databases. Using sequence databases, machine learning models have already made interesting findings such as predicting antibiotic resistance genes in collected environmental DNA (Arango-Argoty *et al.*, 2018) (Mahe & Tournoud 2018). There is a lot of potential for this to predict ecological changes in any sampled environment. As a result, knowing which type of model that can perform gene classification is of interest. The following models were evaluated: Random Forest, Support Vector Machine, XGBoost, KNearestNeighbor, Naïve-Bayes. They were selected either from previous familiarity or from referencing the models used in the taxonomy classification paper published by Nugent & Adamowicz (2020). The 18S rRNA and 16S mitochondrial rRNA of the Squamata taxon were chosen due to their homology.

Results

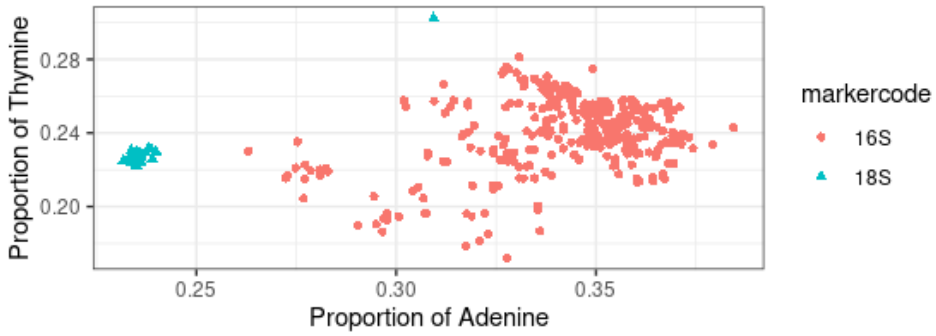

Figure 1.1 Adenine and Thymine Proportion Clustering of 16S and 18S Squamata rRNA

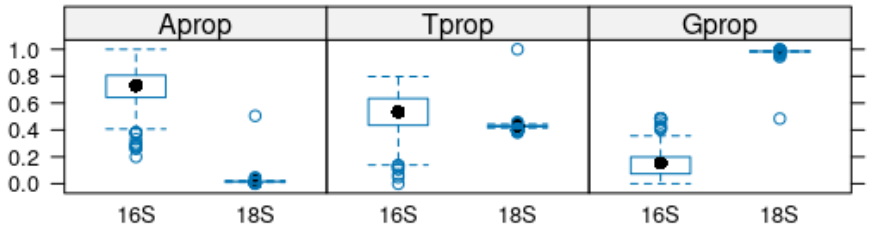

Figure 1.2 Box and Whisker Plot of Predictor Proportion Variables

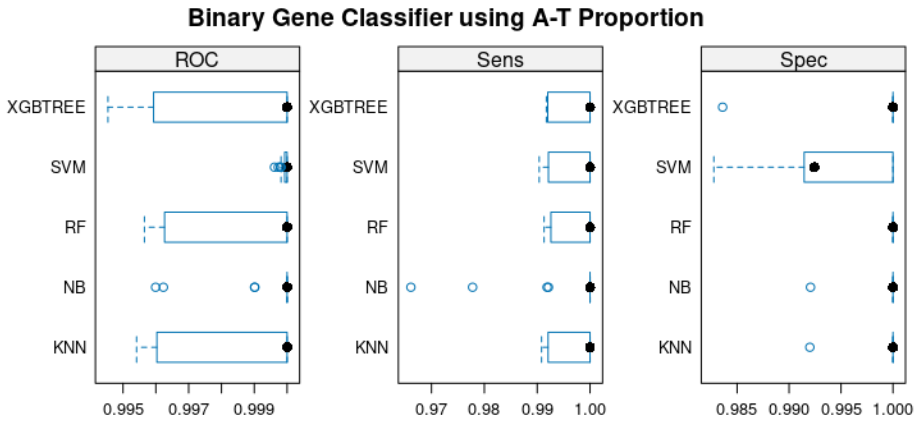

FIgure 2.1 Performance Evaluation of Various Machine Learning Models

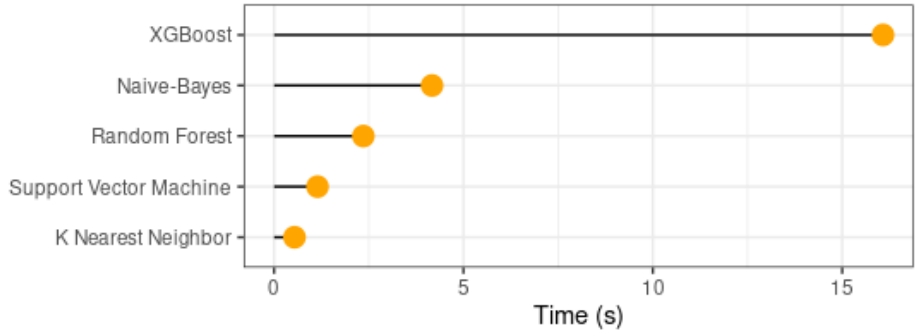

Figure 2.2 Prediction Time Required by Algorithm using A-T Proportion

Discussion

Relationships between two nucleotide proportion features (A-T, A-G, T-G) were generated. All scatter plots show a distinct clustering pattern between the 16S and 18S, but the A-T proportion has a closer distance between the two clusters (Figure 1.1). Another feature analysis was done by plotting out the proportions on a box and whisker plot (Figure 1.2). The thymine proportion difference between the 16S and 18S genes was small. Meanwhile, the guanine proportion had the largest difference between the two genes. Therefore, the A-T proportion was chosen as the two features to increase the difficulty for the machine learning models.

All models were evaluated using the ROC value, which is the proportion of true positives and negatives correctly classified (Flach 2016). Every model had an accuracy of over 99% so it

was difficult to compare the accuracy between models (Figure 2.1). While it was known that the 16S mitochondrial gene is known to have a high mutation rate (Yang *et al*., 2014), the similarity between the 16S and 18S was still overestimated. For future model evaluations, more genes with closer ancestry than the 16S and 18S should be used instead, given enough samples in online databases. When all the models are practically perfect, it depends on the time the algorithm takes to acquire this accuracy. The run times may depend on the computer used and can vary during the day, but KNN (~0.5 s) takes roughly half the time it takes for SVM (~1.0 s). Random Forest (~4.0 s) takes roughly four times as long as KNN (Figure 2.2). For reference, the computer used for this analysis runs a Ryzen 7 5000 Series processor on a Linux operating system.

In conclusion, when features cluster well in relation to each other, as seen in Figure 1.1, K Nearest Neighbor is one of the most optimal models to use for binary gene classification, due to both its accuracy and time performance.

References

Arango-Argoty, G., Garner, E., Pruden, A., Heath, L. S., Vikesland, P., & Zhang, L. (2018). DeepARG: a deep learning approach for predicting antibiotic resistance genes from metagenomic data. *Microbiome*, *6*, 1-15.

Flach, P. A. (2016). ROC analysis. In *Encyclopedia of machine learning and data mining* (pp. 1-8). Springer.

Mahé, P., & Tournoud, M. (2018). Predicting bacterial resistance from whole-genome sequences using k-mers and stability selection. *BMC bioinformatics*, *19*, 1-11.

Nugent, C. M., & Adamowicz, S. J. (2020). Alignment-free classification of COI DNA barcode data with the Python package Alfie. *Metabarcoding and Metagenomics*, *4*, e55815.

Yang, L., Tan, Z., Wang, D., Xue, L., Guan, M. X., Huang, T., & Li, R. (2014). Species identification through mitochondrial rRNA genetic analysis. *Scientific reports*, *4*(1), 4089.

R Code

```
#############################################################

# PART 1.1 - Loading NCBI Squamata Dataset for 18S rRNA

Squamata18S_search <- entrez_search(db = "nuccore", term = "Squamata[ORGN] AND 18S rRNA AND biomol_rRNA[PROP]")


# Entrez Search Results with 498 hits as of October 26, 2023
```

Squamata18S_search

All 498 hits were acquired and use_history set to TRUE due to size of search hits

Squamata18S_search <- entrez_search(db = "nuccore", term = "Squamata[ORGN] AND 18S rRNA AND biomol_rRNA[PROP]", retmax = 498, use_history = T)

#Summary of NCBI Squamata hits acquired

Squamata18S_summ <- entrez_summary(db = "nuccore", web_history =Squamata18S_search$web_history)

#Checking the number of unique species in the summary

unique(extract_from_esummary(Squamata18S_summ, "organism"))

Sq18Stable <- table(extract_from_esummary(Squamata18S_summ, "organism"))

Sq18Stable

Turning the species count table into a dataframe and plotting a frequency bar chart for the species

Sq18S_species_count <- as.data.frame(Sq18Stable)

Sq18plot <- ggplot(Sq18S_species_count, aes(x=Var1, y=Freq)) + geom_bar(stat = "identity") +

labs(title = "Species count of Squamata 18S Hits", x = "Species", y = "Frequency") +

coord_flip()

Bar chart shows an overrepresentation of the Eublepharis macularis species, but should not be a huge issue because this is not a taxonomy classifier

Sq18plot

Fetch Squamata 18S rRNA data from NCBI's nucleotide database using the saved web history variable. Fasta files for all species are saved into one file on the physical drive. Convert fasta sequence into Bioconductor DNAStringSet object.

Squamata18S_fetch <- entrez_fetch(db = "nuccore", web_history = Squamata18S_search$web_history, rettype = "fasta")

write(Squamata18S_fetch, "squamata18S_fetch.fasta", sep = "\n")

```
stringSet <- readDNAStringSet("squamata18S_fetch.fasta")


# Convert to dataframe format

dfSquamata18S <- data.frame(Squamata18S_Title = names(stringSet), Squamata18S_Sequence = paste(stringSet))

view(dfSquamata18S)


# Double checking to make sure all entries are 18S rRNA. Two entries do not contain mention of 18S rRNA in the table. Filtering out the two entries.

table(str_detect(dfSquamata18S$Squamata18S_Title, "18S"))


dfSquamata18S <- dfSquamata18S %>%

  filter(str_detect(dfSquamata18S$Squamata18S_Title, "18S"))

table(str_detect(dfSquamata18S$Squamata18S_Title, "18S"))


# A lot of entries contain the word "Predicted" and have the species name starting on the third word. Quick check showing that all but two entries share this pattern.

table(str_detect(dfSquamata18S$Squamata18S_Title, "PREDICTED:"))


# Create a new column with species name extracted from the fasta headers using the "Predicted" and third-fourth word pattern. The two edge cases are dealt with in a different pattern, extracting the second word

dfSquamata18S <- dfSquamata18S %>%

  mutate(Species_Name = if_else(str_detect( Squamata18S_Title, "PREDICTED:"), word(Squamata18S_Title, 3L, 4L), word(Squamata18S_Title, 2L)))


# Labeling all entries with the 18S markercode prior to merging with the BOLD data frame.

dfSquamata18S <- dfSquamata18S %>%

  mutate(markercode = "18S")


# Rearranging the columns
```

```
dfSquamata18S <- dfSquamata18S[, c("Squamata18S_Title", "Species_Name",
"Squamata18S_Sequence", "markercode")]

# Renaming the variables to better match up with the BOLD data set

names(dfSquamata18S) <- c("processid", "species_name", "nucleotides", "markercode")

view(dfSquamata18S)

summary(nchar(dfSquamata18S$nucleotides))


# Very good concentrated nucleotide count distribution for the 18S entries in the dataset

hist(nchar(dfSquamata18S$nucleotides))

############################################################

# PART 1.2 - Loading Squamata BOLD Dataset for 16S


#Load Squamata dataset from BOLD into a dataframe

dfBOLDSquamata <-
read_tsv("http://www.boldsystems.org/index.php/API_Public/combined?taxon=Squamata&form
at=tsv")

unique(dfBOLDSquamata$markercode)


# Filter out the entries that do not contain 16S rRNA.

dfBOLDSquamata16S <- dfBOLDSquamata %>%

  filter(markercode == "16S")

unique(dfBOLDSquamata16S$markercode)


# BOLD 16S Dataset contains 210 unique species as of October 26, 2023. It also contains 31 NA
values for species names.

unique(dfBOLDSquamata16S$species_name)

sum(is.na(dfBOLDSquamata16S$species_name))


# If the species name is NA, paste the genus name and "sp." together into the species name
column.
```

```
dfBOLDSquamata16S$species_name <- if_else(is.na(dfBOLDSquamata16S$species_name),
paste(dfBOLDSquamata16S$genus_name, "sp.", sep = " "),
dfBOLDSquamata16S$species_name)


sum(is.na(dfBOLDSquamata16S$species_name))


# Very broad distribution when compared to the 18S rRNA dataset from NCBI. Will require
some filtering keep the distribution more consistent with the NCBI dataset.

summary(nchar(dfBOLDSquamata16S$nucleotides))

hist(nchar(dfBOLDSquamata16S$nucleotides))


#Filtering Squamata 16S based on nucleotide length within the first and third quartiles

q1 <- quantile(nchar(dfBOLDSquamata16S$nucleotides), probs = 0.25, na.rm = TRUE)

q3 <- quantile(nchar(dfBOLDSquamata16S$nucleotides), probs = 0.75, na.rm = TRUE)


dfBOLDSquamata16S <- dfBOLDSquamata16S %>%

  filter(((str_count(nucleotides) >= q1 & str_count(nucleotides) <= q3)))

summary(nchar(dfBOLDSquamata16S$nucleotides))

hist(nchar(dfBOLDSquamata16S$nucleotides))


################################################################

# PART 1.3 - Merging NCBI and BOLD Datasets

dfJoin <- full_join(dfSquamata18S, dfBOLDSquamata16S, by = c("processid", "species_name",
"nucleotides", "markercode"))


# Creating new nucleotides2 column that will contain a DNAStringSet object format of the
sequences

dfJoin <- dfJoin %>%

  select(c("processid", "species_name", "nucleotides", "markercode")) %>%

  mutate(nucleotides2 = nucleotides)

dfJoin$nucleotides2 <- DNAStringSet(dfJoin$nucleotides2)
```

```
# Removing gaps/Ns if they belong to the start or end of the sequence. Removing all occurrences of gaps in nucleotide sequences. Filter out the entries if they contain more than 10% N's in the sequence

dfJoinClassifier <- dfJoin %>%
  mutate(nucleotides2 = str_remove(nucleotides, "^[-N]+")) %>%
  mutate(nucleotides2 = str_remove(nucleotides2, "[-N]+$")) %>%
  mutate(nucleotides2 = str_remove_all(nucleotides2, "-+")) %>%
  filter(str_count(nucleotides2, "N") <= (0.1 * str_count(nucleotides)))


dfJoinClassifier$nucleotides2 = DNAStringSet(dfJoinClassifier$nucleotides2)


# Add frequency counts of each nucleotide base for each entry

dfJoinClassifier <- cbind(dfJoinClassifier,
as.data.frame(letterFrequency(dfJoinClassifier$nucleotides2, letters = c("A", "C","G", "T"))))

# Add adenine, thymine, and guanine percentage of sequences as columns

dfJoinClassifier$Aprop <- (dfJoinClassifier$A) / (dfJoinClassifier$A + dfJoinClassifier$T +
dfJoinClassifier$C + dfJoinClassifier$G)

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

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


# Quick exploration of clustering between the different proportions of nucleotide bases. All of them have fairly defined cluster groups for 18S and 16S. Adenine and Thymine proportion was chosen moving on, since the clustering was slightly closer and might challenge the algorithms better

ggplot(data = dfJoinClassifier, aes(x = Aprop, y = Tprop, colour = markercode,
shape=markercode)) +
  geom_point() +
  xlab("Proportion of Adenine") +
  ylab("Proportion of Thymine") +
```

```
  theme_bw()

ggplot(data = dfJoinClassifier, aes(x = Aprop, y = Gprop, colour = markercode)) +
  geom_point() +
  xlab("Proportion of Adenine") +
  ylab("Proportion of Guanine") +
  theme_bw()

ggplot(data = dfJoinClassifier, aes(x = Gprop, y = Tprop, colour = markercode)) +
  geom_point() +
  xlab("Proportion of Guanine") +
  ylab("Proportion of Thymine") +
  theme_bw()

# Change class of nucleotides from DNAStringSet back to characters for easier compatibility with machine learning functions
dfJoinClassifier$nucleotides2 <- as.character(dfJoinClassifier$nucleotides2)

#Seeing counts by markercode
table(dfJoinClassifier$markercode)

# Determining how much samples to use for the validation set based on 25% of the lowest sampled gene in the dataframe
validation_num <- table(dfJoinClassifier$markercode)
validation_num <- min(validation_num)*0.25
validation_num <- floor(validation_num)
# The amount of samples in the training set will then be the difference between the total and the validation sample size
training_num <- table(dfJoinClassifier$markercode)
training_num <- min(training_num) - validation_num
```

```
# Randomly sampling for the validation dataset
set.seed(202)
dfValidation <- dfJoinClassifier %>%
  group_by(markercode) %>%
  sample_n(validation_num)

# Checking that the validation dataset has even numbers for both gene classes
table(dfValidation$markercode)

#Filtering out the entries that are already in the validation dataset and randomly sampling for the training dataset
set.seed(102)
dfTraining <- dfJoinClassifier %>%
  filter(!processid %in% dfValidation$processid) %>%
  group_by(markercode) %>%
  sample_n(training_num)

# Checking that the training dataset also even numbers between gene classes
table(dfTraining$markercode)

#Checking our variable names and the column numbers
names(dfTraining)
names(dfValidation)

# Dividing Training and Validation Datasets into two more simplified datasets that only contain the necessary features: A+T Proportion, and Dimer Proportions. This will allow us to compare the effectiveness of both feature types in classification with the Caret package
dfTraining_ATprop <- dfTraining[,c(4,10:11)]
dfValidation_ATprop <- dfValidation[,c(4,10:11)]
```

```
names(dfTraining_ATprop)
names(dfValidation_ATprop)


table(dfTraining_ATprop$markercode)
table(dfValidation_ATprop$markercode)


# Quick analysis of how important some predictors are, prior to training the models
# Data is normalized with the range method using the preprocess function.
preProcess_ATGprop <- preProcess(dfTraining[,c("Aprop","Tprop", "Gprop")], method = "range")
dfATprop_norm <- predict(preProcess_ATGprop, newdata = dfTraining)
featurePlot(x= dfATprop_norm[,c("Aprop","Tprop", "Gprop")], y = factor(dfTraining_ATprop$markercode), plot = "box", labels = c("",""))
 ############################################################
# PART 2 Comparing ML algorithms for classification using Caret
# Code adapted from https://www.machinelearningplus.com/machine-learning/caret-package/#72hyperparametertuningusingtunelength
# Define the training control for Caret algorithms
fitControl <- trainControl(
  method = 'boot',                  # bootstrapping (Default method for Caret)
  number = 25,                      # number of folds
  savePredictions = 'final',        # saves predictions for optimal tuning parameter
  classProbs = T,                   # class probabilities should be returned
  summaryFunction=twoClassSummary  # results summary function
)


# Training all the algorithms on AT proportion features to predict markercode. Time is recorded for each algorithm to determine performance. Tune length is number of different parameter adjustments attempted. In this case, it was set at 2 (eg. Evaluating 3 sampled predictors vs. 10 sampled predictors for random forest mtry nodes)
```

```
#Random Forest - AT Proportion

starttime = Sys.time()

set.seed(50)

modelATprop_rf = train(markercode ~ ., data=dfTraining_ATprop, method='rf', tuneLength=2, trControl = fitControl)

timeATprop_rf <- print(Sys.time()- starttime)

modelATprop_rf


#Support Vector Machine (Linear) - AT Proportion

starttime = Sys.time()

set.seed(60)

modelATprop_svm = train(markercode ~ ., data=dfTraining_ATprop, method='svmLinear', tuneLength=2, trControl = fitControl)

timeATprop_svm <- print(Sys.time()- starttime)

modelATprop_svm


#XGBoost (Tree) - AT Proportion

starttime = Sys.time()

set.seed(70)

modelATprop_xgb = train(markercode ~ ., data=dfTraining_ATprop, method='xgbTree', tuneLength=2, trControl = fitControl, verbosity = 0)

timeATprop_xgb <- print(Sys.time()- starttime)

modelATprop_xgb


# K nearest neighbour - AT Proportion

starttime = Sys.time()

set.seed(80)

modelATprop_knn = train(markercode ~ ., data=dfTraining_ATprop, method='knn', tuneLength=2, trControl = fitControl)
```

```
timeATprop_knn <- print(Sys.time()- starttime)

modelATprop_knn


# Naive-Bayes - AT Proportion

starttime = Sys.time()

set.seed(90)

modelATprop_nb = train(markercode ~ ., data=dfTraining_ATprop, method='nb', tuneLength=2, trControl = fitControl)

timeATprop_nb <- print(Sys.time()- starttime)

modelATprop_nb

##################################################

# Compare the algorithms when the AT proportion is used as features

modelsATprop_compare <- resamples(list( RF=modelATprop_rf, XGBTREE=modelATprop_xgb, SVM=modelATprop_svm, KNN=modelATprop_knn, NB=modelATprop_nb))

summary(modelsATprop_compare)


# Models are compared on their performance by plotting according to ROC (Binary Classification Performance), Sensitivity (True Positive), and Specificity (True Negative)

scales <- list(x=list(relation="free"), y=list(relation="free"))

modelsATprop_plot <- bwplot(modelsATprop_compare, scales=scales, main="Binary Gene Classifier using A-T Proportion")

modelsATprop_plot


# Evaluating efficiency of models by creating a dataframe with the algorithm times and comparing them

AT_Time <- c(timeATprop_knn, timeATprop_nb, timeATprop_rf, timeATprop_svm, timeATprop_xgb)

MLnames <- c("K Nearest Neighbor", "Naive-Bayes", "Random Forest", "Support Vector Machine", "XGBoost")


#https://r-graph-gallery.com/267-reorder-a-variable-in-ggplot2.html
```

```
# Ordering the models from quickest to slowest
dfTime <- data.frame(MLnames, AT_Time)
dfTime <- dfTime %>%
  arrange(AT_Time) %>%
  mutate(MLnames=factor(MLnames, levels=MLnames))
dfTime


# Plotting out AT Proportion Model Classification Time
ggplot(dfTime, aes(x= MLnames, y = AT_Time)) +
  geom_segment( aes(xend=MLnames, yend=0)) +
  geom_point( size=4, color="orange") +
  coord_flip() +
  theme_bw() +
  xlab("") +
  ylab("Time (s)")


# Validating the trained models against the validation data sets. Confusion matrix generated to
confirm accuracy.
# Random Forest
predictionATprop_rf <- predict(modelATprop_rf, dfValidation_ATprop)
confusionMatrix(data = predictionATprop_rf, reference =
factor(dfValidation_ATprop$markercode))
# Support Vector Machine
predictionATprop_svm <- predict(modelATprop_svm, dfValidation_ATprop)
confusionMatrix(data = predictionATprop_svm, reference =
factor(dfValidation_ATprop$markercode))
# XGBoost
predictionATprop_xgb <- predict(modelATprop_xgb, dfValidation_ATprop)
confusionMatrix(data = predictionATprop_xgb, reference =
factor(dfValidation_ATprop$markercode))
```

```
# K Nearest Neighbor

predictionATprop_knn <- predict(modelATprop_knn, dfValidation_ATprop)

confusionMatrix(data = predictionATprop_knn, reference =
factor(dfValidation_ATprop$markercode))

# Naive-Bayes

predictionATprop_nb <- predict(modelATprop_nb, dfValidation_ATprop)

confusionMatrix(data = predictionATprop_nb, reference =
factor(dfValidation_ATprop$markercode))

#############################################################
```