Comparing Random Forest and Logistic Regression Machine Learning Classifiers for Gene Classification

Mirza Ali Murtuza Ahmadi

2024-10-27

Introduction

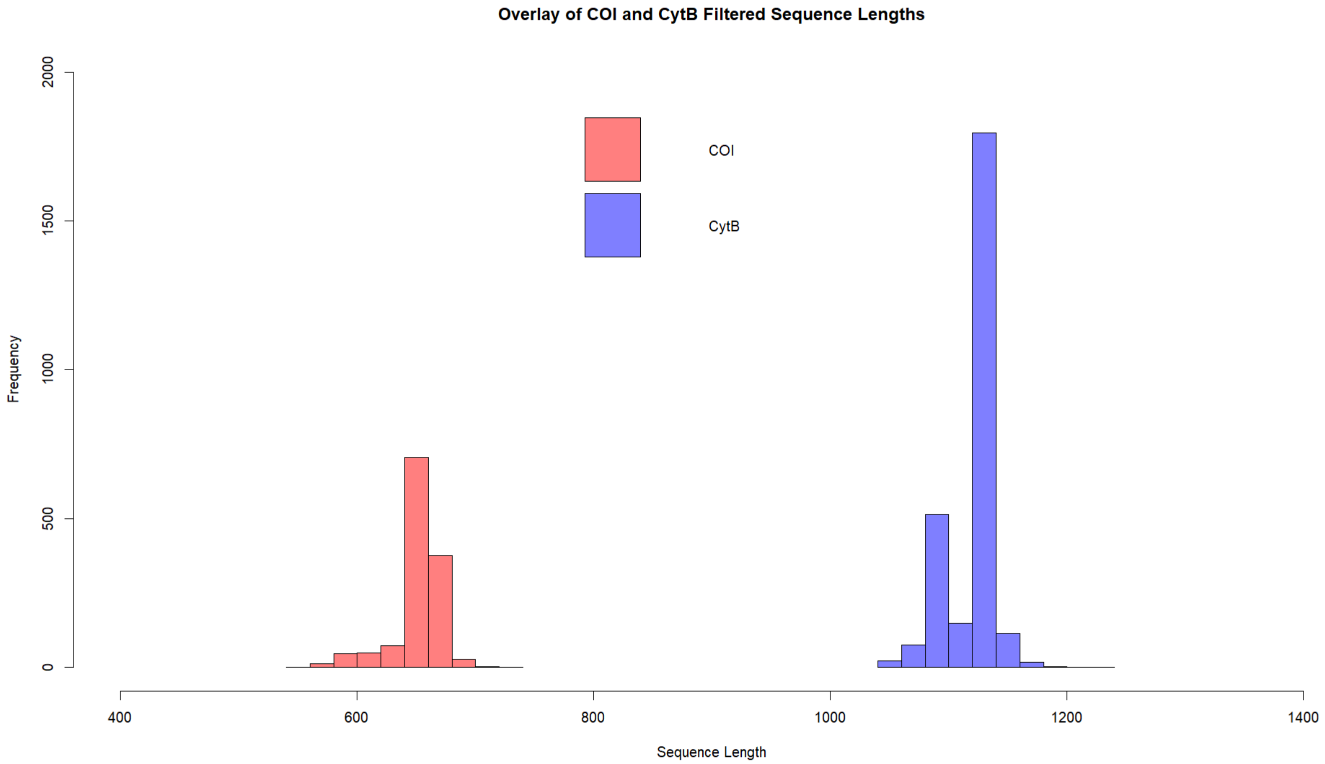
As AI advances, machine learning shows immense promise across industries, particularly in healthcare, where it has the potential to transform genetic analysis. Machine learning models for gene sequence classification are becoming essential in clinical settings, enabling faster, more accurate diagnoses and guiding treatment decisions (Hamed et al., 2023). By classifying gene sequences efficiently, healthcare providers can rapidly identify bacterial strains for targeted public health responses and personalized care, detect predispositions for hereditary diseases, and predict individual responses to medications based on drug metabolism genes (Habehh & Gohel, 2021). This adaptability enhances patient-centered care and public health strategies.

Therefore, optimizing machine learning models for gene classification is therefore critical to improving accuracy, speed, and interpretability in clinical settings. This study aims to compare two prominent algorithms, K-Nearest Neighbors (KNN) and Random Forest, for classifying COI and CytB genes within the Mus genus, focusing on speed, accuracy, and feature importance. The genus Mus is chosen due to its relevance in human health research, as insights from Mus genetics often translate effectively to human applications, and KNN and Random Forest were chosen as they are commonly applied in gene sequence classification (Hamed et al., 2023).

```r
library(rentrez)
library(stringr)
library(dplyr)
library(caret)
library(ggplot2)
library(data.table)
library(Biostrings)
library(readxl)

## 1. Data Acquisition ----

# Set 'retmax' to 0 to retrieve total hit counts for COI and CytB genes in
Mus genus, allowing for subsequent fetching of all sequences
# coi_sequences <- entrez_search(db="nuccore", term='"mus"[Genus] AND
"coi"[Gene]', retmax = 0)
# max_coi_hits <- coi_sequences$count
# cytb_sequences <- entrez_search(db="nuccore", term='"mus"[Genus] AND
"cytB"[Gene]', retmax = 0)
# max_cytb_hits <- cytb_sequences$count

# coi_sequences <- entrez_search(db="nuccore", term='"mus"[Genus] AND
"coi"[Gene] AND 500:800[SLEN]', retmax = max_coi_hits, use_history = TRUE)
# cytb_sequences <- entrez_search(db="nuccore", term='"mus"[Genus] AND
"cytB"[Gene] AND 1000:1250[SLEN]', retmax = max_cytb_hits, use_history =
TRUE)


# Output some general information of what our data looks like for both genes
# cat("Total COI sequences found:", coi_sequences$count, "\n")
# cat("Sample COI sequence IDs:", head(coi_sequences$ids, 5), "\n")

# cat("Total CYTB sequences found:", cytb_sequences$count, "\n")
# cat("Sample CYTB sequence IDs:", head(cytb_sequences$ids, 5), "\n")


# Obtain sequence data in FASTA format
# batch_size <- 500 #Batch size is chosen to be 500 to efficiently retrieve
data in manageable chunks without overloading the server

# Create function which Fetches sequence data by looping through batches,
retrieving sequence info, and accumulation in FASTA format
# fetch_data <- function(entrez_search_results, batch_size) {
# search_results_total_count <- entrez_search_results$count
# complete_sequence_data <- c()

# for (start in seq(0, search_results_total_count, by = batch_size)) {
# batch <- entrez_fetch(db = "nuccore",
# web_history = entrez_search_results$web_history,
# rettype = "fasta",
# retmax = batch_size,
# retstart = start)
# complete_sequence_data <- c(complete_sequence_data, batch)
# }

# return(complete_sequence_data)
# }

# Pass in coi and cytb search results to fetch_data function in order to
output a fasta of all sequences to later be converted to biostirngs
# coi_fasta_seqs <- fetch_data(coi_sequences, batch_size)
# write(coi_fasta_seqs, "coi_sequences.fasta", sep = "\n")

# cytb_fasta_seqs <- fetch_data(cytb_sequences, batch_size)
# write(cytb_fasta_seqs, "cytb_sequences.fasta", sep = "\n")


# Convert FASTA to StringSet for subsequent transformation into a DataFrame.
coi_stringSet <- readDNAStringSet("../data/coi_sequences.fasta")
cytb_stringSet <- readDNAStringSet("../data/cytb_sequences.fasta")

# Convert sequence data into a DataFrame for easier data
filtering/manipulation and analysis for both COI and CytB genes
dfCOI <- data.frame(COI_Title = names(coi_stringSet), COI_Sequence =
paste(coi_stringSet))
dfCytB <- data.frame(CytB_Title = names(cytb_stringSet), CytB_Sequence =
paste(cytb_stringSet))

## 2. Data Exploration ----

explore_data <- function(df) {
  cat("Dimensions:", dim(df), "\n")
  cat("Total number of NA values in sequence column:",
sum(is.na(df[[grep("_Sequence$", names(df))]])), "\n")
  cat("Total number of gaps in sequence columns:",
sum(str_count(df[[grep("_Sequence$", names(df))]], "-")), "\n")

  # Summary of sequence data
  cat("Summary of sequence data:\n")
  print(summary(str_count(df[[grep("_Sequence$", names(df))]]))) # This
prints the typical summary stats

  total_A <- sum(str_count(df[[grep("_Sequence$", names(df))]], "A"))
  total_T <- sum(str_count(df[[grep("_Sequence$", names(df))]], "T"))
  total_C <- sum(str_count(df[[grep("_Sequence$", names(df))]], "C"))
  total_G <- sum(str_count(df[[grep("_Sequence$", names(df))]], "G"))
  total_N <- sum(str_count(df[[grep("_Sequence$", names(df))]], "N"))
  total_nucleotides <- total_A + total_T + total_C + total_G + total_N

  cat("Proportion of A:", round((total_A / total_nucleotides * 100), 2), "%",
"\n")
```

```
  cat("Proportion of T:", round((total_T / total_nucleotides * 100), 2), "%",
"\n")
  cat("Proportion of C:", round((total_C / total_nucleotides * 100), 2), "%",
"\n")
  cat("Proportion of G:", round((total_G / total_nucleotides * 100), 2), "%",
"\n")
  cat("Proportion of N:", round((total_N / total_nucleotides * 100), 4), "%",
"\n")
}

explore_data(dfCOI)
explore_data(dfCytB)

## 3. Data Filtering ----

missing_data <- 0.01
length_var <- 100

# Create new filtered COI and CytB datasets
dfCOI_filtered <- dfCOI %>%
  mutate(Filtered_COI_Sequence = str_remove_all(COI_Sequence, "^N+|N+$|-"))
%>% # remove Ns at start or end of sequence to homogenise data
  filter(str_count(Filtered_COI_Sequence, "N") <= (missing_data *
str_count(Filtered_COI_Sequence))) %>% # To ensure high quality data, remove
any sequences with over 1% of ambiguous nucleotides
  filter(str_count(Filtered_COI_Sequence) >=
median(str_count(Filtered_COI_Sequence)) - length_var &
    str_count(Filtered_COI_Sequence) <=
median(str_count(Filtered_COI_Sequence)) + length_var) %>%
  select(COI_Title, Filtered_COI_Sequence) # Remove sequences that are not
close to the median to homogenise data

dfCytB_filtered <- dfCytB %>%
  mutate(Filtered_CytB_Sequence = str_remove_all(CytB_Sequence, "^N+|N+$|-"))
%>%
  filter(str_count(Filtered_CytB_Sequence, "N") <= (missing_data *
str_count(CytB_Sequence))) %>%
  filter(str_count(Filtered_CytB_Sequence) >=
median(str_count(Filtered_CytB_Sequence)) - length_var &
str_count(Filtered_CytB_Sequence) <=
median(str_count(Filtered_CytB_Sequence)) + length_var) %>%
  select(CytB_Title, Filtered_CytB_Sequence)

explore_data(dfCOI_filtered)
explore_data(dfCytB_filtered)

# 4. Sequence Length Visualization and K-Mer Frequency Visualization ----
# Create a histogram to showcase the distribution of filtered sequence
lengths for COI and CytB. This will show the variation between sequence
lengths in COI and CytB, in order to get a better grasp of the lengths of
```

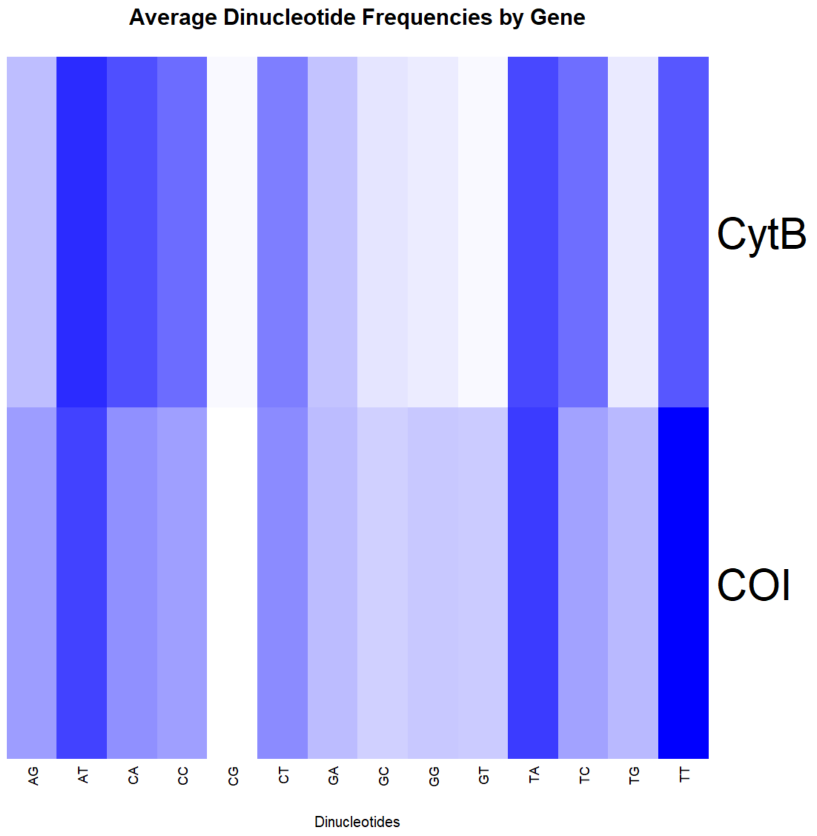
```
sequence data we are working with as well as showcasing a visual distinction
between the sequence lengths of the different genes. The outputted figure
shows two normal distributions around the median sequence length around each
gene type, with them being distant from each other on the x axis, indicating
the differences in sequence length between the two genes.

seq_length_COI_filtered <- nchar(dfCOI_filtered$Filtered_COI_Sequence)
seq_length_CytB_filtered <- nchar(dfCytB_filtered$Filtered_CytB_Sequence)

hist(seq_length_COI_filtered,
  xlab = "Sequence Length",
  ylab = "Frequency",
  main = "Overlay of COI and CytB Filtered Sequence Lengths",
  col = rgb(1, 0, 0, 0.5),
  breaks = "Sturges",
  xlim = c(400, 1400), # Extend x-axis to accomodate sequence length of both
COI and CytB
  ylim = c(0, 2000)
) # Extend y-axis to accomodate sequence length of CytB

# Overlay CytB histogram
hist(seq_length_CytB_filtered,
  col = rgb(0, 0, 1, 0.5),
  breaks = "Sturges",
  add = TRUE
)

legend("top",
  legend = c("COI", "CytB"),
  fill = c(rgb(1, 0, 0, 0.5), rgb(0, 0, 1, 0.5)),
  cex = 1,
  pt.cex = 4,
  bty = "n",
  x.intersp = 1,
  y.intersp = 0.6
)
```



```
# Create a heatmap to show dinucleotide frequency distribution comparing COI
and CytB averages. The result of this heatmap indicated the differences in
dinucelotide proportions between COI and CytB in terms of dinucleotide
frequency, which is an important metric that can be used whne training
classifiers downstream.


dfCOI_filtered$Filtered_COI_Sequence <-
DNAStringSet(dfCOI_filtered$Filtered_COI_Sequence)
dfCytB_filtered$Filtered_CytB_Sequence <-
DNAStringSet(dfCytB_filtered$Filtered_CytB_Sequence)

# Obtain the letter frequency for both sequences
dfCOI_filtered <- cbind(dfCOI_filtered,
as.data.frame(letterFrequency(dfCOI_filtered$Filtered_COI_Sequence, letters =
c("A", "C", "G", "T"))))
dfCytB_filtered <- cbind(dfCytB_filtered,
as.data.frame(letterFrequency(dfCytB_filtered$Filtered_CytB_Sequence, letters
= c("A", "C", "G", "T"))))

# Attain the proportions of each nucleotide per sequence to later attain
dinucleotide proportion, remove unneeded 'total' column
dfCOI_filtered <- dfCOI_filtered %>%
  mutate(
    total = A + T + C + G,
```

```r
    Aprop = A / total,
    Cprop = C / total,
    Tprop = T / total,
    Gprop = G / total,
  ) %>%
  select(-total)

dfCytB_filtered <- dfCytB_filtered %>%
  mutate(
    total = A + T + C + G,
    Aprop = A / total,
    Cprop = C / total,
    Tprop = T / total,
    Gprop = G / total,
  ) %>%
  select(-total)

# Calculate dinucleotide frequency for each sequence for CytB and COI
dfCOI_filtered <- cbind(dfCOI_filtered,
as.data.frame(dinucleotideFrequency(dfCOI_filtered$Filtered_COI_Sequence,
as.prob = TRUE)))

dfCytB_filtered <- cbind(dfCytB_filtered,
as.data.frame(dinucleotideFrequency(dfCytB_filtered$Filtered_CytB_Sequence,
as.prob = TRUE)))


# Obtain the average dinucleotide frequencies for COI and CytB and merge them
into one dataframe to make the heatmap
dinucleotide_columns <- ncol(dfCytB_filtered) - 16:ncol(dfCytB_filtered)

# Calculate the averages for each dinucleotide for COI and CytB gene
average_dinucleotide_proportions_COI <- colMeans(dfCOI_filtered[,
13:ncol(dfCOI_filtered)], na.rm = TRUE)
average_dinucleotide_proportions_COI_df <-
as.data.frame(t(average_dinucleotide_proportions_COI)) %>%
  mutate(Gene = "COI") %>%
  select(Gene, everything())

average_dinucleotide_proportions_CytB <- colMeans(dfCytB_filtered[,
13:ncol(dfCytB_filtered)], na.rm = TRUE)
average_dinucleotide_proportions_CytB_df <-
as.data.frame(t(average_dinucleotide_proportions_CytB)) %>%
  mutate(Gene = "CytB") %>%
  select(Gene, everything())

combined_average_CytB_COI <- rbind(average_dinucleotide_proportions_COI_df,
average_dinucleotide_proportions_CytB_df)

# Create the heatmap to visualize the differences in dinucleotide proportions
```

```
between COI and CytB
rownames(combined_average_CytB_COI) <- combined_average_CytB_COI$Gene
combined_average_CytB_COI <- combined_average_CytB_COI[, -1]
matrix_heatmap <- as.matrix(combined_average_CytB_COI)

heatmap(matrix_heatmap,
  Rowv = NA, Colv = NA, scale = "none",
  col = colorRampPalette(c("white", "blue"))(100),
  xlab = "Dinucleotides", main = "Average Dinucleotide Frequencies by Gene"
)
```



```
5. Train, test and compare classifiers ----

# Convert sequence data back into character data in order to more readily use
tidyverse functions
dfCOI_filtered$Filtered_COI_Sequence <-
as.character(dfCOI_filtered$Filtered_COI_Sequence)
dfCytB_filtered$Filtered_CytB_Sequence <-
as.character(dfCytB_filtered$Filtered_CytB_Sequence)

dfCOI_filtered <- as_tibble(dfCOI_filtered)
dfCytB_filtered <- as_tibble(dfCytB_filtered)
```

```r
# create validation which samples 250 sequences, this number was chosen to
follow the conventional 80:20 training/testing split in machine learning
set.seed(727)
dfValidation_COI <- dfCOI_filtered %>%
  sample_n(250) %>%
  mutate(Gene = "COI") %>%
  dplyr::rename(Title = COI_Title, Sequence = Filtered_COI_Sequence)


dfValidation_CytB <- dfCytB_filtered %>%
  sample_n(250) %>%
  mutate(Gene = "CytB") %>%
  dplyr::rename(Title = CytB_Title, Sequence = Filtered_CytB_Sequence)

# Combine both validation datasets into one dataset for easier implementation
into classifier
combined_validation_df <- rbind(dfValidation_COI, dfValidation_CytB)

# create training dataset which samples 950 sequences - this number was
chosen to follow the conventional 80:20 training/testing split in machine
learning
set.seed(277)
dfTraining_COI <- dfCOI_filtered %>%
  filter(!COI_Title %in% dfValidation_COI$COI_Title) %>%
  sample_n(950) %>%
  mutate(Gene = "COI") %>%
  dplyr::rename(Title = COI_Title, Sequence = Filtered_COI_Sequence)
```

```
## Warning: There was 1 warning in `filter()`.
## ℹ In argument: `!COI_Title %in% dfValidation_COI$COI_Title`.
## Caused by warning:
## ! Unknown or uninitialised column: `COI_Title`.
```

```r
dfTraining_CytB <- dfCytB_filtered %>%
  filter(!CytB_Title %in% dfValidation_CytB$CytB_Title) %>%
  sample_n(950) %>%
  mutate(Gene = "CytB") %>%
  dplyr::rename(Title = CytB_Title, Sequence = Filtered_CytB_Sequence)
```

```
## Warning: There was 1 warning in `filter()`.
## ℹ In argument: `!CytB_Title %in% dfValidation_CytB$CytB_Title`.
## Caused by warning:
## ! Unknown or uninitialised column: `CytB_Title`.
```

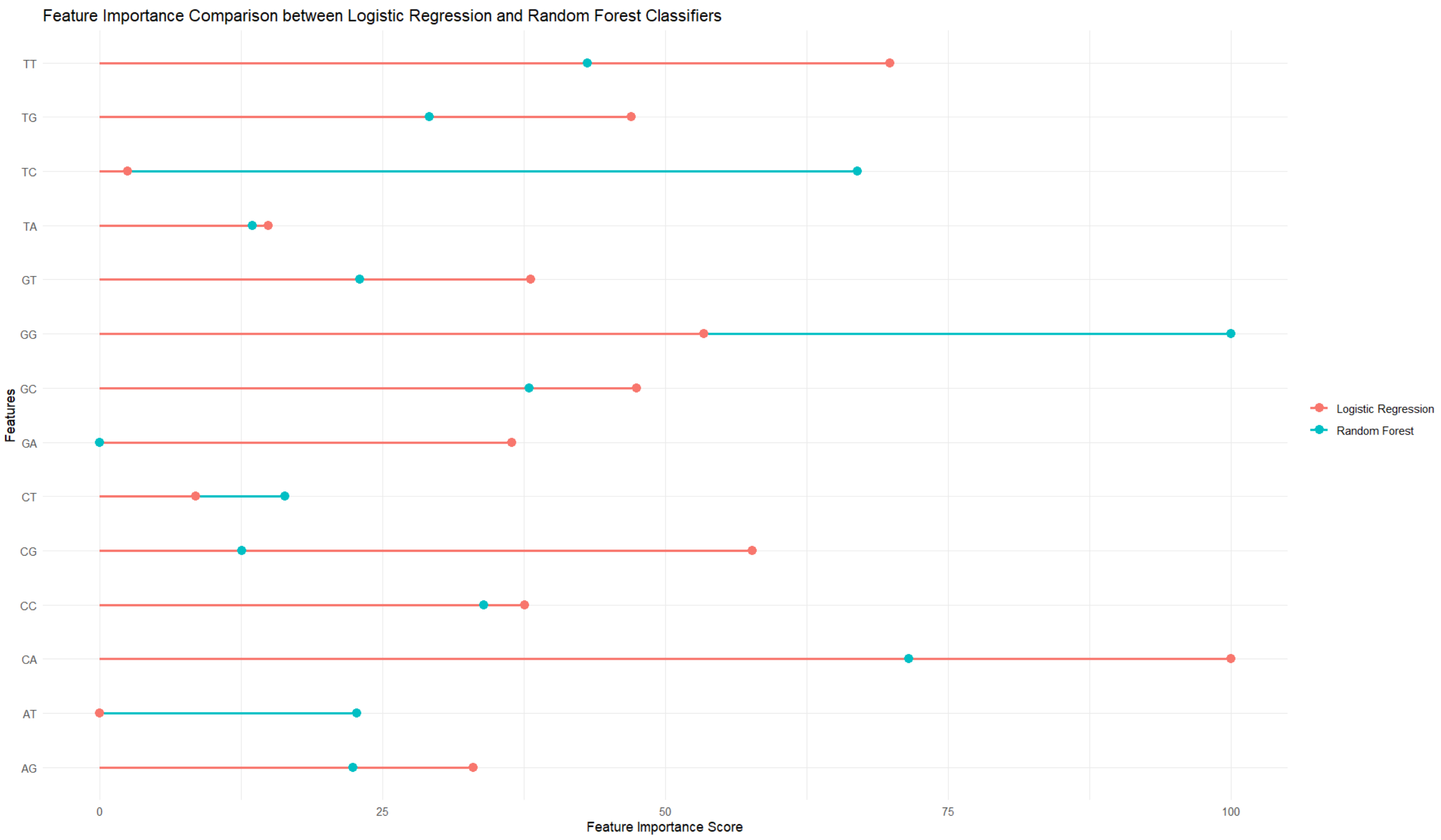
```r
# Combine training datasets for easier implementation into model
combined_training_df <- rbind(dfTraining_COI, dfTraining_CytB)

# Ensure that the Gene column is a factor not a character vector so that it
can be used to to train the model
combined_training_df$Gene <- as.factor(combined_training_df$Gene)
# use the random forest algorithm for training a classifier
```

```r
rf_gene_classifier <- train(Gene ~ .,
  data = combined_training_df[, 13:27],
  method = "rf",
  importance = TRUE,
  trControl = trainControl(method = "cv", number = 10)
)

# using the linear regression algorithm to train a separate classifier
lr_gene_classifier <- train(Gene ~ .,
  data = combined_training_df[, 13:27],
  method = "glm",
  family = "binomial",
  tuneLength = 10
)

rf_gene_classifier

lr_gene_classifier

# Assess the accuracy of the testing dataset with linear regression
predictValidation_lg <- predict(lr_gene_classifier, combined_validation_df[,
c(27, 13:27)])
table(observed = combined_validation_df$Gene, predicted =
predictValidation_rf)
```

```
##          predicted
## observed COI CytB
##     COI  250    0
##     CytB   1  249
```

```r
# 6. Create visualizations to compare feature importance and speed of RF and
KNN classifiers ----

# lollipop plot which shows the feature importance of different dinucleotides
for the random forest and logistic regression algorithm. The different
algorithms assign different importance scores to different dinucleotides due
to how the different algorithms learn from the data and what they prioritize.
combined_validation_df$Gene <- factor(combined_validation_df$Gene, levels =
c("COI", "CytB"))

feature_importance_RF <- varImp(rf_gene_classifier)
feature_importance_LR <- varImp(lr_gene_classifier)

# Convert to data.tables for easier manipulation
imp_RF <- data.table(
  Feature = rownames(feature_importance_RF$importance),
  Importance = feature_importance_RF$importance[, "COI"]
)
imp_LR <- data.table(
  Feature = rownames(feature_importance_LR$importance),
  Importance = feature_importance_LR$importance[, "Overall"]
)

# Add a model identifier
imp_RF[, Model := "Random Forest"]
imp_LR[, Model := "Logistic Regression"]

# Combine the two data.tables
combined_imp <- rbind(imp_RF, imp_LR)

# Create the lollipop plot
ggplot(combined_imp, aes(x = Feature, y = Importance, color = Model)) +
  geom_segment(aes(x = Feature, xend = Feature, y = 0, yend = Importance),
size = 1) +
  geom_point(size = 3) +
  coord_flip() +
  theme_minimal() +
  labs(title = "Feature Importance Comparison between Logistic Regression and
Random Forest Classifiers", x = "Features", y = "Feature Importance Score") +
  theme(legend.title = element_blank())
```

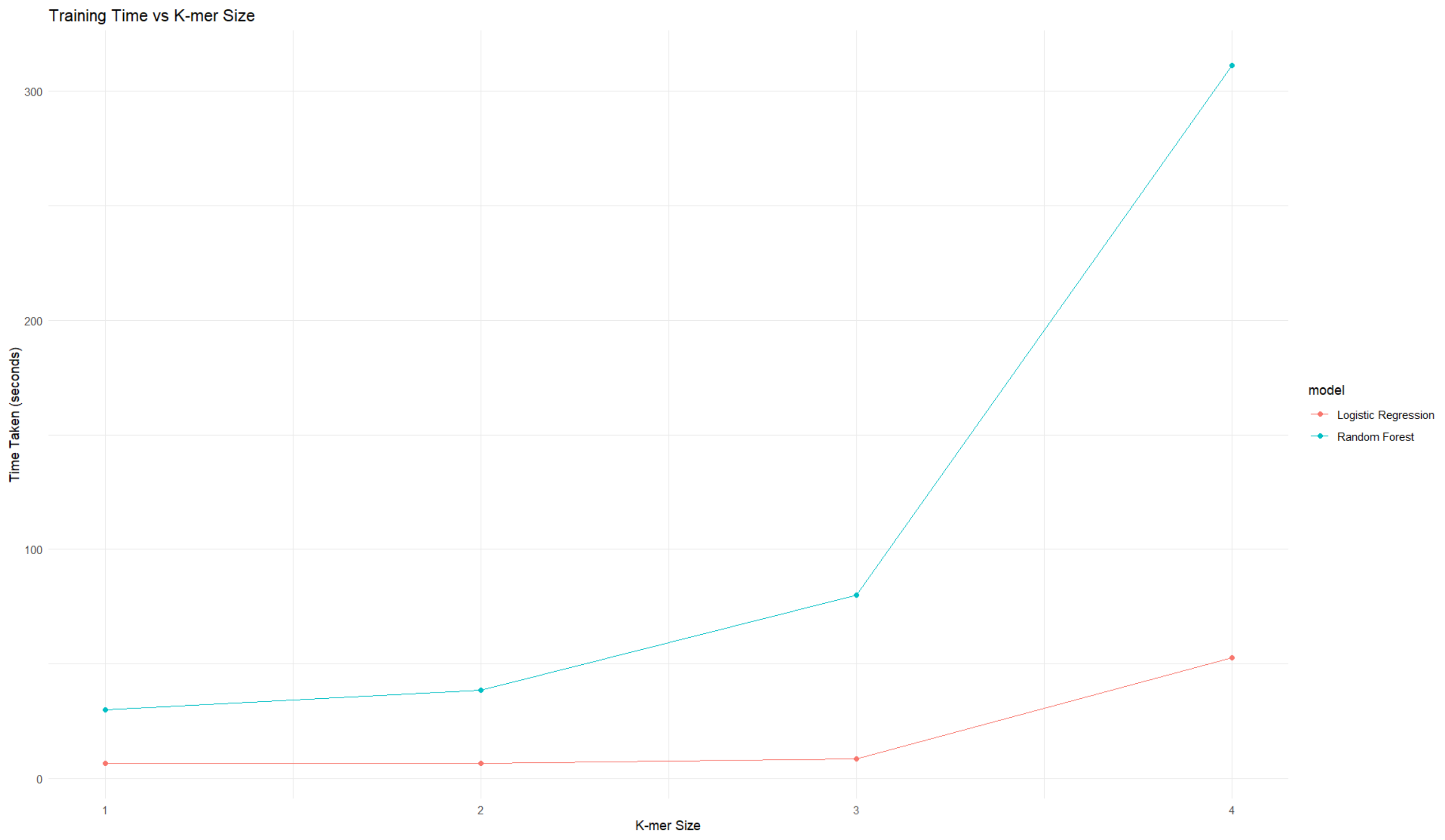
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```r
# K-mer number vs Time elapsed Line Graph - which shows the speed of training
at different k mer sizes for the logistic regression and random forest
algorithms. The random forest algorithm was slower at all k mer sizes,
especially as k mer continued to increase the difference between the two
algorithms in terms of speed grew farther and farther apart. Random forest
seems to be increasingly slower with higher k mer, potentially due to the
greater increase of decision tree complexity and number as k mer size
continues to get larger.

# Simplify training data to only include relevant columns
combined_training_df_simplified <- combined_training_df %>%
  select(Title, Sequence, Gene)

# This function generates k-mers from a DNA sequences and calculates the
frequency of each k-mer
get_kmer_profile <- function(sequence, k) {
  oligonucleotideFrequency(DNAString(sequence), width = k, as.prob = T)
}

# Loop through k-mer sizes 1 to 4 for both linear regression and random
forest algorithms, storing their results in data frames for downstream
visualization

# I have commented the below code because it takes quite a while to run and
come up with the output data table. I have saved the data table in my 'data'
folder as an excel file, so that it is quick and convenient to run without
having to run the for loop

# set.seed(710) # Set global seed for reproducibility

# Initialize empty data frames in order to hold the results for various
metrics
# rf_results <- data.frame(k = integer(), accuracy = numeric(), model =
character(), time.taken.seconds = numeric())
# lr_results <- data.frame(k = integer(), accuracy = numeric(), model =
character(), time.taken.seconds = numeric())

# for (k in seq(1, 4)) {
  # kmer_features <- t(sapply(combined_training_df_simplified$Sequence,
get_kmer_profile, k = k))
  # targets <- combined_training_df_simplified$Gene
  # cross.validation <- trainControl(method = "cv", number = 15) # cross
validating  helps to ensure the model's performance is reliable and reduces
overfitting by evaluating it on multiple, diverse subsets of the data

  # rf_time <- system.time({
    # set.seed(710) # Ensure reproducibility for RF model
    # rf_model <- train(x = kmer_features, y = targets, method = "rf",
trControl = cross.validation)
  # })["elapsed"] # ensure elapsed time is recorded to see how long it took
to train the algorithm
  # rf_results <- rbind(
    # rf_results,
    # data.frame(
      # k = k,
      # accuracy = max(rf_model$results$Accuracy),
      # model = "Random Forest",
      # time.taken.seconds = rf_time
    # )
  # )

  # lr_time <- system.time({
    # set.seed(710) # Ensure reproducibility for LR model
    # lr_model <- train(x = kmer_features, y = targets, method = "glm",
family = binomial, trControl = cross.validation)
  # })["elapsed"]

  # lr_results <- rbind(
    # lr_results,
    # data.frame(
      # k = k,
      # accuracy = max(lr_model$results$Accuracy),
      # model = "Logistic Regression",
      # time.taken.seconds = lr_time
    # )
  # )
# }

# combined_results <- rbind(rf_results, lr_results)

combined_results <- read_excel("../data/kmer_data.xlsx")
```

```
## New names:
## • `` -> `...1`
```

```r
# Plot line graph to showcase the speed of the algorithm based on k-mer
number for RF and LR algorithms
ggplot(combined_results, aes(x = k, y = time.taken.seconds, color = model,
group = model)) +
  geom_line() +
  geom_point() +
  labs(
    title = "Training Time vs K-mer Size",
    x = "K-mer Size",
    y = "Time Taken (seconds)"
  ) +
  theme_minimal()
```



Discussion

After training, testing, and comparing my classifiers, I found that both performed equally well, accurately classifying 99% of the sequences during the testing phase. However, they differed in feature importance: Random Forest assigned the highest importance to the dinucleotide frequencies of GG, TC, and CA, while Logistic Regression prioritized CA, TT, and CG. This similarity in accuracy was expected, given the large, high-quality training dataset with minimal noise, which allowed both algorithms to robustly handle various data types and generalize effectively to unseen data. Although it was initially unexpected, further research into how different models learn from data clarified why feature importance varied between the two. Different classifiers prioritize features differently based on their learning algorithms, which explains the disparities in feature importance between Random Forest and Logistic Regression (Pudjihartono et al., 2022).

Additionally, while I anticipated that an increase in k-mer size would lead to longer training times, I was surprised to find that Random Forest was generally slower across all k-mer sizes. In particular, it took nearly six times longer than Logistic Regression when the k-mer size reached 4. I suspect this is due to the significant scaling of decision trees and their increasing complexity as the k-mer count rises, which contributes to much slower training times. Overall, both models demonstrated solid accuracy; however, the differences in training times became more pronounced as the k-mer size increased. To further develop this project, future research can investigate sequence classifications using more ambiguous and unstructured data (which is commonplace in genomic sequence data), which often contain gaps and ambiguous nucleotides. Future tests with such data will help determine whether these algorithms maintain their accuracy and speed.

Acknowledgements

I would like to thank Thomas Tekle for helping me to refine my question and objective early on, as he encouraged me come up with a simple question and then build on it - whereas before, I was overthinking a more complex question, not even knowing where to start. He saved me a lot of time and effort in this regard. I would also like to thank Isha Baxi who helped to make sure I has all the information regarding the assignment 1 feedback and expectations when I was sick for the week.

References

Habehh, H., & Gohel, S. Machine Learning in Healthcare. Current genomics, 22(4), 291–300 (2021). https://doi.org/10.2174/1389202922666210705124359

Hamed, B.A., Ibrahim, O.A.S. & Abd El-Hafeez, T. Optimizing classification efficiency with machine learning techniques for pattern matching. J Big Data 10, 124 (2023). https://doi.org/10.1186/s40537-023-00804-6

Pudjihartono, N., Fadason, T., Kempa-Liehr, A. W., & O'Sullivan, J. M. A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction. Frontiers in bioinformatics, 2, 927312 (2022). https://doi.org/10.3389/fbinf.2022.927312