######Assignment 2 a) Supervised Machine Learning and Classification Using Sequence Features----

#Script by Sichong Xu

#PART 1 Introduction----

#Machine learning techniques have been used extensively in the computational biology field, especially for large-scale gene classification and prediction with complex sequence data analysis. Every nucleotide sequence corresponds to a gene; however, we are often only able to obtain fragments of sequence without knowing the encoding gene. Therefore, correct gene identification and classification are very crucial. The correct gene classification could lead to proper annotation and characterization of an organism's functional genomic profile (Silva et al., 2019). Gene classification is counted as one of the significant biological applications of supervised machine learning. Supervised learning uses labeled data as the training set, to teach the model to recognize patterns within the input-output pairs and learn how to categorize data properly or predict desired outcomes (Osisanwo et al., 2017). The Random Forest method shows better performance and can form a more robust model among other classical supervised learning approaches. (Silva et al., 2019) Random Forest is an ensemble learning method, which builds on a collection of uncorrelated decision trees. (Qi, 2012). The randomForest package in the R program can be used to adopt the random forest method.

#For this assignment, the main objective is by using the R program, building a solid classifier to determine gene type from different nucleotide sequences with random forest method, and evaluating its performance and accuracy. The data used in the assignment will be obtained from the Nucleotide database of NCBI, which includes the sequences from several sources, including GenBank, RefSeq, TPA, and PDB. The selected taxonomic group is *Bombus* genus. Two genes are the cytochrome oxidase subunit I (COI) gene and cytochrome B (CytB) gene; both are protein-coding and mitochondrial genes and have similar properties.

#PART 2 Data Acquisition, Exploration, Filtering, and Quality Control----

```
# Load the required R packages first.
library(rentrez)
library(seqinr)
library(Biostrings)
library(tidyverse)
library(stringr)
library(randomForest)
library(caret)
library(reshape)
library(ggplot2)
```

#I use the entrez_dbs() function to retrieve the names of databases that are accessible through E-utilities and determine which databases should be used to obtain data from.
entrez_dbs()
#"Nucleotide" database is appropriate for theme a) Supervised Machine Learning and Classification Using Sequence Features. And "nuccore" is the correspondent database name for E-utilities.

#entrez_db_summary() and entrez_db_searchable() present the summary information and searchable fields that can be used in "nuccore" database.
entrez_db_summary(db = "nuccore")
entrez_db_searchable(db = "nuccore")
#"GENE: Name of gene associated with sequence", "SLEN: Length of sequence", "ORGN: Scientific and common names of organism, and all higher levels of taxonomy" are example fields of interest for this assignment theme.

#For this assignment, I choose *Bombus* genus as my taxonomic groups, two genes are cytochrome oxidase subunit I (COI) and cytochrome B (CytB).

#I use entrez_search () to obtain nucleotide sequences for *Bombus* from database "nuccore". The length for sequence form COI gene is set between 500 to 1200 bp, since a COI sequence fragment that has at least 500 bp is desired for species identification.
Bombus_COI_search <- entrez_search(db = "nuccore", term = "(Bombus[ORGN] AND COI[Gene] AND 500:1200[SLEN])")
Bombus_COI_search
#Result shows there are 5110 hits for this search term, so "retmax" is set as 1000 to obtain 1000 sequence records for COI gene. Because a large amount of COI sequence is obtained from "nuccore", I choose to use web_history object.
Bombus_COI <- entrez_search(db = "nuccore", term = "(Bombus[ORGN] AND COI[Gene] AND 500:1200[SLEN])", retmax = 1000, use_history=TRUE)
Bombus_COI

Check the NCBI web history object for COI sequences.
Bombus_COI$web_history

#Look at the unique ID of returned sequences.
Bombus_COI$ids

#I use length() to check the number of unique ID returned by the search, which should be the same as "retmax"'s setting
length(Bombus_COI$ids)

#I use the same method to obtain the *Bombus* sequence from gene cytochrome B. To avoid getting sequence with whole mitochondrial genome, the sequence length is also restricted between 500 to 1200 bp for partial gene. Moreover, as Jacqueline mentioned, researchers could use different name method "Cyt B" or "CytB" for this gene, so both name methods are checked.

```
Bombus_CytB_search <- entrez_search(db = "nuccore", term = "(Bombus[ORGN] AND CytB[Gene] AND 500:1200[SLEN])")
Bombus_CytB_search
Bombus_Cyt_B_search <- entrez_search(db = "nuccore", term = "(Bombus[ORGN] AND Cyt B[Gene] AND 500:1200[SLEN])")
Bombus_Cyt_B_search
```

#Results show there are only 159 hits for "CytB" and 0 hits for "Cyt B", so a total of 159 hits will need to be retained for the downstream analysis, the "retmax" is set as 159.

```
Bombus_CytB <- entrez_search(db = "nuccore", term = "(Bombus[ORGN] AND CytB[Gene] AND 500:1200[SLEN])", retmax = 159)
Bombus_CytB
length(Bombus_CytB$ids)
```

#I use entrez_summary() to access the metadata of every returned sequence.

```
BCytB_summ <- entrez_summary(db = "nuccore", id = Bombus_CytB$ids)
BCytB_summ
BCOI_summ <- entrez_summary(db = "nuccore", web_history = Bombus_COI$web_history, retmax = 500) #Maximum 500 web history UIDs can be used for esummary.
BCOI_summ
```

#There is a list of 32 metadata for each record, such as the sequence length and genome information.

#Then I use extract_from_esummary() function to extract the length of each COI and CytB sequence record, and I want to draw 2 histograms to see the distribution of sequence length.

```
L_COI <- extract_from_esummary(BCOI_summ, "slen")
dfL_COI <- as.data.frame(L_COI)
dfL_COI <- dfL_COI %>%
  remove_rownames
hist(dfL_COI$L_COI, main = "Histogram of COI Sequence Length", xlab = "Sequence Length")

L_CytB <- extract_from_esummary(BCytB_summ, "slen")
dfL_CytB <- as.data.frame(L_CytB)
dfL_CytB <- dfL_CytB %>%
  remove_rownames
hist(dfL_CytB$L_CytB, main = "Histogram of CytB Sequence Length", xlab = "Sequence Length")
```

#As the histograms illustrate, there is no record that falls out of the expected range set for sequence length (500: 1200 bp) for both COI and CytB genes; most of the sequences have a length around 650 to 700 bp.

```
#I also check the metadata "title" and "genome" for COI sequences.
extract_from_esummary(BCOI_summ, "title")
extract_from_esummary(BCOI_summ, "genome")
#As expected, all the records belong to the mitochondrion.

#I then use entrez_fetch() to download COI and CytB sequence data into the FASTA format.
Bombus_COI_fetch <- entrez_fetch(db = "nuccore", web_history = Bombus_COI$web_history,
rettype = "fasta", retmax = 1000)
Bombus_CytB_fetch <- entrez_fetch(db = "nuccore", id = Bombus_CytB$ids, rettype = "fasta")

#I check the class and take a look at the head portion of the fetch results.
class(Bombus_COI_fetch) #Character
head(Bombus_CytB_fetch)
head(Bombus_COI_fetch)

#I set the right working directory, and write all the sequence data of two genes into FASTA files,
with new line (\n) deliminator.
#setwd("~/Desktop/6210 Software Tools/Assignment 2")
write(Bombus_COI_fetch, "Bombus_COI_fetch.fasta", sep = "\n")
write(Bombus_CytB_fetch, "Bombus_CytB_fetch.fasta", sep = "\n")
#I view both FASTA files in TextEdit, contents look nice.

#I use readDNAStringSet() function from "Biostrings" package to read the FASTA files as DNA
StringSet objects.
BCOI_String <- readDNAStringSet("Bombus_COI_fetch.fasta")
BCytB_String <- readDNAStringSet("Bombus_CytB_fetch.fasta")

#And I check if the class of "BCOI_String' and "BCytB_String" are set as expected.
class(BCOI_String)
class(BCytB_String)
#Both objects' class is DNAStringSet.

#Then I put COI and CytB sequences into data frame format for the following exploration.
dfBCOI<- data.frame(BCOI_Title = names(BCOI_String), BCOI_Sequence = paste(BCOI_String))
View(dfBCOI)
dfBCytB <- data.frame(BCytB_Title = names(BCytB_String), BCytB_Sequence =
paste(BCytB_String))
View(dfBCytB)

#I use word() function to select specific words by position from the string and add new column
as the species name of each sequence.
dfBCOI$Species_Name <- word(dfBCOI$BCOI_Title, 2L, 3L)
dfBCytB$Species_Name <- word(dfBCytB$BCytB_Title, 2L, 3L)
```

```
#I use str_extract_all() to extract the gene name "COI" from "BCOI_Title", and I use mutate() to add a new column for gene name to the data frame.
dfBCOI_G <- dfBCOI %>%
  mutate(Gene_Name = str_extract_all(BCOI_Title, "COI"))

unique(dfBCOI_G$Gene_Name) #To check what gene I have in the data frame.

View(dfBCOI_G)

#For CytB sequences, I firstly use str_to_lower() function to lowercase the whole "BCytB_Title" content, because researchers used "cytb", "Cytb" and "cytB" when collected the sequence information. After, I use str_extract_all() again to extract the gene name "cytb" from "BCytB_Title" as a new column.
dfBCytB_G <- dfBCytB %>%
  mutate(Gene_Name = str_to_lower(BCytB_Title)) %>%
  mutate(Gene_Name = str_extract_all(Gene_Name, "cytb"))

unique(dfBCytB_G$Gene_Name)

View(dfBCytB_G)

#I use colnames() function to rename the columns, which will help me to combine two data frames later.
colnames(dfBCOI_G) <- c("Title", "Sequence", "Species_Name", "Gene_Name")
colnames(dfBCytB_G) <- c("Title", "Sequence", "Species_Name", "Gene_Name")

#I combine two data frames by rows, columns are matched by names.
dfBCOI_cytb <- bind_rows(dfBCOI_G, dfBCytB_G)
#Rows for COI are at the top part, and rows for cytb are at the bottom of the data frame.

#To check the class and dimension of new combined data frame.
class(dfBCOI_cytb)
dim(dfBCOI_cytb)

View(dfBCOI_cytb)
#Now, the new data frame "dfBCOI_cytb" has 1159 observations, and 4 variables.

#By looking at the sequences themselves, all the sequences look tidy. Nevertheless, it is better to adopt the sequence cleaning steps as Sally did in script9 before actually applying the random forest method.
#I used str_remove() to trim off any gaps(-) or Ns at the beginning(^) and end($) of the sequence. Then I use str_remove_all() function to remove all the gaps in the entire sequence
```

string. I set 5% of the total sequence length as the restriction of internal Ns, so sequence with a higher degree (>5%) of internal missing data will be filtered out.

```
dfGene <- dfBCOI_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(Sequence2)))

identical(dfGene$Sequence, dfGene$Sequence2)
#After applying the sequence cleaning, sequences from "Sequence" and "Sequence2" are exactly the same, so the data downloaded from NCBI seems to have good quality already.

#I replace all the "cytb" in dfGene to "CytB".
dfGene[dfGene == "cytb"] <- "CytB"

#Have a quick check on the final data frame "dfGene" that will be work on.
summary(str_count(dfGene$Sequence2))
unique(dfGene$Species_Name)
unique(dfGene$Gene_Name)
class(dfGene)
view(dfGene)
#The length of all sequences is between 507 and 929 bp. There are 76 unique species names of Bombus (include both scientific species name and interim species name). And unique genes are COI and CytB.


#PART 3 Gene Sequence Analysis and Classification----

#In order to calculate the sequence feature, I use DNAStringSet() convert the nucleotides to a DNAStringSet again.
dfGene$Sequence2 <- DNAStringSet(dfGene$Sequence2)
class(dfGene$Sequence2)

#I use letterFrequency() to calculate the frequency of nucleotide letters: "A", "T", "C", "G" in each sequence string, and add the results as a new column with cbind() function.
dfGene <- cbind(dfGene, as.data.frame(letterFrequency(dfGene$Sequence2, letters = c("A", "T","C", "G"), as.prob = FALSE)))

#Then I convert the frequency of "A", "T", "C", "G" to proportions of total nucleotides, also add them as new columns at the end of data frame.
dfGene$Aprop <- (dfGene$A) / (dfGene$A + dfGene$T + dfGene$C + dfGene$G)

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

```
dfGene$Cprop <- (dfGene$C) / (dfGene$A + dfGene$T + dfGene$C + dfGene$G)

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

#The above steps of getting proportions of "A", "T", "C", and "G" could be done in one
command line by setting "as.prob = TRUE".
#dfGene <- cbind(dfGene, as.data.frame(letterFrequency(dfGene$Sequence2, letters = c("A",
"T","C", "G"), as.prob = TRUE)))

#Other than just having the frequency of the single nucleotide, I also add the frequency of
dinucleotide and trinucleotide (k-mers of length 2 and 3) to the data frame.
dfGene <- cbind(dfGene, as.data.frame(dinucleotideFrequency(dfGene$Sequence2, as.prob =
TRUE)))

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

#Recheck the data frame to see if it is in the proper format.
View(dfGene)

#Convert the format of "Gene_Name" and "Sequence2" to character data.
class(dfGene$Gene_Name)
class(dfGene$Sequence2)
dfGene$Gene_Name <- as.character(dfGene$Gene_Name)
dfGene$Sequence2 <- as.character(dfGene$Sequence2)
class(dfGene$Gene_Name)
class(dfGene$Sequence2)
table(dfGene$Gene_Name)
#Currently, we have 1000 records for COI gene, and 159 records for CytB gene.

#Since I am applying supervised machine learning, I will need a training data set to help find the
rules to separate the data and a validation data set to validate the rules. I plan to sample 80% of
the total data as the training group, and the rest 20% will be the validation group.
#Because of the considerable difference in sample size between COI and CytB gene, I will set
the number of data for both the training and validation data sets based on the smaller gene
size.
#Use the floor() function to round the result to the nearest largest integer. ceiling() can round
the result to the nearest smallest integer.
smaller_gene <- min(table(dfGene$Gene_Name))
smaller_gene
```

```
floor(0.2 * smaller_gene)
#The validation data set will have 31 records for each gene, in total 62.
ceiling(0.8 * smaller_gene)
#The training data set will have 128 records for each gene, in total 256.

#I use sample_n()to randomly get the required number of rows (31) from each group (COI and CytB), and output as the validation data set "dfGValidation".
#set.seed() function makes sure the same records will be chosen every time.
set.seed(2)

dfGValidation <- dfGene %>%
  group_by(Gene_Name) %>%
  sample_n(floor(0.2 * smaller_gene))

table(dfGValidation$Gene_Name)

# To have the records in the training set not overlap with the records in the validation data set, the validation data set needs to be filtered out of the original data frame first. Then same as the above procedure, 128 samples will be selected for both groups and will be saved into training group "dfGTraining" to train the model later.
set.seed(22)

dfGTraining <- dfGene %>%
  filter(!Title %in% dfGValidation$Title) %>%
  group_by(Gene_Name) %>%
  sample_n(ceiling(0.8 * smaller_gene))

table(dfGTraining$Gene_Name)

#I use data.frame(colnames()) to get the position of each variables of the data frame.
data.frame(colnames(dfGTraining))

#Because I chose random forest as the method for supervised learning, I use the randomForest() from the "randomForest' package to create the model. To build the gene classifier for COI and CytB, I first use the single nucleotide feature (columns 10-13) as predictors, and the response variable is the gene name. Furthermore, I set the number of decision trees as 120 (ntree = 120), that every input can be predicted at least a few times.
#I am setting the seed here to ensure obtaining the same classifier every time. Because random forest uses bagging method, the decision trees are chosen for each run randomly, and each tree's input data is also randomly sampled with replacement.
set.seed(22222)

Gene_classifier <- randomForest(x = dfGTraining[, 10:13], y =
as.factor(dfGTraining$Gene_Name), ntree = 120, importance = TRUE, proximity=TRUE)
```

```
#View the random forest results.
Gene_classifier
#The classifier result shows that it is a good binary classifier to separate COI's sequences and CytB's sequences for Bombus, even by using the simple nucleotide feature as the predictor. OOB estimate of error rate is 1.56%.

#To check the importance of each predictor (the proportion of "A" "T" "C" "G" single nucleotide).
importance(Gene_classifier)

#To check the classification performance with confusion matrix.
confusion <- as.data.frame(Gene_classifier$confusion)
confusion
#The confusion matrix shows that 2 of gene COI are mis-predicted as CytB, and 2 sequences of gene CytB are mis-predicted as COI.

#Calculate the accuracy of classifier.
Accuracy <- (confusion[1,1] + confusion[2,2]) / (confusion[1,1] + confusion[1,2] + confusion[2,1] + confusion[2,2])
Accuracy

#I am checking the times of case are "out-of-bad", and out-of-bag error rate for each tree.
Gene_classifier$oob.times
Gene_classifier$err.rate

#I am plotting the OOB error rate of each tree.
plot(Gene_classifier$err.rate[,1], type = "l", main = "Out-of-bag Error Rate for Decision Tree", xlab = "Nubmer of Decision Tree", ylab = "OOB Error Rate", col = "blue")

#In order to check the universality of the newly built classifier, the classifier is used to predict the gene for the unseen validation data set.
predictValidation <- predict(Gene_classifier, dfGValidation[, c(4, 10:13)])

predictValidation
table(observed = dfGValidation$Gene_Name, predicted = predictValidation)
#By looking at the prediction result and the confusion matrix, the classifier does an excellent job; all the sequences are classified correctly.
```

```
#In addition, I build another gene classifier for COI and CytB by using the dinucleotide frequency (columns 14-29) as predictors.
set.seed(222222)

Gene_classifier2 <- randomForest(x = dfGTraining[, 14:29], y = as.factor(dfGTraining$Gene_Name), ntree = 120, importance = TRUE)

Gene_classifier2

predictValidation2 <- predict(Gene_classifier2, dfGValidation[, c(4, 14:29)])

table(observed = dfGValidation$Gene_Name, predicted = predictValidation2)

#To generate the confusion matrix and performance accuracy for Gene_classifier2
confusion2 <- as.data.frame(Gene_classifier2$confusion)
confusion2

Accuracy2 <- (confusion2[1,1] + confusion2[2,2]) / (confusion2[1,1] + confusion2[1,2] + confusion2[2,1] + confusion2[2,2])

#Compare the accuracy of 2 classifiers.
Accuracy2 #0.9960938
Accuracy #0.984375
#In terms of classification accuracy, using the dinucleotide frequencies as predictors seems to produce a better classifier for COI and CytB genes than single nucleotide frequencies.

#Even though these two genes have similar properties and both are mitochondrial genes, they are easy to classify by using simple features.
```

```
#PART 4 Visualization----

#After generating the classification model for COI and CytB sequences, I want to produce some graphs to evaluate the performance of the classifier.

#Here, I am plotting the classification error rate for OOB, and two genes COI and CytB for the first "Gene_classifier".
Error <- as.data.frame(Gene_classifier$err.rate)
summary(Error)

plot(Gene_classifier, main = "Error Rate of Gene Classifier")
lines(Error$OOB, col = "blue", lty = 1)
lines(Error$COI, col = "DeepPink", lty = 2)
lines(Error$cyt, col = "green", lty = 3)
legend('topright',legend = c("OOB", "COI", "CytB"),col= c("blue", "DeepPink", "green"),lty = 1:3, cex = 0.6)
#All the decision trees have error rate lower than 0.03 when predicting the class for COI, CytB, and out-of-bag samples in the training data set. Decision trees tend to have a more stable performance when predicting COI samples.

#I plan to draw 4 box plots to investigate the distribution of each nucleotide for each gene.
#I first extract the required information from "dfGene", and rename the column names.
Nucleotide_box1 <- dfGene[,c(4,10:13)]
colnames(Nucleotide_box1) <- c("Gene_Name", "A", "T", "C", "G")

#In order to have the box plot for 4 nucleotides together in one figure, I use melt() function from the "reshape" package to reshape data into the desired structure.
Nucleotide_box = melt(Nucleotide_box1)

#I use geom_boxplot() from "ggplot2" package to form a grouped box plot that can illustrate each nucleotide's distribution in terms of different genes.
ggplot(data = Nucleotide_box, aes(x = Gene_Name, y = value)) +
  geom_boxplot(aes(fill=Gene_Name), outlier.shape = 21,) +
  theme_bw() +
  facet_grid(~variable) +
  labs( x = "Gene", y = "Proportion of Nucleotide", fill = "Gene Name", title = "Nucleotide Sequence Composition VS. Gene") + theme(plot.title = element_text(hjust = 0.5))
#When looking at the box plot, it is obvious that nucleotides "A" and "T" are more abundant than "C" and "G" in both COI and CytB sequence. Moreover, the differences of "A" and "G" between the two genes are more distinct than "T" and "C"
```

```
#I also want to have a bar graph to show how relatively important each nucleotide is to the classification model.
Importance <- as.data.frame(Gene_classifier$importance)
barplot(Importance[,3], main = "Importance of Nucleotide Feature", xlab = "Nucleotide", ylab = "Importance", names.arg = c("A", "T", "C", "G"),
col=c("#E5562A","#491A5B","#8C6CA8","#BD1B8A","#7CB6E4"))
#This bar graph shows the importance of nucleotide "A" and "G" being higher than "T" and "C" in the gene classification model.

#I use the varImpPlot() from "randomForest" package to form the variables importance dot chart.
varImpPlot(Gene_classifier, type = 1, main = "Importance of Nucleotide Feature", lcolor = "grey", sort = TRUE)
#This dot chart of mean decrease of accuracy also shares the same observation as the above grouped box plot and bar graph. Nucleotide "A" and "G" have more importance.  "G" composition is the most important to the classifier among all 4 nucleotides, that if we drop predictor "G", the accuracy of the classification model will decrease the most.


#PART 5 Results and Discussion----

#This assignment aims to use supervised machine learning to generate a high-accuracy binary classifier that can categorize the Bombus's cytochrome oxidase subunit I (COI) gene and cytochrome B (CytB) gene by their nucleotide sequence composition. The region of the cytochrome oxidase subunit I (COI) gene is essential for animal DNA barcodes which can be used to identify species. As a result, having a classification model that can distinguish COI from other genes is critical, especially from the genes with a similar structure to COI, such as CytB. I downloaded the nucleotide records from NCBI, there are over 5000 sequences for COI, but only 159 belong to CytB that matches the sequence length requirement (500-1200 bp). The sample size for the CytB gene is not large enough compared to COI, but it is still possible for the model to learn patterns from the limited sample size and generate a classifier. More potential sequences for the CytB gene should be included in future studies to have a more comprehensive and accurate classification model.

#For the random forest approach, the single nucleotide frequencies and the dinucleotide frequencies were used as predictors to generate 2 classifiers. I plotted a grouped box plot (Figure 1) to evaluate and demonstrate the frequency of each nucleotide ("A", "T", "C", "G") in sequence for two genes. The box plot illustrates that in both COI and CytB sequences, the nucleotide "A" and "T" have a higher proportion than "C" and "G." Meanwhile, the nucleotide "A" and "G" 's difference between the two genes are more distinct than "T" and "C". These observed patterns of single nucleotides would be the keys for the first gene classifier. After I generated both classifiers, I created a variable importance bar graph and a dot chart to show how each single nucleotide frequency contributes to the classification model. The dot plot (Figure 2) is generated based on the mean decrease of accuracy. The result shows that if we
```

drop predictor "G" from the model, the accuracy of the classifier will decrease the most, so nucleotide "G" has the most importance to this specific classifier. In addition, I used the data from the confusion matrix to compare the accuracy of 2 classifiers. The classifier using dinucleotide frequencies as predictors has a higher accuracy score for separating sequences into the correct gene class than the classifier using single nucleotide frequencies as predictors. If I am going to have a larger project on this topic, I would try to create more classifiers with different nucleotide composition features as the predictors, then choose the one with the highest accuracy and lowest error rate to classify large-scale data.

#PART 6 Figures----

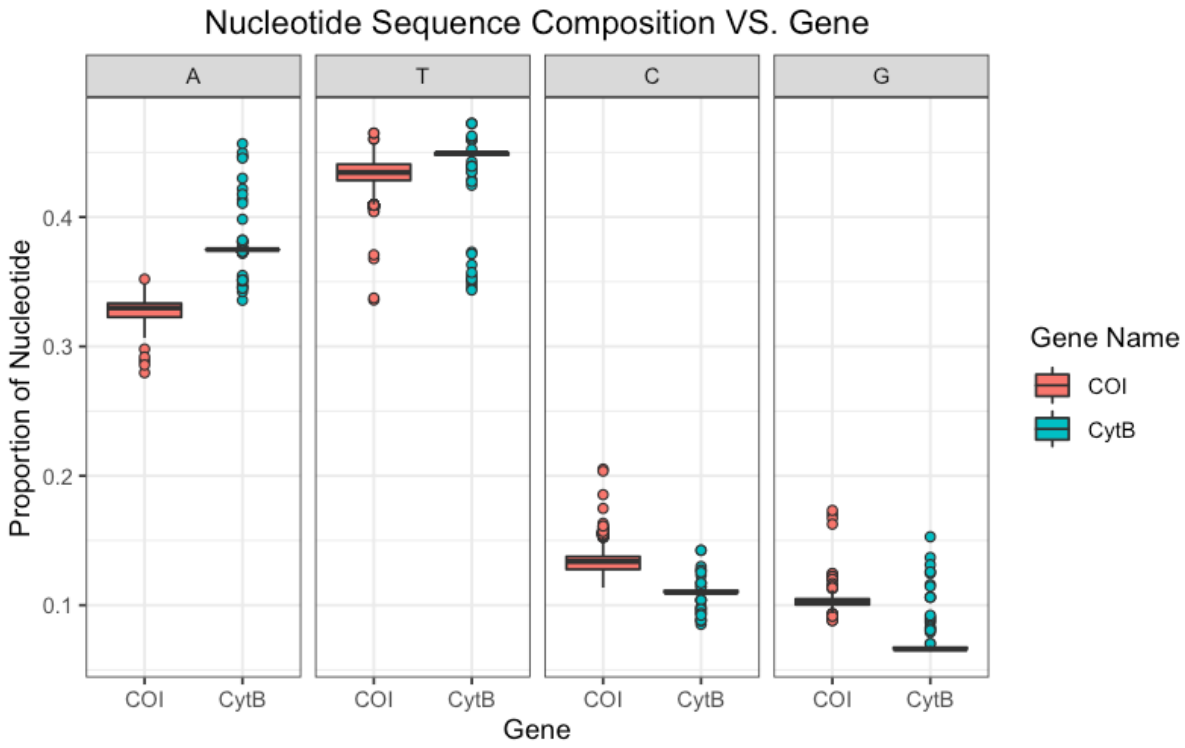

Figure 1. The grouped box plot shows the frequency of every single nucleotide ("A", "T", "C", "G") in sequence for the cytochrome oxidase subunit I (COI) gene and the cytochrome B (CytB) gene.
"A" and "T" have higher proportion in nucleotide sequence than "C" and "G" in both COI and CytB gene. Moreover, the differences of nucleotide "A" and "G" between the two genes are more distinct than "T" and "C".

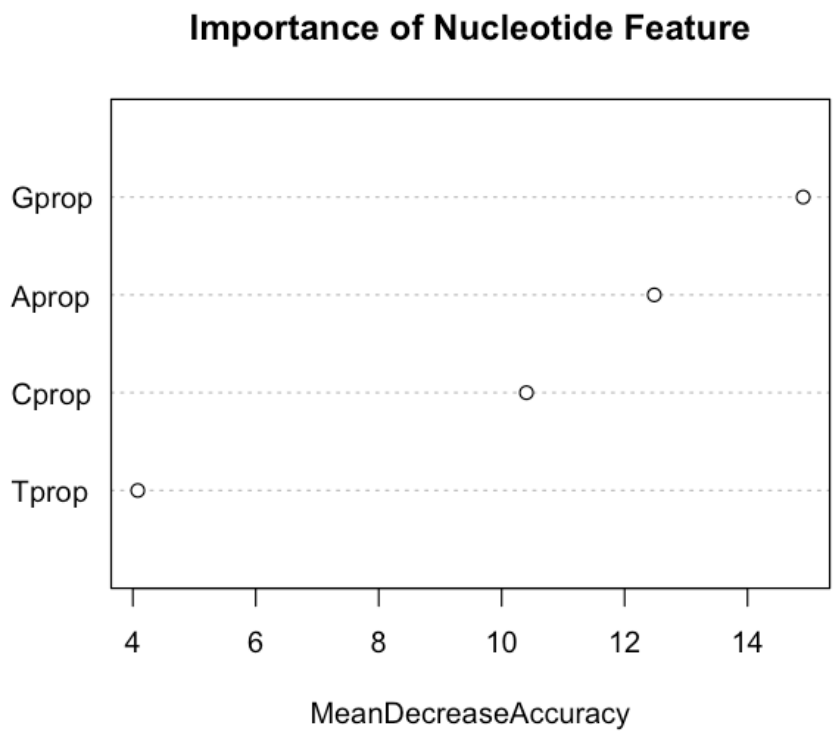

Figure 2 The dot plot () of mean decrease of classifier's accuracy.
The result shows that nucleotide "G" proportion is the most important to gene classifier, if we drop predictor "G" from the model, the accuracy of the classifier will decrease the most.

#PART 7 Acknowledgements----

#Chor Kwan Lam
#Exchanged the general idea of this assignment.

#Dr. Sarah Adamowicz
#Really appreciate the help on how to visualize the gene classifier.

#Jacqueline May
#Thanks for the generous general help and tips during help session.

#PART 8 Reference----

#Osisanwo, F. Y., Akinsola, J. E. T., Awodele, O., Hinmikaiye, J. O., Olakanmi, O., & Akinjobi, J. (2017). Supervised machine learning algorithms: classification and comparison. International Journal of Computer Trends and Technology (IJCTT), 48(3), 128-138.
#Qi, Y.J. (2012). Random Forest for Bioinformatics. Ensemble Machine Learning, 307–323. Springer US. https://doi.org/10.1007/978-1-4419-9326-7_11

#Silva, R., Padovani, K., Goes, F., & Cley Alves, R. (2019). A Random Forest Classifier for Prokaryotes Gene Prediction. 2019 8th Brazilian Conference on Intelligent Systems (BRACIS), 545–550. https://doi.org/10.1109/BRACIS.2019.00101
#https://blog.csdn.net/sinat_26917383/article/details/51308061

#https://cran.r-project.org/web/packages/rentrez/vignettes/rentrez_tutorial.html#using-ncbis-web-history-features

#https://stackoverflow.com/questions/37888619/difference-between-varimp-caret-and-importance-randomforest-for-random-fores

#https://stats.stackexchange.com/questions/36165/does-the-optimal-number-of-trees-in-a-random-forest-depend-on-the-number-of-pred/36183

#https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710

#https://www-users.cse.umn.edu/~kumar001/dmbook/ch4.pdf

#https://www.cnblogs.com/chenwenyan/p/6440141.html

#https://www.cnblogs.com/nxld/p/6062950.html

#https://www.guru99.com/r-random-forest-tutorial.html

#https://www.javaroad.cn/questions/292863

#https://www.r-bloggers.com/2015/06/variable-importance-plot-and-variable-selection/

#https://www.r-graph-gallery.com/265-grouped-boxplot-with-ggplot2.html

#ncbi.nlm.nih.gov/books/NBK25497/