Math 895: Research Project Report

**Covid in the Genes**

Gabrielle Salamanca

San Francisco State University

December 12, 2025

# **Contents**

1. **Introduction**
2. **Explanation of Data**
3. **Data Analysis**
4. **Model Selection**
   1. Logistic Regression
   2. Linear Discriminant Analysis
   3. Quadratic Discriminant Analysis
   4. K-Nearest Neighbors
   5. Classification Tree
   6. Random Forest
   7. Boosting
5. **Conclusion**
6. **Future Path**
7. **Appendix**
   1. References
   2. Code

# 

# **Introduction**

In Spring 2023, I began working with Professor Tao He as a research assistant on a project motivated by her earlier work applying machine learning methods on next-generation sequencing of T-Cell receptor (TCR) Repertoire data. Her prior research examined whether the VJ genes of T-Cell and B-Cell Receptors can be used to predict cancer types. Building on that idea, this project aimed to investigate whether those genes could distinguish COVID-19 patients from healthy individuals. Professor He provided three datasets: one with the VJ gene usage for COVID-19 patients, one for healthy patients, and a supplementary one with patient-level information.

During my time as her research assistant, our focus was primarily on exploratory data analysis using a new software package and a technique that was unfamiliar to me at the time: Principal Component Analysis (PCA). For the current project, however, I aim to extend this work by formally addressing the predictive question. To do so, I applied methods learned in *Math 448: Introduction to Statistical Learning and Data Mining*, as well as some concepts from *Math 748: Theory and Applications of Statistical Machine Learning* that I had not previously used in practice.

# **Explanation of Data**

As explained in the introduction, there were three datasets provided for the analysis: VJ gene usage for COVID-19 patients (699 rows, 71 columns), VJ gene usage for healthy patients (687 rows, 40 columns), and patient information (92 rows, 8 columns).

During my work as Professor He’s research assistant, the two VJ gene datasets were merged using the shared column vjGene. Several entries in the count or expression value column were missing, and these were replaced with a small constant to avoid computational issues during log-transformation. The rows were renamed according to their corresponding VJ gene, and a logarithmic transformation was applied to the dataset before transposing it. This step produced a matrix where genes formed the columns and patients formed the rows. The log-transformed data was then standardized.

To incorporate patient-level metadata, the row names (patient identifiers) were updated to match those in the patient information dataset, enabling a successful merge. After merging, several columns were removed or rearranged as necessary, and a binary outcome variable Y was created.

The original outcome variable Y1 contained three categories: heathy, recovered, and active. For this project, these were consolidated into two classes: healthy and disease (combining recovered and active). After all preprocessing steps, the dataset contained 109 rows and 679 columns.

During quality checks, I identified missing values in both Y and Y1. After consulting with Professor He, we confirmed that patients labeled “HD” should be treated as healthy. Cross-referencing with the patient information dataset verified this interpretation. One row could not be matched to any entry in the patient information file and was therefore removed. The final cleaned dataset consists of 108 patients and 679 columns, 676 of which are genes.

**Table 1: Dataset Representation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sample.ID** | **TRBV10-1\_TRBJ1-1** | **…** | **Y** | **Y1** |
| 1\_1 | -1.3458166 | … | disease | active |
| 1\_2 | -1.3458166 | … | disease | active |
| . . . | .  .  . | … | .  .  . | .  .  . |

# **Data Analysis**

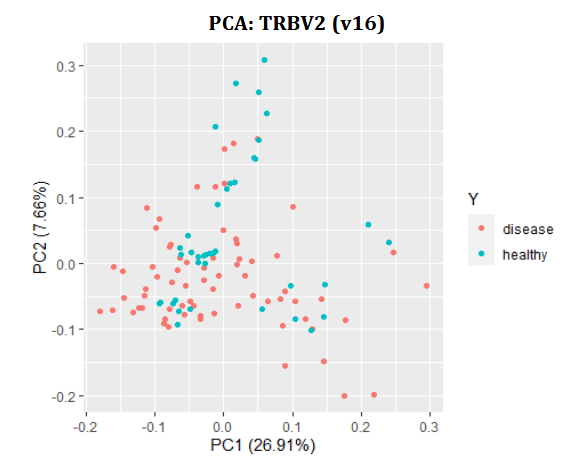
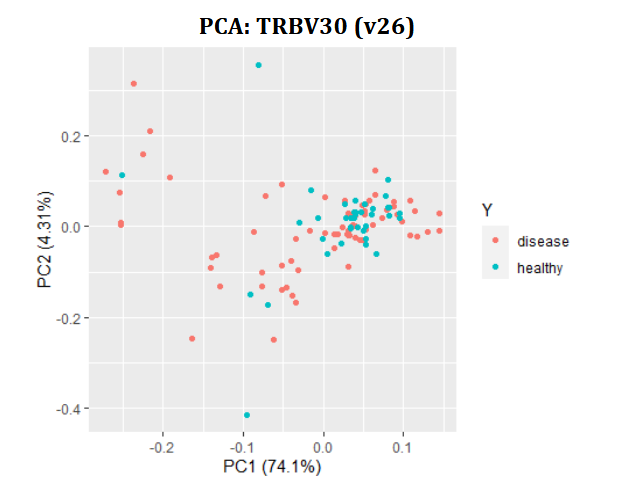
During my time as Professor He’s research assistant, our initial analysis focused on exploring the data using the SKAT package (Sequence Kernel Association Test). It’s a package used for genetic analysis, particularly for rare variant association analysis. We used the SKAT\_Null\_Model function to compute model parameters and residuals, and the SKATBinary function to compute p-values of Burden test, SKAT, and SKAT-O for binary traits using asymptotic and efficient resampling methods. These tests were applied separately to the v and j gene sets to identify the significant ones before and after p-value adjustments.

Following the association testing, Principal Component Analysis was performed. PCA is an unsupervised learning technique commonly used for data preprocessing, dimension reduction, and exploratory analysis. It identifies principal components– linear combinations of the original features– that capture the maximum variance in the dataset. By projecting the data onto these components, PCA reduces dimensionality while preserving as much of the original structure as possible.

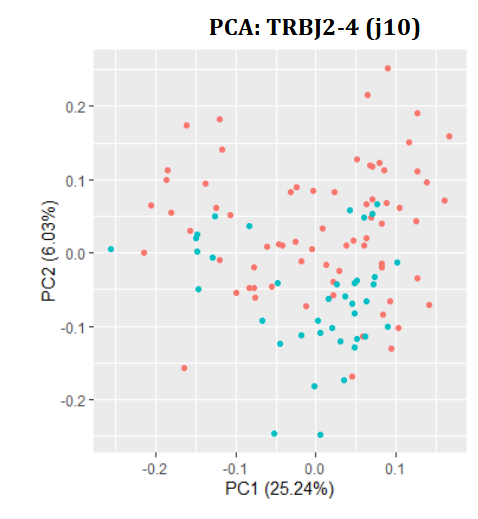
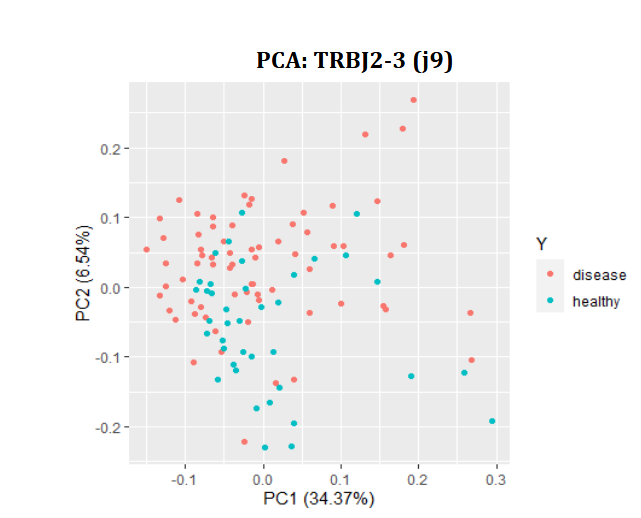
At the time, this technique was performed on the entire dataset without splitting the data into training and test sets. Both outcome variables, Y and Y1, were used for separate PCA analyses on the V and J genes, and selected subsets of genes. For the purposes of this project, I will only focus on the PCA results using the binary outcome variable Y.

Overall, most genes identified as significant through SKAT did not explain a large proportion of the variance. For almost all PCA plots, the first principal component explained less than 35% of the variance, and the second component explained roughly 10% or less. However, one gene was an exception, explaining approximately 74% of the variance in the first principal component and under 5% in the second.

**Figure 1: v Gene PCA Plot Comparison**

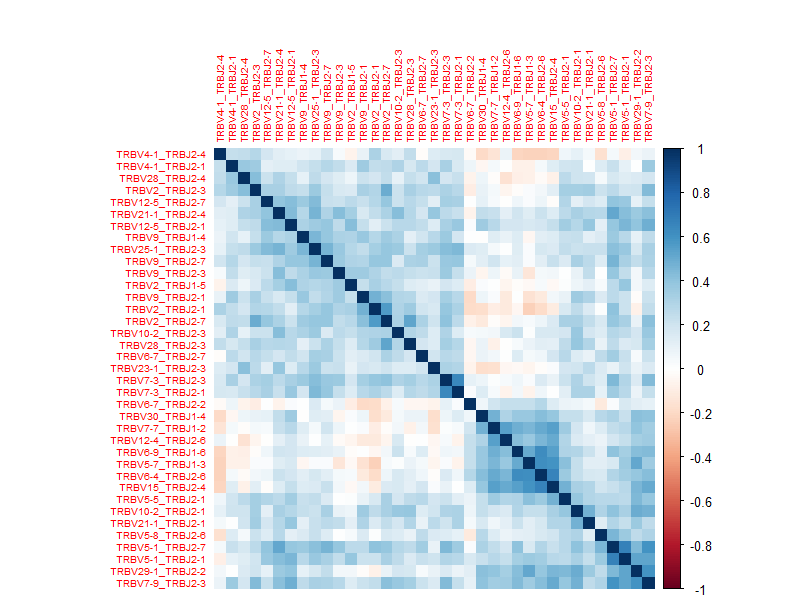


**Figure 2: j Gene PCA Plot Comparison**

****

Across all plots, no distinct clusters emerged, the classes looking highly entangled. There are a few factors that may explain this: the dataset perhaps is high-dimensional and complex, with variance spread across many components; the features may not be extremely correlated; or the biological or technical noise is high. So, Professor He recommended generating a correlation plot of the significant genes.

**Figure 3: Correlation plot of the Significant Genes**



The correlation plot shows mostly weak to moderate correlations among the genes, mostly positive. This pattern helps explain why the earlier PCA plots lacked clear group separation– the underlying features do not form strongly correlated structures that would allow PCA to capture disease-related variation in the first few components.

# **Model Selection**

## Logistic Regression

Logistic regression is a supervised machine learning algorithm commonly used for classification problems. It models the probability that an input belongs to a particular class using a logistic (sigmoid) function. Because the response variable Y in this project has two possible outcomes– healthy or disease– we use the binomial form of logistic regression, which is the standard approach for binary classification.

Before applying any predictive models, it was necessary to identify the most informative features among the 676 genes. To do this, a univariate logistic regression model was fit for each gene separately using a for-loop. For each model, the p-value associated with the gene’s coefficient was extracted and stored. After sorting the results by significance, genes with p-values below 0.05 were retained. Out of the 676 genes, 37 were identified as statistically significant.

Each predictive method evaluated in this project will use these 37 selected genes, but in incrementally increasing batches. Specifically, the first iteration uses the top 5, the next uses the top 10, and the process continues in increments of 5 until the final iterations, which includes the remaining 2 genes for a total of 37. When logistic regression was applied to these subsets of features, the results were as follows:

**Table 2: Logistic Regression Test Accuracy Results**

|  |  |
| --- | --- |
| **Groups** | **Test Accuracy** |
| Top 5 | 0.6818 |
| Top 10 | 0.7273 |
| Top 15 | 0.6818 |
| Top 20 | 0.8182 |
| Top 25 | 0.7727 |
| Top 30 | 0.7727 |
| Top 35 | 0.7727 |
| Top 37 | 0.7727 |

During this step, two warnings were encountered while fitting the logistic regression models: the algorithm didn’t converge, where fitted probabilities numerically 0 or 1 occurred; and there were non-integer #successses in binomial glm.

The first warning indicates that the iterative optimization procedure failed to obtain stable coefficient estimates. The presence of fitted probabilities extremely close to 0 or 1 suggests complete or quasi-complete separation, a condition where a predictor perfectly, or nearly perfectly, separates the two classes. This leads to numerical instability because this method relies on the log-odds transformation, and values approaching 0 or 1 can push coefficients toward infinity.

The second warning typically arises when the response variable isn’t coded as strict 0/1 integers. Non-integer values, improperly coerced factors, or accidental inclusion of probabilities instead of class labels can trigger this message. Although the response variable was converted to numeric labels (0 for healthy and 1 for disease) and then factored, it’s possible that an unintended coercion or structural issue in the dataset led to this warning.

In the first warning, the algorithm not converging means that the iterative optimization procedure failed to find stable coefficient estimates, and the fitted probabilities mean that for some observations, the model predicts 1 or 0 for the probability. Those values in the second part cause numerical instability, because logistic regression uses log-odds. Therefore, or is problematic or infinite. Finally, the non-integer warning means that Y contains non-integers such as: numbers like 0.3, probabilities instead of binary values, and factors incorrectly converted to numeric.

To address the first issue, a separation-safe GLM was used. This type of logistic regression is specifically designed to handle complete or quasi-complete separation, which is common in high-dimensional biological datasets where individual features may strongly, or even perfectly, distinguish between outcome groups. The separation-safe GLM successfully resolved the convergence-related warnings. However, it didn’t eliminate the second warning concerning non-integer successes, suggesting the issue arose from the underlying structure or encoding of the response variable rather than the modeling procedure itself.

## Linear Discriminant Analysis

Although Quadratic Discriminant Analysis (QDA) was implemented earlier in my workflow, for clarity I present Linear Discriminant Analysis (LDA) first as QDA builds directly on the assumptions of this method.

Linear Discriminant Analysis is a classification method that models each class as arising from a multivariate normal distribution with a shared covariance matrix. Under this assumption, LDA produces a linear decision boundary that approximates the Bayes Classifier. Because LDA requires estimating only a single covariance matrix rather than one per class, it’s more stable than QDA, especially when sample sizes are limited.

**Table 3: LDA Test Accuracy Results**

|  |  |
| --- | --- |
| **Groups** | **Test Accuracy** |
| Top 5 | 0.6818 |
| Top 10 | 0.7727 |
| Top 15 | 0.7727 |
| Top 20 | 0.7727 |
| Top 25 | 0.8182 |
| Top 30 | 0.7727 |
| Top 35 | 0.7727 |
| Top 37 | 0.7273 |

## Quadratic Discriminant Analysis

Quadratic Discriminant Analysis, also known as QDA, is a classification method closely related to LDA. Like LDA, QDA assumes that the predictors within each class are drawn from a multivariate normal distribution. However, it relaxes one key assumption: the requirement of classes sharing a common covariance matrix. Here, the method estimates a separate covariance matrix for each class, which allows for quadratic decision boundaries rather than linear.

Because QDA retains the normality assumption, the presence of categorical predictors or non-normally distributed features may violate this assumption. In practice, it’s known to be more flexible but also more data-hungry than LDA, since estimating a full covariance matrix for each class requires a sufficient number of samples relative to the number of features.

During implementation, the following warning occurred: some group too small for QDA. This warning indicates that at least one outcome group doesn’t have enough observations to estimate its covariance matrix. When this condition isn’t met, the covariance matrix becomes singular (non-invertible), and the method can’t compute the discriminant function for that group. As a result, the final three feature subsets failed to produce valid test accuracies, because QDA couldn’t be fit for those cases.

**Table 4: QDA Test Accuracy Results**

|  |  |
| --- | --- |
| **Groups** | **Test Accuracy** |
| Top 5 | 0.6818 |
| Top 10 | 0.7273 |
| Top 15 | 0.7273 |
| Top 20 | 0.7273 |
| Top 25 | 0.6364 |
| Top 30 | NA |
| Top 35 | NA |
| Top 37 | NA |

## K-Nearest Neighbor

K-Nearest Neighbors, also known as KNN, is a non-parametric classification method that predicts the class of a test observation based on the Euclidean distance to its nearest neighbors in the training set. Rather than estimating model parameters, KNN directly approximates the Bayes classifier by assigning the test point to the most common class among its K closest training observations.

Although this method can achieve strong classification performance, it doesn’t provide information about the feature importance or which variables contribute most to the classification. The primary tuning parameter is K, the number of neighbors considered. KNN is a highly flexible method, and its flexibility increases as K decreases. Small values of K can capture complex, nonlinear boundaries, but may also be more sensitive to noise.

For this project, three different values were evaluated: , , and . This allowed a performance comparison across varying levels of model flexibility.

**Table 5: KNN Test Accuracy Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Groups** | **Test Accuracy:** | **Test Accuracy:** | **Test Accuracy:** |
| Top 5 | 0.7273 | 0.7273 | 0.7273 |
| Top 10 | 0.7273 | 0.7727 | 0.7727 |
| Top 15 | 0.8182 | 0.7727 | 0.7727 |
| Top 20 | 0.7727 | 0.7727 | 0.7273 |
| Top 25 | 0.8182 | 0.7727 | 0.7273 |
| Top 30 | 0.7727 | 0.7727 | 0.8182 |
| Top 35 | 0.8182 | 0.7727 | 0.8182 |
| Top 37 | 0.7273 | 0.7727 | 0.8636 |

## Classification Tree

A classification tree is a predictive modeling technique that recursively partitions the feature space to classify observations into distinct groups. At each step, the algorithm selects a feature and a corresponding split that best separates the classes, typically using measures such as Gini impurity or entropy. The resulting model is represented as a tree structure: internal nodes correspond to decision rules on specific features, branches represent the outcomes of those decisions, and leaf nodes assign the final predicted class. Classification trees are intuitive, easy to interpret, and capable of capturing nonlinear relationships between predictors and the response

**Table 6: Classification Tree Test Accuracy Results**

|  |  |
| --- | --- |
| **Groups** | **Test Accuracy** |
| Top 5 | 0.6364 |
| Top 10 | 0.6818 |
| Top 15 | 0.6818 |
| Top 20 | 0.7273 |
| Top 25 | 0.6818 |
| Top 30 | 0.5909 |
| Top 35 | 0.6364 |
| Top 37 | 0.7273 |

## Random Forest

Random forest is a powerful ensemble learning method that constructs a large collection of decision trees, each trained on a randomly selected subset of the data and a randomly chosen subset of features. This randomness reduces overfitting, enabling the model to capture complex nonlinear relationships and improving the robustness in the presence of noise or many predictors. However, these models are less interpretable than logistic regression or a single decision tree, and they may be computationally slower on very large datasets.

In the table below, two additional metrics are reported: best mtry and OOB error. Best mtry refers to the optimal number of features randomly selected at each split. If it’s small, the randomness increases among trees, and it can lower overfitting. If it’s large, then randomness decreases and may improve performance, but it increases the risk of overfitting. OOB error is out of bag error, which is an internal estimate of test error. It’s computed using only the observations not included in each bootstrap sample. This serves as a built-in form of cross-validation, allowing us to tune mtry without a separate validation set.

Here’s an OOB error range as a general guideline: less than 5% means there’s excellent separation between classes and strong predictive features, between 5 and 15% means a good performance with a clear signal but imperfect class separation, and between 15 and 25% means a moderate performance, suggesting overlapping classes or noisy predictors.

**Table 7: Random Forest Test Accuracy Results**

|  |  |  |  |
| --- | --- | --- | --- |
| **Groups** | **Test Accuracy** | **Best mtry** | **OOB Error** |
| Top 5 | 0.7273 | 2 | 0.2442 |
| Top 10 | 0.7727 | 6 | 0.1512 |
| Top 15 | 0.8182 | 2 | 0.1628 |
| Top 20 | 0.7273 | 4 | 0.1744 |
| Top 25 | 0.7273 | 8 | 0.1512 |
| Top 30 | 0.8182 | 30 | 0.1628 |
| Top 35 | 0.8182 | 6 | 0.1512 |
| Top 37 | 0.7727 | 6 | 0.1744 |

## Boosting

Boosting is a supervised machine learning technique that builds a strong predictive model by combining the outputs of many weak learners, typically shallow decision trees. Unlike bagging methods such as Random Forest, this method trains models sequentially. Each new model is fit to emphasize observations that were misclassified in previous iterations, allowing the algorithm to iteratively correct its mistakes and improve overall performance. Through this process, boosting can capture complex patterns and achieve high predictive accuracy.

In the table below, two important tuning parameters are reported: shrinkage and depth. Shrinkage is known as the learning rate, meaning it controls the contribution of each tree to the final model. A smaller value will slow down the learning process, which reduces the risk of overfitting. This is especially beneficial in noisy biological datasets. Although it will require more trees to achieve good performance, they tend to produce more stable and generalizable models. Larger values allow the model to adapt more quickly, but may increase the risk of overfitting. Depth meanwhile sets the maximum number of splits in each individual tree. Shallow trees create smoother decision boundaries and promote better generalization, but deeper trees can capture more complex interactions among variables are more prone to overfitting, particularly when the sample size is small.

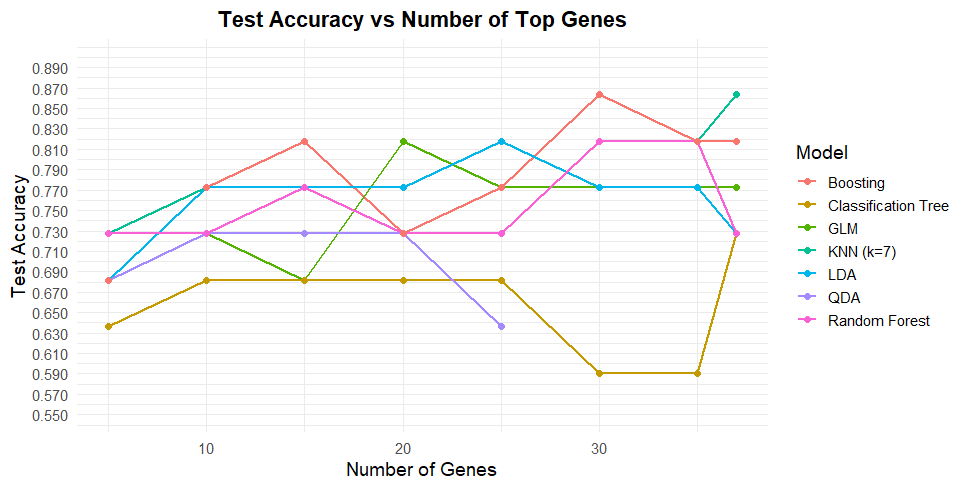
**Table 8: Boosting Test Accuracy Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Groups** | **Test Accuracy** | **Shrinkage** | **Depth** | **Trees** |
| Top 5 | 0.6818 | 0.01 | 3 | 352 |
| Top 10 | 0.6818 | 0.1 | 4 | 29 |
| Top 15 | 0.8182 | 0.05 | 1 | 152 |
| Top 20 | 0.7273 | 0.01 | 1 | 737 |
| Top 25 | 0.7273 | 0.01 | 1 | 942 |
| Top 30 | 0.8182 | 0.05 | 1 | 147 |
| Top 35 | 0.8182 | 0.1 | 2 | 94 |
| Top 37 | 0.8182 | 0.01 | 1 | 1358 |

# 

# **Conclusion**

**Figure 4: Test Accuracy vs Number of Genes for all Models**



Across all models, test accuracies ranged from nearly 60% to nearly 90%, with several methods performing particularly well. Below are is a table with each method and their best test accuracy.

**Table 9: Method Rankings**

|  |  |
| --- | --- |
| **Method** | **Test Accuracy** |
| KNN (k=7) | 0.8636 |
| Log Regression | 0.8182 |
| LDA | 0.8182 |
| KNN (k=3) | 0.8182 |
| Boosting | 0.8182 |
| Random Forest | 0.8182 |
| KNN (k=5) | 0.7727 |
| Classification Tree | 0.7273 |
| QDA | 0.7273 |

Out of the nine approaches evaluated, KNN where achieved the highest test accuracy at 86.36%, indicating that a moderately smoothed neighborhood-based classifier was the most effective for this dataset. Several other methods– including logistic regression, LDA, KNN with , boosting, and random forest– performed consistently, each reaching a test accuracy of 81.82%. Meanwhile, QDA had the weakest performance overall, not only producing the lowest accuracy but also failing to complete the final iterations of the feature subsets due to covariance matrix estimation issues, resulting in missing test accuracy values for the last three groups.

Even though some methods performed fairly well, there are several limitations that make this a challenging dataset. This dataset is extremely high-dimensional, with only 108 samples and over 600 gene features. The classes also don’t separate cleanly in PCA, meaning the biological signal is relatively weak compared to noise. This helps explain why simpler models like logistic regression and LDA performed similarly to more flexible ones, and why QDA had trouble completing the analysis.

Despite these challenges, the results show that the VJ genes do contain enough information to reach moderate to excellent predictive accuracy. However, the data’s complexity and dimensionality make classification difficult. The best performing method indicates that moderately flexible models can capture patterns in the data better than both very simple and very flexible alternatives.

# **Future Path**

There are several directions that can be taken to continue and improve this project. As mentioned, one of the main challenges with the current dataset is the size of it. It has a small sample size and an immense number of gene features. Future work could focus on reducing the dimensionality of the data before applying classification methods. Techniques such as PCA, feature filtering, or selecting gene groups instead of individual genes may help create more stable models and improve performance.

Another possible direction is exploring the regularized models such as LASSO or elastic net logistic regression. These methods are specifically designed for high-dimensional settings and could help identify a smaller set of genes that contribute the most to classification accuracy. More advanced machine-learning approaches could also be considered, such as other ensemble methods or testing kernel-based methods.

Finally, increasing the number of samples, especially disease cases, would allow for more reliable model training and testing. A larger dataset would help reduce the issues seen in QDA and improve the generalizability of all methods. If additional metadata or clinical information becomes available, combining gene usage with other features could also strengthen predictive performance. However, I recognize that this may not be possible, especially due to ethical considerations around patient data collection.

Overall, there is substantial room to refine the modeling pipeline, incorporate more advanced statistical learning techniques, and explore additional biological insights from the vj gene data

# **Appendix**

## References

Air Force Institute of Technology. 2018. Random forests. (May 2018). Retrieved December 8,

2025 from <https://afit-r.github.io/random_forests>.

IBM. 2025. What is boosting? (November 2025). Retrieved December 8, 2025 from

<https://www.ibm.com/think/topics/boosting>.

IBM. 2025. What is Principal Component Analysis (PCA)? (November 2025). Retrieved

December 7, 2025 from <https://www.ibm.com/think/topics/principal-component-analysis>.

Kartik. 2025. Logistic regression in machine learning. (November 2025). Retrieved December 8,

2025 from

<https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression/>.

Seunggeun Lee , Zhangchen Zhao, Larisa Miropolsky , and Michael Wu. 2016. Software: SNU

GSDS Lee Lab. (August 2016). Retrieved December 7, 2025 from

https://www.leelabsg.org/software.

Tao He. 2025. A Comparative Study of Machine Learning Methods on Next-Generation

Sequencing of T-Cell Receptor Repertoire Data. *Math Colloquium* (2025).

Tao He. 2025. Unsupervised Learning. *Math 748: Theory and Applications of Statistical*

*Machine Learning* (2025).

## Code: CE TCR Basics

knitr::opts\_chunk$set(echo = FALSE)  
# data manipulation  
library(dplyr)  
library(psych)  
library(readr)  
library(readxl)  
library(tidyr)  
  
# visualize  
library(factoextra)  
library(ggfortify)  
library(ggplot2)  
library(gridExtra)  
  
# techniques  
library(brglm2) # bias-reduced log regress  
library(caret) # data partition, CV  
library(class) # KNN   
library(gbm) # boosting  
library(MASS) # QDA & LDA  
library(randomForest) # bag, rand forest  
library(rpart) # class tree  
library(stats)  
library(tree) # regression tree  
gene <- read\_excel("D:/Coding/R Storage/Summer TCR Project/TCR Datasets/2025/fullgenes.xlsx")  
  
cat("The dimensions of the dataset is:")  
dim(gene)  
  
cat("\nAre there any NAs in the Y column?")  
table(is.na(gene$Y))  
cat("\nAre there NAs in the Y1 column?")  
table(is.na(gene$Y1))  
  
cat("\nWe remove row 22, to then have the dimensions:")  
genedit <- gene[-22,]  
dim(genedit)  
  
cat("\n")  
  
for (lab in intersect(c("Y","Y1"), names(genedit))) {  
 genedit[[lab]][is.na(genedit[[lab]])] <- "healthy"  
}  
  
cat("After replacing NAs with the healthy tag, are there any lingering NAs in the Y column?")  
table(is.na(genedit$Y))  
cat("\nin the Y1 column?")  
table(is.na(genedit$Y1))  
set.seed(895)  
  
train <- sample(1:nrow(genedit),0.8\*nrow(genedit))  
train.data <- genedit[train,]  
test.data <- genedit[-train,]  
  
cat("The dimensions of the training set is:", dim(train.data), "\n")  
cat("The dimensions of the test is:", dim(test.data), "\n")  
  
# turn Y binary  
train.data$Y <- as.numeric(ifelse(train.data$Y == "disease", 1, 0))  
test.data$Y <- as.numeric(ifelse(test.data$Y == "disease", 1, 0))  
col <- ncol(train.data)  
ycol <- match("Y", names(train.data))  
gene\_idx <- 2:(col - 2)  
gene.name <- names(train.data)[gene\_idx]  
pvalue <- numeric(length(gene\_idx))  
for (i in seq\_along(gene\_idx))  
{  
 gene\_name <- gene.name[i]  
 Xi <- train.data[, gene\_idx[i], drop = FALSE]  
 names(Xi) <- gene\_name # set column name to the gene  
   
 dat <- data.frame(Y = train.data[[ycol]], Xi,   
 check.names = FALSE)  
 glm.fit <- glm(Y ~ ., data = dat, family = binomial())  
 pvalue[i] <- coef(summary(glm.fit))[2, 4]  
}  
# Combine results into a nice table:  
results <- data.frame(Gene = gene.name, P\_value = pvalue)  
  
# Sort by significance  
results <- results[order(results$P\_value), ]  
head(results)  
alpha <- results[results$P\_value < 0.05,]  
cat("The dimensions of the significant genes (alpha) dataset is:")  
dim(alpha)  
  
ranked <- alpha$Gene  
ranked <- intersect(ranked, intersect(names(train.data), names(test.data)))  
train.data$Y <- factor(train.data$Y, levels = c(0,1))  
test.data$Y <- factor(test.data$Y, levels = levels(train.data$Y))  
  
# steps  
max\_k <- min(37L, length(ranked))  
if (max\_k < 5L) max\_k <- length(ranked)  
steps <- seq(5L, max\_k, by = 5L)  
if (tail(steps, 1) != max\_k) steps <- c(steps, max\_k)  
  
# helper  
impute\_from\_train <- function(Xtr, Xte) {  
 for (nm in colnames(Xtr)) {  
 med <- suppressWarnings(median(Xtr[[nm]], na.rm = TRUE))  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
 list(Xtr = Xtr, Xte = Xte)  
}  
# storage  
tree\_curve <- data.frame(TopGenes = integer(), Test\_Error = numeric())  
for (k in steps) {  
 genes\_k <- ranked[1:k]  
 genes\_k <- intersect(genes\_k, intersect(names(train.data), names(test.data)))  
 if (!length(genes\_k)) {  
 tree\_curve <- rbind(tree\_curve, data.frame(TopGenes = k, Test\_Acc = NA\_real\_))  
 next  
 }  
  
 # 1) Extract predictors  
 Xtr <- as.data.frame(train.data[, genes\_k, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_k, drop = FALSE])  
  
 # 2) Impute TEST from TRAIN medians  
 for (nm in colnames(Xtr)) {  
 med <- suppressWarnings(median(Xtr[[nm]], na.rm = TRUE))  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
  
 # 3) ONE canonical name map; apply to BOTH sets BEFORE building data frames  
 safe\_names <- make.names(genes\_k, unique = TRUE)  
 colnames(Xtr) <- safe\_names  
 colnames(Xte) <- safe\_names  
  
 # 4) Modeling frames (same columns & order)  
 dat\_tr <- data.frame(Y = factor(train.data$Y, levels = c(0,1)),  
 Xtr[, safe\_names, drop = FALSE],  
 check.names = FALSE)  
 dat\_te <- data.frame(Y = factor(test.data$Y, levels = c(0,1)),  
 Xte[, safe\_names, drop = FALSE],  
 check.names = FALSE)  
  
 # 5) Formula from the SAME names (avoid paste() pitfalls)  
 form\_tree <- reformulate(termlabels = safe\_names, response = "Y")  
  
 # 6) Fit + prune  
 fit <- rpart(form\_tree, data = dat\_tr, method = "class",  
 control = rpart.control(cp = 0.001, minsplit = 10, xval = 10))  
  
 cpt <- fit$cptable  
 imin <- which.min(replace(cpt[, "xerror"], is.na(cpt[, "xerror"]), Inf))  
 cp\_opt <- cpt[imin, "CP"]  
 fit\_pruned <- prune(fit, cp = cp\_opt)  
  
 # 7) Align TEST to what the model expects (names & order)  
 vars\_needed <- attr(fit\_pruned$terms, "term.labels") # should equal safe\_names  
 # quick guard: must be empty  
 if (length(setdiff(vars\_needed, names(dat\_te)))) {  
 stop("Missing in test: ", paste(setdiff(vars\_needed, names(dat\_te)), collapse = ", "))  
 }  
 new\_te <- dat\_te[, vars\_needed, drop = FALSE]  
 # ensure identical name vector  
 stopifnot(identical(names(new\_te), vars\_needed))  
  
 # 8) Predict & TEST ACCURACY  
 pred <- predict(fit\_pruned, newdata = new\_te, type = "class")  
 test\_acc <- mean(pred == dat\_te$Y)  
  
 cat("Top", k, "genes → Test Accuracy:", sprintf("%.4f", test\_acc), "\n")  
 tree\_curve <- rbind(tree\_curve, data.frame(TopGenes = k, Test\_Acc = test\_acc))  
}  
randFor <- data.frame(TopGenes = integer(),  
 mtry = integer(),  
 OOB\_Error = numeric(),  
 Test\_Acc = numeric())  
for (k in steps) {  
 genes\_k <- ranked[1:k]  
  
 # Extract predictors  
 Xtr <- as.data.frame(train.data[, genes\_k, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_k, drop = FALSE])  
  
 # Impute NAs (train medians → apply to test)  
 tmp <- impute\_from\_train(Xtr, Xte); Xtr <- tmp$Xtr; Xte <- tmp$Xte  
  
 # Safe, consistent names  
 safe\_names <- make.names(genes\_k, unique = TRUE)  
 colnames(Xtr) <- safe\_names  
 colnames(Xte) <- safe\_names  
  
 dat\_tr <- data.frame(Y = train.data$Y, Xtr, check.names = FALSE)  
 dat\_te <- data.frame(Y = test.data$Y, Xte, check.names = FALSE)  
  
 # ----- Auto-tune mtry via OOB error -----  
 p <- length(safe\_names)  
 # candidate grid (clipped to [1, p], unique integers)  
 grid\_raw <- c(sqrt(p)/2, sqrt(p), 2\*sqrt(p), p/3, p/2, p)  
 mtry\_grid <- sort(unique(pmax(1L, pmin(p, round(grid\_raw)))))  
  
 best\_oob <- Inf  
 best\_fit <- NULL  
 best\_mtry <- NA\_integer\_  
  
 form\_rf <- reformulate(termlabels = safe\_names, response = "Y")  
  
 for (m in mtry\_grid) {  
 rf.fit <- randomForest(form\_rf, data = dat\_tr,  
 mtry = m, ntree = 500,  
 nodesize = 1, importance = FALSE)  
 # OOB error from the running err.rate table  
 oob\_err <- tail(rf.fit$err.rate[, "OOB"], 1)  
 if (!is.finite(oob\_err)) {  
 # rare fallback if OOB is NA  
 oob\_err <- mean(rf.fit$y != rf.fit$predicted, na.rm = TRUE)  
 }  
 if (oob\_err < best\_oob) {  
 best\_oob <- oob\_err  
 best\_fit <- rf.fit  
 best\_mtry <- m  
 }  
 }  
  
 # Predict on TEST with the best model and compute TEST ACCURACY  
 pred <- predict(best\_fit, newdata = dat\_te, type = "class")  
 test\_acc <- mean(pred == dat\_te$Y)  
  
 # Print minimal info (as requested)  
 cat("Top", k, "genes → Test Accuracy:", sprintf("%.4f", test\_acc),  
 " | best mtry =", best\_mtry, " | OOB Err =", sprintf("%.4f", best\_oob), "\n")  
  
 # Store  
 randFor <- rbind(randFor,  
 data.frame(TopGenes = k,  
 mtry = best\_mtry,  
 OOB\_Error = best\_oob,  
 Test\_Acc = test\_acc))  
}  
# Tuning grids  
shrinkage\_grid <- c(0.1, 0.05, 0.01)  
depth\_grid <- c(1, 2, 3, 4)  
n\_trees\_max <- 3000 # enough for small shrinkage  
cv\_folds <- 5  
# storage  
boost <- data.frame(TopGenes = integer(), shrinkage = numeric(),  
 depth = integer(), best\_iter = integer(),  
 CV\_Deviance = numeric(), Test\_Acc = numeric())  
for (k in steps) {  
  
 genes\_k <- ranked[1:k]  
 # Extract & impute  
 Xtr <- as.data.frame(train.data[, genes\_k, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_k, drop = FALSE])  
 tmp <- impute\_from\_train(Xtr, Xte); Xtr <- tmp$Xtr; Xte <- tmp$Xte  
  
 # Safe names  
 safe\_names <- make.names(genes\_k, unique = TRUE)  
 colnames(Xtr) <- safe\_names  
 colnames(Xte) <- safe\_names  
  
 # gbm wants 0/1 numeric Y  
 dat\_tr <- data.frame(Y = as.integer(train.data$Y) - 1L, Xtr, check.names = FALSE)  
 dat\_te <- data.frame(Y = as.integer(test.data$Y) - 1L, Xte, check.names = FALSE)  
  
 # Auto-tune shrinkage & depth by CV deviance  
 best\_fit <- NULL  
 best\_iter <- NA\_integer\_  
 best\_cv <- Inf  
 best\_pars <- c(shrink = NA\_real\_, depth = NA\_integer\_)  
  
 form\_boost <- reformulate(termlabels = safe\_names, response = "Y")  
  
 for (sh in shrinkage\_grid) {  
 for (dp in depth\_grid) {  
 fit <- gbm(  
 formula = form\_boost, data = dat\_tr, distribution = "bernoulli",  
 n.trees = n\_trees\_max, interaction.depth = dp,  
 shrinkage = sh, n.minobsinnode = 10, bag.fraction = 0.5,  
 cv.folds = cv\_folds, keep.data = FALSE, verbose = FALSE  
 )  
 bi <- gbm.perf(fit, method = "cv", plot.it = FALSE)  
 cv\_min <- suppressWarnings(min(fit$cv.error[is.finite(fit$cv.error)]))  
  
 if (is.finite(cv\_min) && cv\_min < best\_cv) {  
 best\_cv <- cv\_min  
 best\_fit <- fit  
 best\_iter <- bi  
 best\_pars <- c(shrink = sh, depth = dp)  
 }  
 }  
 }  
   
 # Predict TEST with tuned model  
 prob <- predict(best\_fit, newdata = dat\_te, n.trees = best\_iter, type = "response")  
 pred <- ifelse(prob > 0.5, 1L, 0L)  
 test\_acc <- mean(pred == dat\_te$Y)  
  
 cat("Top ", k, " genes → Test Accuracy:", sprintf("%.4f", test\_acc),  
 " | shrinkage = ", best\_pars["shrink"], " depth = ", best\_pars["depth"],  
 " trees = ", best\_iter, "\n", sep="")  
  
 boost <- rbind(boost, data.frame(TopGenes = k,   
 shrinkage = as.numeric(best\_pars["shrink"]),  
 depth = as.integer(best\_pars["depth"]),  
 best\_iter = best\_iter,  
 CV\_Deviance = best\_cv,   
 Test\_Acc = test\_acc)  
 )  
}

## Code: CE TCR Advanced

knitr::opts\_chunk$set(echo = FALSE)  
# data manipulation  
library(dplyr)  
library(psych)  
library(readr)  
library(readxl)  
library(tidyr)  
  
# visualize  
library(factoextra)  
library(ggfortify)  
library(ggplot2)  
library(gridExtra)  
  
# techniques  
library(brglm2) # bias-reduced log regress  
library(caret) # data partition, CV  
library(class) # KNN   
library(gbm) # boosting  
library(MASS) # QDA & LDA  
library(randomForest) # bag, rand forest  
library(rpart) # class tree  
library(stats)  
library(tree) # regression tree  
gene <- read\_excel("D:/Coding/R Storage/Summer TCR Project/TCR Datasets/2025/fullgenes.xlsx")  
  
cat("The dimensions of the dataset is:")  
dim(gene)  
  
cat("\nAre there any NAs in the Y column?")  
table(is.na(gene$Y))  
cat("\nAre there NAs in the Y1 column?")  
table(is.na(gene$Y1))  
  
cat("\nWe remove row 22, to then have the dimensions:")  
genedit <- gene[-22,]  
dim(genedit)  
  
cat("\n")  
  
for (lab in intersect(c("Y","Y1"), names(genedit))) {  
 genedit[[lab]][is.na(genedit[[lab]])] <- "healthy"  
}  
  
cat("After replacing NAs with the healthy tag, are there any lingering NAs in the Y column?")  
table(is.na(genedit$Y))  
cat("\nin the Y1 column?")  
table(is.na(genedit$Y1))  
set.seed(895)  
  
train <- sample(1:nrow(genedit),0.8\*nrow(genedit))  
train.data <- genedit[train,]  
test.data <- genedit[-train,]  
  
cat("The dimensions of the training set is:", dim(train.data), "\n")  
cat("The dimensions of the test is:", dim(test.data), "\n")  
  
# turn Y binary  
train.data$Y <- as.numeric(ifelse(train.data$Y == "disease", 1, 0))  
test.data$Y <- as.numeric(ifelse(test.data$Y == "disease", 1, 0))  
col <- ncol(train.data)  
ycol <- match("Y", names(train.data))  
gene\_idx <- 2:(col - 2)  
gene.name <- names(train.data)[gene\_idx]  
pvalue <- numeric(length(gene\_idx))  
for (i in seq\_along(gene\_idx))  
{  
 gene\_name <- gene.name[i]  
 Xi <- train.data[, gene\_idx[i], drop = FALSE]  
 names(Xi) <- gene\_name # set column name to the gene  
   
 dat <- data.frame(Y = train.data[[ycol]], Xi,   
 check.names = FALSE)  
 glm.fit <- glm(Y ~ ., data = dat, family = binomial())  
 pvalue[i] <- coef(summary(glm.fit))[2, 4]  
}  
# Combine results into a nice table:  
results <- data.frame(Gene = gene.name, P\_value = pvalue)  
  
# Sort by significance  
results <- results[order(results$P\_value), ]  
head(results)  
alpha <- results[results$P\_value < 0.05,]  
cat("The dimensions of the significant genes (alpha) dataset is:")  
dim(alpha)  
  
ranked <- alpha$Gene  
ranked <- intersect(ranked, intersect(names(train.data), names(test.data)))  
steps <- seq(5, length(ranked), by = 5)  
  
if (length(ranked) > 0 && tail(steps, 1) != length(ranked)) {  
 steps <- c(steps, length(ranked))  
}  
glm\_curve <- data.frame(TopGenes = integer(), Test\_Acc = numeric())  
  
for (k in steps) {  
 genes\_k <- ranked[1:k]  
  
 Xi\_tr <- as.data.frame(train.data[, genes\_k, drop = FALSE])  
 Xi\_te <- as.data.frame(test.data[, genes\_k, drop = FALSE])  
  
 # Impute NAs in test with TRAIN medians (per feature)  
 for (nm in colnames(Xi\_tr)) {  
 med <- median(Xi\_tr[[nm]], na.rm = TRUE)  
 if (is.finite(med)) {  
 Xi\_tr[[nm]][is.na(Xi\_tr[[nm]])] <- med  
 Xi\_te[[nm]][is.na(Xi\_te[[nm]])] <- med  
 }  
 }  
  
 # Drop zero-variance predictors  
 nzv <- vapply(Xi\_tr, function(x) length(unique(na.omit(x))) > 1, logical(1))  
 Xi\_tr <- Xi\_tr[, nzv, drop = FALSE]  
 Xi\_te <- Xi\_te[, nzv, drop = FALSE]  
 if (ncol(Xi\_tr) == 0L) {  
 glm\_curve <- rbind(glm\_curve, data.frame(TopGenes = k, Test\_Acc = NA))  
 next  
 }  
  
 # build modeling frames  
 dat\_tr <- data.frame(Y = train.data$Y, Xi\_tr, check.names = FALSE)  
 dat\_te <- data.frame(Y = test.data$Y, Xi\_te, check.names = FALSE)  
   
 # Make Y a 2-level factor (most robust for binomial)  
 dat\_tr$Y <- factor(dat\_tr$Y, levels = c(0, 1))  
 dat\_te$Y <- factor(dat\_te$Y, levels = levels(dat\_tr$Y))  
   
   
 # Separation-safe GLM (brglm2)  
   
 # algo didn't converge  
 # fitted probs num 0/1 occured  
 #fit <- glm(Y ~ ., data = dat\_tr, family = binomial)  
   
   
 fit <- tryCatch(glm(Y ~ ., data = dat\_tr,   
 family = binomial(link = "logit"),   
 method = brglm2::brglmFit),  
 error = function(e) NULL)  
  
 if (is.null(fit)) {  
 glm\_curve <- rbind(glm\_curve, data.frame(TopGenes = k,   
 Test\_Acc = NA))  
 next  
 }  
   
 prob <- predict(fit, newdata = dat\_te, type = "response")  
 pred <- ifelse(prob > 0.5, 1, 0)  
 acc <- mean(pred == dat\_te$Y)  
  
 cat("Top", k, "→ Test Acc:", sprintf("%.4f", acc), "\n")  
 glm\_curve <- rbind(glm\_curve, data.frame(TopGenes = k,   
 Test\_Acc = acc))  
}  
# 2-lvl factor in both sets  
train.data$Y <- factor(train.data$Y, levels = c(0,1))  
test.data$Y <- factor(test.data$Y, levels = levels(train.data$Y))  
  
# made sure to run through all the genes  
max\_k <- min(37L, length(ranked))  
if (max\_k < 5L) max\_k <- length(ranked) # if you have <5 genes, just do that many  
steps <- seq(5L, max\_k, by = 5L)  
if (tail(steps, 1) != max\_k) steps <- c(steps, max\_k)  
  
# impute NAs in both train/test using TRAIN medians   
impute\_from\_train <- function(Xtr, Xte) {  
 for (nm in colnames(Xtr)) {  
 med <- median(Xtr[[nm]], na.rm = TRUE)  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 if (nm %in% colnames(Xte)) Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
 list(Xtr = Xtr, Xte = Xte)  
}  
  
# drop 0-var cols  
drop\_nzv <- function(Xtr, Xte) {  
 nzv <- vapply(Xtr, function(x) length(unique(na.omit(x))) > 1, logical(1))  
 Xtr <- Xtr[, nzv, drop = FALSE]  
 Xte <- Xte[, nzv, drop = FALSE]  
 list(Xtr = Xtr, Xte = Xte)  
}  
  
# storage  
qda\_curve <- data.frame(TopGenes = integer(), Test\_Acc = numeric())  
qda\_models <- vector("list", length(steps))  
names(qda\_models) <- paste0("top\_", steps)  
for (s in seq\_along(steps)) {  
 k <- steps[s]  
 genes\_k <- ranked[1:k]  
  
 Xtr <- as.data.frame(train.data[, genes\_k, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_k, drop = FALSE])  
  
 tmp <- impute\_from\_train(Xtr, Xte); Xtr <- tmp$Xtr; Xte <- tmp$Xte  
 tmp <- drop\_nzv(Xtr, Xte); Xtr <- tmp$Xtr; Xte <- tmp$Xte  
 if (ncol(Xtr) == 0L) {  
 qda\_curve <- rbind(qda\_curve, data.frame(TopGenes = k, Test\_Acc = NA\_real\_))  
 next  
 }  
  
 dat\_tr <- data.frame(Y = train.data$Y, Xtr, check.names = FALSE)  
 dat\_te <- data.frame(Y = test.data$Y, Xte, check.names = FALSE)  
  
 qda.fit <- tryCatch(qda(Y ~ ., data = dat\_tr),  
 error = function(e) { warning("top\_", k, ": ", e$message); NULL })  
 if (is.null(qda.fit)) {  
 qda\_curve <- rbind(qda\_curve, data.frame(TopGenes = k, Test\_Acc = NA\_real\_))  
 next  
 }  
  
 qda\_models[[s]] <- qda.fit  
 pred\_test <- predict(qda.fit, newdata = dat\_te)$class  
 acc <- mean(pred\_test == dat\_te$Y)  
 cat("Top", k, "→ Test Acc:", sprintf("%.4f", acc), "\n")  
  
 qda\_curve <- rbind(qda\_curve, data.frame(TopGenes = k, Test\_Acc = acc))  
}  
train.data$Y <- factor(train.data$Y, levels = c(0,1))  
test.data$Y <- factor(test.data$Y, levels = levels(train.data$Y))  
  
# helpers  
impute\_from\_train <- function(Xtr, Xte) {  
 for (nm in colnames(Xtr)) {  
 med <- median(Xtr[[nm]], na.rm = TRUE)  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 if (nm %in% colnames(Xte)) Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
 list(Xtr = Xtr, Xte = Xte)  
}  
drop\_nzv <- function(Xtr, Xte) {  
 nzv <- vapply(Xtr, function(x) length(unique(na.omit(x))) > 1, logical(1))  
 Xtr <- Xtr[, nzv, drop = FALSE]  
 Xte <- Xte[, nzv, drop = FALSE]  
 list(Xtr = Xtr, Xte = Xte)  
}  
  
# lda prep  
lda\_curve <- data.frame(TopGenes = integer(), Test\_Acc = numeric())  
lda\_models <- vector("list", length(steps))  
names(lda\_models) <- paste0("top\_", steps)  
for (s in seq\_along(steps)) {  
 k <- steps[s]  
 genes\_k <- ranked[1:k]  
  
 Xtr <- as.data.frame(train.data[, genes\_k, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_k, drop = FALSE])  
  
 # impute NAs (from TRAIN medians) + drop zero-variance  
 tmp <- impute\_from\_train(Xtr, Xte); Xtr <- tmp$Xtr; Xte <- tmp$Xte  
 tmp <- drop\_nzv(Xtr, Xte); Xtr <- tmp$Xtr; Xte <- tmp$Xte  
 if (ncol(Xtr) == 0L) {  
 lda\_curve <- rbind(lda\_curve, data.frame(TopGenes = k,   
 Test\_Acc = NA\_real\_))  
 next  
 }  
  
 dat\_tr <- data.frame(Y = train.data$Y, Xtr, check.names = FALSE)  
 dat\_te <- data.frame(Y = test.data$Y, Xte, check.names = FALSE)  
  
 lda.fit <- tryCatch(lda(Y ~ ., data = dat\_tr),  
 error = function(e)   
 { warning("top\_", k, ": ", e$message); NULL })  
   
 if (is.null(lda.fit)) {  
 lda\_curve <- rbind(lda\_curve, data.frame(TopGenes = k,   
 Test\_Acc = NA\_real\_))  
 next  
 }  
  
 lda\_models[[s]] <- lda.fit  
 pred\_test <- predict(lda.fit, newdata = dat\_te)$class  
 acc <- mean(pred\_test == dat\_te$Y)  
 cat("Top", k, "→ LDA Test Acc:", sprintf("%.4f", acc), "\n")  
  
 lda\_curve <- rbind(lda\_curve, data.frame(TopGenes = k, Test\_Acc = acc))  
}  
train.data$Y <- factor(train.data$Y, levels = c(0,1))  
test.data$Y <- factor(test.data$Y, levels = levels(train.data$Y))  
  
# k  
k1 <- 3  
k2 <- 5  
k3 <- 7  
cat("We will test out these k's:", k1, ",", k2, ",", k3)  
  
knn\_curve <- data.frame(TopGenes = integer(), k = integer(),   
 Test\_Acc = numeric())  
for (m in steps) {  
 genes\_m <- ranked[1:m]  
  
 Xtr <- as.data.frame(train.data[, genes\_m, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_m, drop = FALSE])  
 ytr <- train.data$Y  
 yte <- test.data$Y  
  
 # --- Impute NAs with TRAIN medians (per feature) ---  
 for (nm in colnames(Xtr)) {  
 med <- median(Xtr[[nm]], na.rm = TRUE)  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
  
 # --- Drop zero-variance columns (after impute) ---  
 nzv <- vapply(Xtr, function(x) length(unique(na.omit(x))) > 1, logical(1))  
 if (!any(nzv)) {  
 knn\_curve <- rbind(knn\_curve, data.frame(TopGenes = m, k = k1,   
 Test\_Acc = NA\_real\_))  
 next  
 }  
 Xtr <- Xtr[, nzv, drop = FALSE]  
 Xte <- Xte[, nzv, drop = FALSE]  
  
 # --- Standardize using TRAIN mean/sd (critical for KNN) ---  
 mu <- vapply(Xtr, mean, numeric(1), na.rm = TRUE)  
 sdx <- vapply(Xtr, sd, numeric(1), na.rm = TRUE); sdx[sdx == 0] <- 1  
 Xtr\_sc <- scale(Xtr, center = mu, scale = sdx)  
 Xte\_sc <- scale(Xte, center = mu, scale = sdx)  
  
 # --- KNN prediction on TEST ONLY ---  
 pred <- knn(train = Xtr\_sc, test = Xte\_sc, cl = ytr, k = k1)  
 acc <- mean(pred == yte)  
  
 cat("Top", m, "genes → K =", k1, " Test Acc:",   
 sprintf("%.4f", acc), "\n")  
 knn\_curve <- rbind(knn\_curve, data.frame(TopGenes = m, k = k1,   
 Test\_Acc = acc))  
}  
for (m in steps) {  
 genes\_m <- ranked[1:m]  
  
 Xtr <- as.data.frame(train.data[, genes\_m, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_m, drop = FALSE])  
 ytr <- train.data$Y  
 yte <- test.data$Y  
  
 # --- Impute NAs with TRAIN medians (per feature) ---  
 for (nm in colnames(Xtr)) {  
 med <- median(Xtr[[nm]], na.rm = TRUE)  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
  
 # --- Drop zero-variance columns (after impute) ---  
 nzv <- vapply(Xtr, function(x) length(unique(na.omit(x))) > 1, logical(1))  
 if (!any(nzv)) {  
 knn\_curve <- rbind(knn\_curve, data.frame(TopGenes = m, k = k2,   
 Test\_Acc = NA\_real\_))  
 next  
 }  
 Xtr <- Xtr[, nzv, drop = FALSE]  
 Xte <- Xte[, nzv, drop = FALSE]  
  
 # --- Standardize using TRAIN mean/sd (critical for KNN) ---  
 mu <- vapply(Xtr, mean, numeric(1), na.rm = TRUE)  
 sdx <- vapply(Xtr, sd, numeric(1), na.rm = TRUE); sdx[sdx == 0] <- 1  
 Xtr\_sc <- scale(Xtr, center = mu, scale = sdx)  
 Xte\_sc <- scale(Xte, center = mu, scale = sdx)  
  
 # --- KNN prediction on TEST ONLY ---  
 pred <- knn(train = Xtr\_sc, test = Xte\_sc, cl = ytr, k = k2)  
 acc <- mean(pred == yte)  
  
 cat("Top", m, "genes → K =", k2, " Test Acc:",   
 sprintf("%.4f", acc), "\n")  
 knn\_curve <- rbind(knn\_curve, data.frame(TopGenes = m, k = k2,   
 Test\_Acc = acc))  
}  
for (m in steps) {  
 genes\_m <- ranked[1:m]  
  
 Xtr <- as.data.frame(train.data[, genes\_m, drop = FALSE])  
 Xte <- as.data.frame(test.data[, genes\_m, drop = FALSE])  
 ytr <- train.data$Y  
 yte <- test.data$Y  
  
 # --- Impute NAs with TRAIN medians (per feature) ---  
 for (nm in colnames(Xtr)) {  
 med <- median(Xtr[[nm]], na.rm = TRUE)  
 if (is.finite(med)) {  
 Xtr[[nm]][is.na(Xtr[[nm]])] <- med  
 Xte[[nm]][is.na(Xte[[nm]])] <- med  
 }  
 }  
  
 # --- Drop zero-variance columns (after impute) ---  
 nzv <- vapply(Xtr, function(x) length(unique(na.omit(x))) > 1, logical(1))  
 if (!any(nzv)) {  
 knn\_curve <- rbind(knn\_curve, data.frame(TopGenes = m, k = k3,   
 Test\_Acc = NA\_real\_))  
 next  
 }  
 Xtr <- Xtr[, nzv, drop = FALSE]  
 Xte <- Xte[, nzv, drop = FALSE]  
  
 # --- Standardize using TRAIN mean/sd (critical for KNN) ---  
 mu <- vapply(Xtr, mean, numeric(1), na.rm = TRUE)  
 sdx <- vapply(Xtr, sd, numeric(1), na.rm = TRUE); sdx[sdx == 0] <- 1  
 Xtr\_sc <- scale(Xtr, center = mu, scale = sdx)  
 Xte\_sc <- scale(Xte, center = mu, scale = sdx)  
  
 # --- KNN prediction on TEST ONLY ---  
 pred <- knn(train = Xtr\_sc, test = Xte\_sc, cl = ytr, k = k3)  
 acc <- mean(pred == yte)  
  
 cat("Top", m, "genes → K =", k3, " Test Acc:",   
 sprintf("%.4f", acc), "\n")  
 knn\_curve <- rbind(knn\_curve, data.frame(TopGenes = m, k = k3,   
 Test\_Acc = acc))  
}