Sameh Mohamed – 1301909 - BINF*6210 – 2024/10/25

Application of Random Forest Machine Learning for TCRA and TCRB Gene Classification Using k-mers Frequencies

1. Introduction

Most peripheral T cells utilize T-cell receptor (TCR) molecules consisting of TCRα and TCRβ chains for their specific antigenic recognition [1]. The TCRA and TCRB genes, which encode the TCRα and TCRβ chains, respectively, play a critical role in determining the structure of TCR and how it interacts with its target. TCR is involved in immune response, immunotherapy and various diseases including cancer, immune deficiency, autoimmune diseases and viral infections [2]. Therefore, studying the sequence, structure and mutations of TCRA and TCRB genes is essential. However, due to significant sequence similarity between these genes, differentiation and identification of these genes is challenging.

The Random Forest (RF) machine learning algorithm is widely recognized for its robust performance across various classification tasks, including gene classification [4]. Using k-mers frequency features to train RF-based supervised machine learning models is a well-established approach for genes and species classification [5]. Therefore, this study aims to investigate whether TCRA and TCRB sequences (obtained from NCBI) can be classified using a Random Forest supervised machine learning algorithm trained with various k-mer features and to compare its performance with that of another machine learning algorithm

2. Code - Part 1

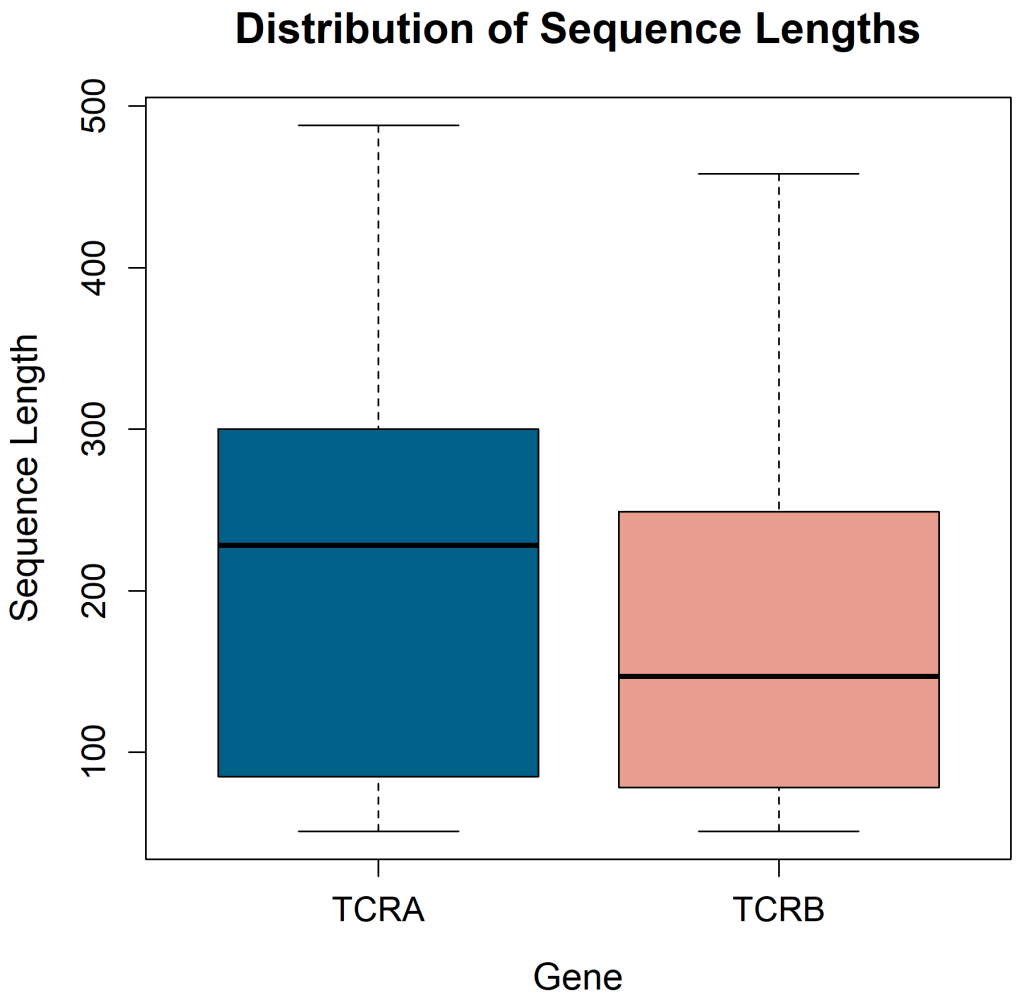
```
# 1- Load necessary libraries ----
library(tidyverse)
library(viridis)
library(rentrez)
library(Biostrings)
library(randomForest)
library(xgboost)
library(pROC)

# 2- Conflict resolution ----
conflicted::conflicts_prefer(dplyr::filter())

# 3- Obtain data from NCBI using Entrez ----
# Uncomment the following lines to obtain data from Entrez
```

```
# TCRA_search = entrez_search(db = "nuccore", term = "human[ORGN] AND
TCRA[gene]")
# TCRA_search$count
# TCRB_search = entrez_search(db = "nuccore", term = "human[ORGN] AND
TCRB[gene]")
# TCRB_search$count
# TCRA_IDs = entrez_search(db = "nuccore", term = "human[ORGN] AND TCRA[gene]",
retmax = TCRA_search$count)
# TCRA_IDs
# TCRB_IDs = entrez_search(db = "nuccore", term = "human[ORGN] AND TCRB[gene]",
use_history = TRUE, retmax = TCRB_search$count)
# TCRB_IDs
# Fetching FASTA files
# TCRA_Fasta = entrez_fetch(db = "nuccore", id = TCRA_IDs$ids, rettype = "fasta")
# TCRA_Fasta
# TCRB_Fasta = entrez_fetch(db = "nuccore", web_history = TCRB_IDs$web_history,
rettype = "fasta")
# TCRB_Fasta
# Save the fetched FASTA files
# write(TCRA_Fasta, "../Data/TCRA_FASTA.fasta", sep = "\n")
# write(TCRB_Fasta, "../Data/TCRB_FASTA.fasta", sep = "\n")
# rm(TCRA_Fasta, TCRB_Fasta, TCRA_IDs, TCRA_search, TCRB_IDs, TCRB_search)

# 4- Load and preprocess TCRA and TCRB data ----
# Read DNA sequences from FASTA files
TCRA_stringSet <- readDNAStringSet("../Data/TCRA_FASTA.fasta")
TCRB_stringSet <- readDNAStringSet("../Data/TCRB_FASTA.fasta")
# Check the class and structure of the data
class(TCRA_stringSet)
class(TCRB_stringSet)
head(TCRA_stringSet)
head(TCRB_stringSet)
# Convert to data frames, Extract the Accession number and Set gene name
df_TCRA <- data.frame(
```

```
  Accession_Number = word(names(TCRA_stringSet), 1L),
  Gene = "TCRA",
  Sequence = paste(TCRA_stringSet)
)
class(df_TCRA) # Check the class of df_TCRA
df_TCRB <- data.frame(
  Accession_Number = word(names(TCRB_stringSet), 1L),
  Gene = "TCRB",
  Sequence = paste(TCRB_stringSet)
)
class(df_TCRB) # Check the class of df_TCRB
# Remove data that will not be used again
rm(TCRA_stringSet, TCRB_stringSet)

# 5- Data filtering and Sequence Quality Control ----
# Boxplot for sequence lengths
boxplot(nchar(df_TCRA$Sequence), nchar(df_TCRB$Sequence),
  names = c("TCRA", "TCRB"), main = "Distribution of Sequence Lengths",
  ylab = "Sequence Length"
)
# According to the Boxplot, data has a few outliers with very large sequence lengths
# Filter out sequences that exceed a reasonable threshold (e.g., sequences longer than 1000
bp)
df_TCRA <- df_TCRA %>% filter(nchar(Sequence) <= 1000)
df_TCRB <- df_TCRB %>% filter(nchar(Sequence) <= 1000)
# Boxplot for sequence lengths
boxplot(nchar(df_TCRA$Sequence), nchar(df_TCRB$Sequence),
  names = c("TCRA", "TCRB"), main = "Distribution of Sequence Lengths",
  ylab = "Sequence Length"
)
# According to second Boxplot, data still has some outliers
# Filter out outliers to retain sequences within a biologically relevant range (50 to 500 bp).
df_TCRA <- df_TCRA %>% filter(nchar(Sequence) < 500 & nchar(Sequence) > 50)
df_TCRB <- df_TCRB %>% filter(nchar(Sequence) < 500 & nchar(Sequence) > 50)
```

```
# Check for gaps or invalid characters in TCRA sequences
any(str_detect(df_TCRA$Sequence, "[-]")) # Check for gaps
any(str_detect(df_TCRA$Sequence, "^[N]")) # Check if 'N' is at the start
any(str_detect(df_TCRA$Sequence, "[N]$")) # Check if 'N' is at the end
any(str_count(df_TCRA$Sequence, "N") >= (0.05 * str_count(df_TCRA$Sequence))) #
Check if 'N' makes up more than 5% of the sequence
# According to previous quality checking steps, in TCRA sequences no gaps were detected
and "N" was only detected at the start
# Filter sequences with 'N' at the start
df_TCRA <- df_TCRA %>% filter(!str_detect(Sequence, "^[N]"))
# Check for gaps or invalid characters in TCRB sequences
any(str_detect(df_TCRB$Sequence, "[-]")) # Check for gaps
any(str_detect(df_TCRB$Sequence, "^[N]")) # Check if 'N' is at the start
any(str_detect(df_TCRB$Sequence, "[N]$")) # Check if 'N' is at the end
any(str_count(df_TCRB$Sequence, "N") >= (0.05 * str_count(df_TCRB$Sequence))) #
Check if 'N' makes up more than 5% of the sequence
# According to previous quality checking steps, in TCRB sequences no gaps were detected
and no sequences was found to start or end with "N" or has 'N' comprising more than 5% of
its total length
# Sample df_TCRB to match number of sequences in df_TCRA
set.seed(1)
df_TCRB <- sample_n(df_TCRB, 341)
# Export the boxplot as a PNG file
png("TCRA_TCRB_Distribution_of_Sequence_Lengths.png", width = 6.3, height = 6.3,
units = "in", res = 300)
# Boxplot for genes sequence lengths after all filtering steps
boxplot(nchar(df_TCRA$Sequence), nchar(df_TCRB$Sequence),
  names = c("TCRA", "TCRB"),
  main = "Distribution of Sequence Lengths",
  ylab = "Sequence Length",
  xlab = "Gene",
  col = c("#006088", "#E99F90"),
  cex.main = 1.5,
  cex.lab = 1.3,
```

```
  cex.axis = 1.2
)
# Close the PNG device
dev.off()
```



3. Code – Part 2

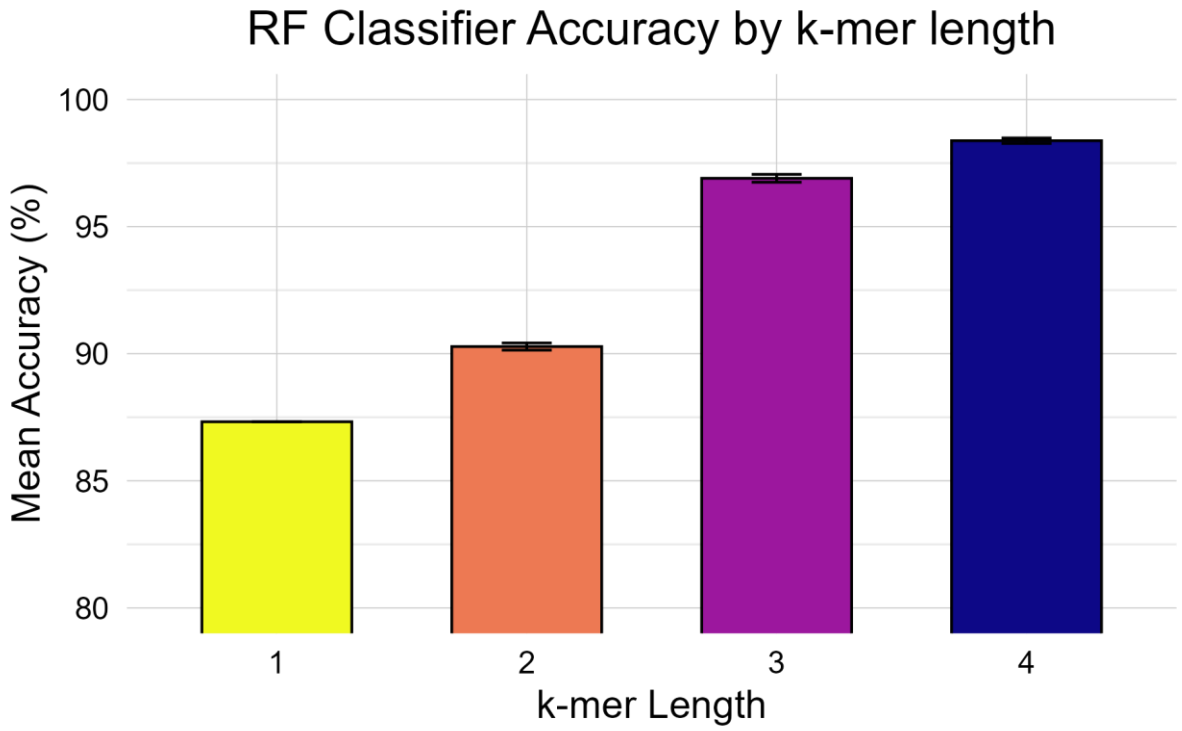
```
# 6- Combine TCRA and TCRB data and Extract nucleotide frequency features----
# Combine TCRA and TCRB data
df_Combined <- rbind(df_TCRA, df_TCRB)
# Check the content of new data
table(df_Combined$Gene)
# Remove data that will not be used again
rm(df_TCRA, df_TCRB)
# Calculate single nucleotide proportions (A, T, G, C)
df_Combined$Sequence <- DNAStringSet(df_Combined$Sequence)
```

```
class(df_Combined)
df_Combined <- cbind(
  df_Combined,
  as.data.frame(letterFrequency(df_Combined$Sequence, letters = c("A", "C", "G", "T")))
)
df_Combined$Aprop <- (df_Combined$A) / (df_Combined$A + df_Combined$T +
df_Combined$C + df_Combined$G)
df_Combined$Tprop <- (df_Combined$T) / (df_Combined$A + df_Combined$T +
df_Combined$C + df_Combined$G)
df_Combined$Gprop <- (df_Combined$G) / (df_Combined$A + df_Combined$T +
df_Combined$C + df_Combined$G)
# Calculate di-,tri- and oligonucleotide frequencies
df_Combined <- cbind(
  df_Combined,
  as.data.frame(dinucleotideFrequency(df_Combined$Sequence, as.prob = TRUE)),
  as.data.frame(trinucleotideFrequency(df_Combined$Sequence, as.prob = TRUE)),
  as.data.frame(oligonucleotideFrequency(df_Combined$Sequence, width = 4, as.prob =
TRUE))
)
# Convert sequence back to character
df_Combined$Sequence <- as.character(df_Combined$Sequence)

# 7- Split Data into Training and Validation sets ----
# Training data set will include about 80% of the available sequences and the remaining 20%
will be used in validation
set.seed(1)
df_Training <- df_Combined %>%
  group_by(Gene) %>%
  sample_n(270)
table(df_Training$Gene)
df_Validation <- df_Combined %>%
  filter(!Accession_Number %in% df_Training$Accession_Number)
table(df_Validation$Gene)
```

```
# 8- Random Forest Classifiers for Different k-mers ----
# Define column ranges for each k-mer
kmer_ranges <- list("1" = 8:10, "2" = 11:26, "3" = 27:90, "4" = 91:346)
n_iterations <- 10
# Function to run Random Forest classifiers for different k-mer
run_classifier <- function(kmer_columns) {
  kmer_accuracy_results <- numeric(n_iterations) # Create a numeric vector to store accuracy
values for each iteration.
  for (i in 1:n_iterations) {
    set.seed(i) # Set a seed for each iteration
    # Train the Random Forest classifier
    kmer_classifier <- randomForest(
      x = df_Training[, kmer_columns],
      y = as.factor(df_Training$Gene),
      ntree = 500
    )
    # Predict and calculate accuracy
    kmer_validation <- predict(kmer_classifier, df_Validation[, kmer_columns])
    kmer_confusion_matrix <- table(df_Validation$Gene, kmer_validation)
    kmer_accuracy_results[i] <- sum(diag(kmer_confusion_matrix)) /
sum(kmer_confusion_matrix) * 100
  }
  return(kmer_accuracy_results)
}
# Run classifiers and calculate accuracies for each k-mer
kmer_classifier_results <- lapply(kmer_ranges, run_classifier)
# Calculate the mean accuracy for each k-mer
kmer_classifier_mean_accuracy <- sapply(kmer_classifier_results, mean)
# Calculate the standard error (SE) of accuracy for each k-mer
kmer_classifier_SE <- sapply(kmer_classifier_results, function(res) sd(res) /
sqrt(n_iterations))
# Print results for each k-mer classifier
for (kmer_name in names(kmer_classifier_mean_accuracy)) {
  cat(paste(
```

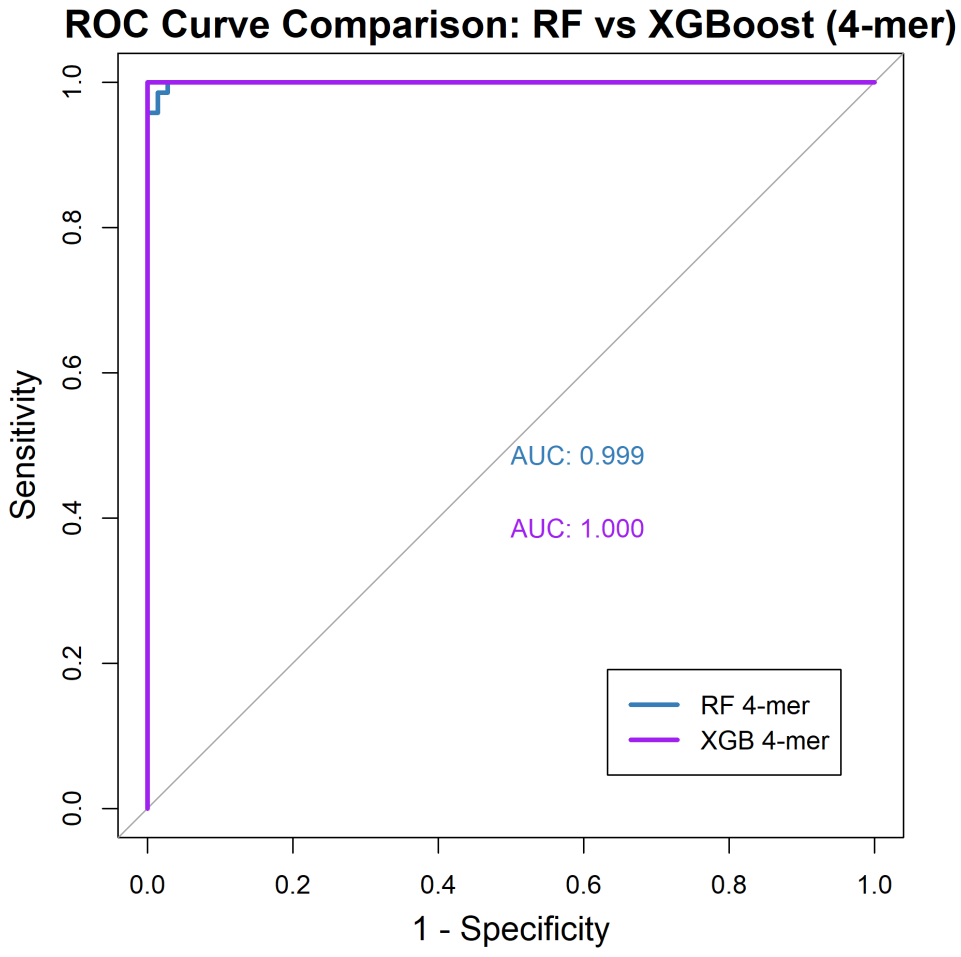
```
    kmer_name, ": Mean accuracy =", round(kmer_classifier_mean_accuracy[kmer_name], 2),
    ", SE =", round(kmer_classifier_SE[kmer_name], 2), "\n"
  ))
}
# ==> Results
# 1 : Mean accuracy = 87.32 , SE = 0
# 2 : Mean accuracy = 90.28 , SE = 0.14
# 3 : Mean accuracy = 96.9 , SE = 0.16
# 4 : Mean accuracy = 98.38 , SE = 0.11
# Prepare data for ANOVA
kmer_accuracy_data <- data.frame(
  Kmer = rep(names(kmer_classifier_results), each = n_iterations),
  Accuracy = unlist(kmer_classifier_results)
)
# Perform ANOVA
kmer_anova_result <- aov(Accuracy ~ Kmer, data = kmer_accuracy_data)
summary(kmer_anova_result)
# ==> One-way ANOVA results
# p-value = <2e-16
# ANOVA results suggest a highly significant effect of the Kmer factor on classifier accuracy.
# Perform post-hoc comparisons - Tukey's HSD
kmer_tukey_result <- TukeyHSD(kmer_anova_result)
print(kmer_tukey_result)
# ==> Tukey's HSD results
# All pairwise comparisons are statistically significant because all the adjusted p-values are 0
# Thus, there are significant differences between all pairs of k-mer sizes.

# 9- Visualize Classifier Accuracy by K-mer length ----
# Create a data frame for plotting
kmer_plot_data <- data.frame(
  Kmer = factor(names(kmer_classifier_mean_accuracy), levels =
names(kmer_classifier_mean_accuracy)),
  Mean = kmer_classifier_mean_accuracy,
  SE = kmer_classifier_SE
```

```
)
# Plotting the data
ggplot(kmer_plot_data, aes(x = Kmer, y = Mean, fill = Kmer)) +
  geom_bar(stat = "identity", width = 0.6, color = "black") +
  geom_errorbar(aes(ymin = Mean - SE, ymax = Mean + SE), width = 0.2, color = "black") +
  labs(title = "RF Classifier Accuracy by k-mer length", x = "k-mer Length", y = "Mean
Accuracy (%)") +
  scale_fill_viridis_d(option = "plasma", direction = -1) +
  theme_minimal(base_size = 15) +
  theme(
    legend.position = "none",
    plot.title = element_text(hjust = 0.5),
    axis.text = element_text(color = "black"),
    axis.title = element_text(color = "black"),
    panel.grid.major = element_line(size = 0.2, color = "gray80")
  ) +
  coord_cartesian(ylim = c(80, 100))# Exporting the Plot with size suitable for A4 paper
ggsave("RF Classifier Accuracy by k-mer length.PNG", width = 6.4, height = 4)
```



```
rm(kmer_accuracy_data, kmer_anova_result, kmer_classifier_mean_accuracy,
kmer_classifier_results, kmer_classifier_SE, kmer_name, kmer_ranges, kmer_tukey_result,
kmer_plot_data, n_iterations)

# 10- Train Random Forest and XGBoost Classifiers using 4-mers and calculate roc curves ---
-
# Train Random Forest 4-mer Classifier
set.seed(1)
RF_Classifier_4_mer <- randomForest(
  x = df_Training[, 91:346],
  y = as.factor(df_Training$Gene),
  ntree = 500,
  importance = TRUE
)
# Get predicted probabilities for RF classifier on validation set
RF_Validation_4_mer_probs <- predict(RF_Classifier_4_mer, df_Validation[, 91:346], type
= "prob")
# Train XGBoost 4-mer Classifier
set.seed(1) # To ensure consistency in results
X_train <- as.matrix(df_Training[, 91:346]) # Convert data to matrix as needed for XGBoost
y_train <- as.numeric(as.factor(df_Training$Gene)) - 1 # Converting the gene label into a
binary numerical factor
X_valid <- as.matrix(df_Validation[, 91:346])
y_valid <- as.numeric(as.factor(df_Validation$Gene)) - 1
# Train XGBoost model with 500 boosting rounds to generate probabilities
XGB_model <- xgboost(data = X_train, label = y_train, nrounds = 500, objective =
"binary:logistic", verbose = 0)
# Get predicted probabilities for XGBoost classifier on validation set
XGB_predictions_probs <- predict(XGB_model, X_valid)
# Open a PNG file
png("ROC Curve - Comparison of RF and XGBoost 4mer.png", width = 6.3, height = 6.3,
units = "in", res = 300)
# Calculate ROC for Random Forest 4-mer model
roc(df_Validation$Gene, RF_Validation_4_mer_probs[, "TCRA"],
```

```
  plot = TRUE,
  legacy.axes = TRUE, lwd = 3, print.auc = TRUE, col = "#377eb8", cex.lab = 1.3
)
# Calculate ROC for XGBoost 4-mer model
roc(y_valid, XGB_predictions_probs,
  plot = TRUE, add = TRUE,
  legacy.axes = TRUE, lwd = 3, print.auc = TRUE, print.auc.y = 0.4, col = "purple"
)
title(main = "ROC Curve Comparison: RF vs XGBoost (4-mer)", line = 2.5, cex.main = 1.5)
# Add legend
legend("bottomright",
  legend = c("RF 4-mer", "XGB 4-mer"),
  col = c("#377eb8", "purple"), lwd = 3,
  inset = 0.08
)# Close the PNG device
dev.off()
```



4. Discussion and conclusion

Aiming to design a supervised RF machine learning model with high accuracy, approximately 80% of the available sequences were allocated for training the model while the remaining 20% were used in model performance validation. Different RF models were trained using different k-mers (k = 1, 2, 3, or 4) features. For each model, accuracy was computed 10 times to account for variability in the model's performance, then the average accuracy and the standard error were calculated. Interestingly, ANOVA and Tukey's HSD tests showed that accuracy significantly increased with longer k-mers, reaching a maximum of 98.38% with 4-mers, a pattern also observed in previous models [6]. Finally, 4-mer oligonucleotide frequencies were used to train RF and XGBoost models to compare their performance in TCRA and TCRB classification. Both models had comparable performance, achieving nearly identical ROC-AUC values on the validation data

In conclusion, the RF algorithm trained with k-mer features effectively classifies genes with high sequence similarity, such as TCRA and TCRB. However, since both RF and XGBoost models trained with 4-mers demonstrated similar performance, it suggests that the choice of training features may play a more decisive role than the machine learning algorithm itself. The main limitation of this study was the limited number of available sequences for the TCRA gene. It is recommended to test this model on a larger number of sequences and with other genes that have closely related sequences.

5. Acknowledgement

I would like to extend my heartfelt gratitude to Dylan Harding, Avery Murphy, Dhruv Mishra and Moiz Syed for their invaluable support throughout the assignment. Their assistance in brainstorming ideas, suggesting new packages and codes, and running my script to ensure it was error-free was fundamental in completing this assignment.

6. References

1. Obata F, Tsunoda M, Kaneko T, et al (1993) Human T-cell receptor TCRAV, TCRBV, and TCRAJ sequences newly found in T-cell clones reactive with allogeneic HLA class II antigens

2. Shah K, Al-Haidari A, Sun J, Kazi JU (2021) T cell receptor (TCR) signaling in health and disease. Signal Transduct Target Ther 6

3. Breiman L (2001) Random forests. Mach Learn 45:5–32. https://doi.org/10.1023/A:1010933404324/METRICS

4. Azar AT, Elshazly HI, Hassanien AE, Elkorany AM (2014) A random forest classifier for lymph diseases. Comput Methods Programs Biomed 113:465–473. https://doi.org/10.1016/J.CMPB.2013.11.004

5. Meher PK, Sahu TK, Rao AR (2016) Identification of species based on DNA barcode using k-mer feature vector and Random Forest classifier. Gene 592:316–324. https://doi.org/10.1016/J.GENE.2016.07.010

6. Meher PK, Sahu TK, Rao AR (2016) Identification of species based on DNA barcode using k-mer feature vector and Random Forest classifier. Gene 591:316–324. https://doi.org/10.1016/j.gene.2016.07.010

7. XGBoost in R: A Step-by-Step Example. Available from: https://www.statology.org/xgboost-in-r/

8. ROC and AUC in R. Available from: https://youtu.be/qcvAqAH60Yw