CSC 659 859 Summer 2025
Homework 2
Zari Haidarian
zhaidarian@sfsu.edu
6/28/25

2. Audit of Original database: describe source of data, and audit the DB following guidance in class slides. Provide all data stats.

The original database provided comes from the research paper "Cell type discovery using single-cell transcriptomics: implications for ontological representation." This paper discusses the results of random forest training on cell data and rna sequencing. The dataset has 608 columns (genes) + Label, and 871 rows (individual cells/nuclei).

- How are the feature (variable) data obtained and their meaning:

Based on the research paper, the features are obtained from single-cell and single-nuclei RNA sequencing. These are methods for studying gene expression levels measured per cell from cortical brain tissue. The features represent gene expression levels per cell, where each column in the CSV is a gene, and rows are individual cells/nuclei. The Label column in the CSV identifies the cell type cluster.

- How are class labels obtained/verified wrt. ground truth:

Class labels are assigned based on cluster membership from random forest clustering using NSforest (a supervised ML approach). Labels are divided to be either 0 or 1.
The ground truth is obtained from distinct transcriptomic signatures and marker gene expression, and is validated against prior biological knowledge.

- Is demography well covered in adequate and fair way:

The paper and the dataset provided data for a diverse range of cells. But besides the cellular diversity, any human demographics are not mentioned. So it is unclear what the level of demographic fairness is.

- Number of samples in each class; is the data unbalanced (unbalanced class is one having less than 10% of all class samples)

The dataset contains 871 total samples. Class 0 = 572 samples (= 65.67%). Class 1 = 299 samples (= 34.33%). A class is considered unbalanced if it has less than 10% of the total samples. Both Class 0 and Class 1 have more than 87 (10%) samples, so the dataset is NOT unbalanced.

- Type of features (numerical, categorical nominal or ordinal)

The features here are numerical. Gene expression values are all given in number form. Label is categorical, since it's either 0 or 1, used as a factor for classification.

- Missing values (do they need to be inputted and how)

There are no missing values.

- Are there enough samples compared to number of features used (must be at least 10 X more)

→ **Total features**: **608** (all gene expression variables used for training the model)

871 samples.

608 features.

871 is not >= (10 * 608)

Therefore there aren't enough samples.

- List and description of features, formats are well documented

Features are gene names (for example, KIT, TESPA1, GAD1, etc) as columns in the CSV. The paper lists many features along with their biological roles and significance. Format is documented by scientific standards (matrix: genes × cells).

- Check privacy issues (no personal features)

According to the research paper, the data consists of gene expression from post-mortem human brain samples. No personal identifiers like names, ages, or anything else were listed.

3. Creation of training DB and verification DB: Explain how you removed two random samples (one class 1 one class 0) and then summarize the stats of the remaining Training DB (original DB minus two random samples) and Verification DB (2 random samples).

Two random samples, one from class 0 and one from class 1, were selected using the sample() function. These were used as the Verification database. The remaining samples were used to train the Random Forest afterwards.

Summary

```
> print("Class Distribution:")
[1] "Class Distribution:"
> print(table(training_data$Label))

  0   1
571 298
```

```
> print("Class Distribution:")
[1] "Class Distribution:"
> print(table(verification_data$Label))

0 1
1 1
```

Prediction

```
> # Show predictions vs actual labels
> print("Verification Predictions:")
[1] "Verification Predictions:"
> print(data.frame(Actual = verification_data$Label, Predicted = predicted_labels))
    Actual Predicted
179      1         1
825      0         0
>
```

4. SW tools: explain what tools you used, for the HW.

I used R, RStudio, and for the libraries I used randomForest, caret, and ggplot2.

5. Experimental Methods and Setup

A. What ntree, mtry, cutoff ranges you used for RF training (e.g. for grid search).

ntree values = 500, 1000
mtry values = (0.5 * sqrt(length(e1data))), floor(0.5 * sqrt(ncol(training_data) - 1))
cutoff values = 0.3, 0.5, 0.7

B. Show formulae for accuracy measures you will use

Accuracy = (TP + TN) / (TP + TN + FP + FN)
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

C. List main SW class/classes/APIs you will use

randomForest, caret, ggplot2.

6. Actual Results of RF Training and Accuracy Estimates: Train RF using training database for chosen range of parameters and show results for best trained RF, namely: best ntree, mtry, cutoff; confusion matrix; OOB; recall, precision, F1;

After testing a few different values, the best-performing model was:

- ntree = 500
- mtry = 12
- cutoff = 0.5

```
>
> # ----------------------------
> # STEP 5: DISPLAY BEST MODEL RESULTS
> # ----------------------------
> cat("\n========================\n")

========================
> cat("Best Model Parameters:\n")
Best Model Parameters:
> print(best_params)
$ntree
[1] 500

$mtry
[1] 12

$cutoff
[1] 0.3

> cat("OOB Error Rate:", best_model$err.rate[best_model$ntree, "OOB"], "\n")
OOB Error Rate: 0.00575374
> print(best_metrics$conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1
```

```
 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

> cat("Accuracy:", best_metrics$accuracy, "\n")
Accuracy: 1
> cat("Precision:", best_metrics$precision, "\n")
Precision: 1
> cat("Recall:", best_metrics$recall, "\n")
Recall: 1
> cat("F1 Score:", best_metrics$f1, "\n")
F1 Score: 1
>
```

7. Feature Ranking: Explain how you obtained the best trained RF. Show feature ranking for top 10 ranged features (use MDA for R, GINI method for SciKit). How does it help you in explaining how RF worked? What did you learn doing this step? How does it compare with ground truth established in biology papers? Discuss and analyze.

I obtained the best trained RF by setting arrays with different possible values for NTREE and cutoff, then looping through the RF code I had with these different values to test all possible combinations. I displayed the predictions and metrics for each combination, then displayed the output for the best model (as shown in part 6).

Using the importance() function, I identified the top 10 features:

```
> cat("\nTop 10 Features (by MDA):\n")

Top 10 Features (by MDA):
> print(top_10_features)
                0        1 MeanDecreaseAccuracy MeanDecreaseGini
TESPA1    7.580010 6.888850             8.328015        16.711646
SLC17A7   6.300724 6.241370             7.478950        13.530986
SLIT3.1   6.825335 5.877262             7.208627         9.419804
ATP1B2    6.094817 5.719319             7.099710         9.397979
SLIT3.2   7.174682 5.591249             6.780755         7.978844
TBR1      6.721345 4.918868             6.776194         9.944416
SLIT3     6.440797 5.534550             6.745697         7.567887
KCNIP1    5.344924 5.868453             6.738172         6.770212
LINC00508 6.532254 5.262402             6.696230         6.054283
SFTA1P    5.974366 5.163071             6.568539         8.726216
>
```

Cluster ID	Cell type name
e1	TESPA1-expressing MTG cortical layer 2 excitatory neuron, human
i1	COL5A2-expressing MTG cortical layer 1 interneuron, human
i2	LHX6-expressing MTG cortical layer 2 interneuron, human
i3	BAGE2 expressing MTG cortical layer 1 interneuron, human
i4	ARHGAP36 expressing MTG cortical layer 1 interneuron, human
i5	KIT-expressing MTG cortical layer 1 interneuron, human
i6	GPR149-expressing MTG cortical layer 1 interneuron, human
i7	TGFBR2-expressing MTG cortical layer 1 interneuron, human
i8	SNCG-expressing MTG cortical layer 1 interneuron, human
i9	VIP-expressing MTG cortical layer 1 interneuron, human

8. RF Run Time test: Create RF run time engine (best trained model). Take 1 positive and 1 negative sample from the verification database and run them through Run Time RF to predict their class. Show tool output and classification results and compare prediction with ground truth. What is your confidence in RF predictions and based on what you determine it? Show RF output as class decision and class probability.

```
Prediction Results on Verification Set:
> print(results)
    Sample_ID Actual_Label Predicted_Label Prob_Class_0 Prob_Class_1
179       179            1               1        0.208        0.792
825       825            0               0        0.974        0.026
>
```

Sample 179 is a true positive. The model predicted class 1 with a probability of 0.792.
Sample 825 is a true negative. The model predicted class 0 with a very high probability of 0.974.

The confidence level is based on the probability associated with the predicted class. Even though the confidence for the positive sample is slightly lower than for the negative, both predictions are correct and within acceptable confidence ranges.

9. Resources: List resources/references used and names of students who helped you (if any)

Class slides and files:

- CART Trees and Random Forest ML (06-17-25)
- An Introduction to RF with R and RStudio_final.pptx
- HW 2 research paper explaning data with ground ruth .pdf

Websites:

- https://www.r-bloggers.com/2018/01/how-to-implement-random-forests-in-r/
- chatgpt.com

I did not work with any other students.

Appendix I: explain how you used GenAI with examples of your interaction and its output 1-2 pages. For each task you sued GenAI

- **– Describe each task for which you sued GenAI**
- **– Show example prompts**
- **– Rank usefulness from 1...5, 5 being very useful**

Task 1

I used it to help me debug my set up code from the slides when testing if my environment setup was working properly. The output when I was running it didn't match what the slides said it should and I couldn't find an online resource with the answer. The result it gave me was a different way of writing the code, as well as telling me not to use cutoff for some reason. I would put the usefulness at 1. I ended up solving my bug on my own and verified that the code it claimed was invalid was actually valid and correct. The error had nothing to do with my code, but with my R Studio set up.

Task 2

I used it to add comments to my code. It added detailed comments step by step, and I asked it to give a summary of what my code does as well which it did so I can verify that everything was doing what I thought and made sure I understood the functions better. I would put the usefulness at 5, it read and understood the code and added detailed and helpful comments.

🧠 Explanation of Key Concepts

Concept/Code	What it Does
`Label <- as.factor(Label)`	Ensures classification, not regression.
`sample()`	Randomly selects 1 sample from each class.
`randomForest(..., mtry = ..., ntree = ...)`	Trains a random forest. `ntree` = number of trees, `mtry` = number of variables considered at each split.
`verification_data`	Holds the 2 random samples (1 from each class) for testing.
`training_data`	Remaining dataset used to train the model.
`predict()`	Applies the trained model to new (unseen) data.
`varImpPlot()`	Visualizes which variables were most influential.

```r
# Install required packages (only run this once)
install.packages("randomForest")
install.packages("caret")
install.packages("ggplot2")

# Load the required libraries
library(randomForest)
library(caret)
library(ggplot2)

# Load the dataset (replace with your actual file path)
e1data <- read.csv("e1_positive.csv", header = TRUE)

# Ensure the 'Label' column is treated as a categorical variable (factor)
e1data$Label <- as.factor(e1data$Label)

# ------------------------------
# Step 1: Create Verification and Training Dataset
# ------------------------------

# Separate the indices by class
class1_indices <- which(e1data$Label == 1)
```

Copy Edit

Task 3

I had chatgpt confirm if my code was good or if there were any improvements needed for a few sections. For this part I would put the usefulness at 2. I mainly tried doing this just to experiment with chatgpt, as well as see if I was on the right track with my code as I went along. Chatgpt always wanted to make changes even if my code was working. Just to test the accuracy I would feed it snippets of code that it created for me (that I did not create) and it would still try to make more changes on top of it. Because of that it was hard to rely on the accuracy of its adjustments when it seemed to think that the code it created was wrong. Also since i dont have much experience with R it was confusing for me to see it constantly change its mind.

Appendix II: Show key pieces of code (copy the code into PDF format – do not send executable). Document the code please.

```
> # Load libraries
> library(randomForest)
> library(caret)
> library(ggplot2)
>
> # ----------------------------
> # STEP 1: LOAD DATA AND PREP
> # ----------------------------
> e1data <- read.csv("e1_positive.csv", header = TRUE)
>
> # Convert Label to factor so Random Forest treats it as categorical
> e1data$Label <- as.factor(e1data$Label)
>
> # ----------------------------
> # STEP 2: CREATE TRAINING AND VERIFICATION SETS
> # ----------------------------
>
> # Randomly pick one sample from each class for verification
> set.seed(123)
> verify1 <- sample(which(e1data$Label == 1), 1)
> verify0 <- sample(which(e1data$Label == 0), 1)
> verify_indices <- c(verify1, verify0)
>
> # Split into training and verification
> verification_data <- e1data[verify_indices, ]
> training_data <- e1data[-verify_indices, ]
>
```

```
> # ----------------------------
> # STEP 3: DEFINE METRIC FUNCTION (Precision, Recall, F1)
> # ----------------------------
>
> evaluate_model <- function(predicted, actual) {
+   cm <- confusionMatrix(predicted, actual)
+   accuracy <- cm$overall['Accuracy']
+   precision <- cm$byClass['Pos Pred Value']
+   recall <- cm$byClass['Sensitivity']
+   f1 <- 2 * (precision * recall) / (precision + recall)
+   return(list(conf_matrix = cm, accuracy = accuracy, precision = precision, recall = recall, f1 = f1))
+ }
>
```

```
> # ----------------------------
> # STEP 4: TRAIN RF WITH PARAMETER VARIATIONS
> # ----------------------------
>
> # Try different ntree and cutoff values
> ntree_values <- c(500, 1000)
> cutoff_values <- c(0.3, 0.5, 0.7)
>
> best_model <- NULL
> best_f1 <- 0
>
> for (ntree in ntree_values) {
+   for (cutoff in cutoff_values) {
+
+     # mtry formula based on number of features (here 608)
+     mtry_val <- floor(0.5 * sqrt(ncol(training_data) - 1))
+
+     model <- randomForest(Label ~ .,
+                           data = training_data,
+                           ntree = ntree,
+                           mtry = mtry_val,
+                           cutoff = c(1 - cutoff, cutoff), # Class 0, Class 1
+                           importance = TRUE,
+                           do.trace = 100)
+
+     # Predict on training data to compute confusion matrix
+     predictions <- predict(model, newdata = training_data)
+     metrics <- evaluate_model(predictions, training_data$Label)
+
+     cat("\n--- Model Evaluation ---\n")
+     cat("ntree:", ntree, " cutoff:", cutoff, "\n")
+     print(metrics$conf_matrix)
+     cat("F1 Score:", metrics$f1, "\n")
+
```

```
+     if (metrics$f1 > best_f1) {
+       best_model <- model
+       best_f1 <- metrics$f1
+       best_params <- list(ntree=ntree, mtry=mtry_val, cutoff=cutoff)
+       best_metrics <- metrics
+     }
+   }
+ }
ntree      OOB      1      2
  100:   0.58%  0.88%  0.00%
  200:   0.58%  0.88%  0.00%
  300:   0.58%  0.88%  0.00%
  400:   0.58%  0.88%  0.00%
  500:   0.58%  0.88%  0.00%

--- Model Evaluation ---
ntree: 500  cutoff: 0.3
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
```

```
            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

F1 Score: 1
ntree      OOB      1      2
  100:   0.81%  0.88%  0.67%
  200:   0.58%  0.53%  0.67%
  300:   0.46%  0.53%  0.34%
  400:   0.58%  0.53%  0.67%
  500:   0.58%  0.53%  0.67%

--- Model Evaluation ---
ntree: 500  cutoff: 0.5
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```

```
            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

F1 Score: 1
ntree      OOB      1      2
  100:   0.92%  0.18%  2.35%
  200:   0.81%  0.18%  2.01%
  300:   0.81%  0.18%  2.01%
  400:   0.69%  0.35%  1.34%
  500:   0.69%  0.18%  1.68%

--- Model Evaluation ---
ntree: 500  cutoff: 0.7
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```

```
            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

F1 Score: 1
ntree      OOB      1      2
  100:   0.69%  0.88%  0.34%
  200:   0.58%  0.88%  0.00%
  300:   0.58%  0.88%  0.00%
  400:   0.58%  0.88%  0.00%
  500:   0.58%  0.88%  0.00%
  600:   0.58%  0.88%  0.00%
  700:   0.58%  0.88%  0.00%
  800:   0.58%  0.88%  0.00%
  900:   0.58%  0.88%  0.00%
 1000:   0.58%  0.88%  0.00%

--- Model Evaluation ---
ntree: 1000  cutoff: 0.3
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
```

```
               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

F1 Score: 1
ntree      OOB      1      2
  100:   0.58%  0.53%  0.67%
  200:   0.46%  0.53%  0.34%
  300:   0.46%  0.53%  0.34%
  400:   0.46%  0.53%  0.34%
  500:   0.46%  0.53%  0.34%
  600:   0.46%  0.53%  0.34%
  700:   0.46%  0.53%  0.34%
  800:   0.46%  0.53%  0.34%
  900:   0.46%  0.53%  0.34%
 1000:   0.46%  0.53%  0.34%

--- Model Evaluation ---
ntree: 1000  cutoff: 0.5
Confusion Matrix and Statistics
```

```
          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

F1 Score: 1
ntree      OOB      1      2
  100:   0.58%  0.18%  1.34%
  200:   0.81%  0.35%  1.68%
  300:   0.69%  0.18%  1.68%
  400:   0.81%  0.35%  1.68%
  500:   0.81%  0.35%  1.68%
  600:   0.81%  0.35%  1.68%
  700:   0.69%  0.35%  1.34%
  800:   0.69%  0.35%  1.34%
  900:   0.69%  0.35%  1.34%
 1000:   0.69%  0.35%  1.34%
```

```
--- Model Evaluation ---
ntree: 1000  cutoff: 0.7
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

F1 Score: 1
>
```

```
> # ----------------------------
> # STEP 5: DISPLAY BEST MODEL RESULTS
> # ----------------------------
> cat("\n==========================\n")

==========================
> cat("Best Model Parameters:\n")
Best Model Parameters:
> print(best_params)
$ntree
[1] 500

$mtry
[1] 12

$cutoff
[1] 0.3

> cat("OOB Error Rate:", best_model$err.rate[best_model$ntree, "OOB"], "\n")
OOB Error Rate: 0.00575374
> print(best_metrics$conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 571   0
         1   0 298

               Accuracy : 1
                 95% CI : (0.9958, 1)
    No Information Rate : 0.6571
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA
```

```
 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6571
         Detection Rate : 0.6571
   Detection Prevalence : 0.6571
      Balanced Accuracy : 1.0000

       'Positive' Class : 0

> cat("Accuracy:", best_metrics$accuracy, "\n")
Accuracy: 1
> cat("Precision:", best_metrics$precision, "\n")
Precision: 1
> cat("Recall:", best_metrics$recall, "\n")
Recall: 1
> cat("F1 Score:", best_metrics$f1, "\n")
F1 Score: 1
>
```

```
> # ----------------------------
> # STEP 6: FEATURE RANKING USING MDA (Mean Decrease Accuracy)
> # ----------------------------
>
> # Get variable importance
> importance_values <- importance(best_model)
> # Sort by MeanDecreaseAccuracy
> top_features <- importance_values[order(-importance_values[, "MeanDecreaseAccuracy"]), ]
> top_10_features <- head(top_features, 10)
>
> # Print top 10 important features
> cat("\nTop 10 Features (by MDA):\n")

Top 10 Features (by MDA):
> print(top_10_features)
                0        1 MeanDecreaseAccuracy MeanDecreaseGini
TESPA1    7.580010 6.888850             8.328015        16.711646
SLC17A7   6.300724 6.241370             7.478950        13.530986
SLIT3.1   6.825335 5.877262             7.208627         9.419804
ATP1B2    6.094817 5.719319             7.099710         9.397979
SLIT3.2   7.174682 5.591249             6.780755         7.978844
TBR1      6.721345 4.918868             6.776194         9.944416
SLIT3     6.440797 5.534550             6.745697         7.567887
KCNIP1    5.344924 5.868453             6.738172         6.770212
LINC00508 6.532254 5.262402             6.696230         6.054283
SFTA1P    5.974366 5.163071             6.568539         8.726216
>
> # Plot importance
> varImpPlot(best_model, type = 1, main = "Top Features by Mean Decrease Accuracy")
>
> #
>
> # ----------------------------
> # STEP 7: PREDICT ON VERIFICATION DATA
> # ----------------------------
> predicted_labels <- predict(best_model, newdata = verification_data)
> predicted_probs <- predict(best_model, newdata = verification_data, type = "prob")
>
> results <- data.frame(
+   Sample_ID = rownames(verification_data),
+   Actual_Label = verification_data$Label,
+   Predicted_Label = predicted_labels,
+   Prob_Class_0 = predicted_probs[, "0"],
+   Prob_Class_1 = predicted_probs[, "1"]
+ )
>
> cat("\nPrediction Results on Verification Set:\n")

Prediction Results on Verification Set:
> print(results)
    Sample_ID Actual_Label Predicted_Label Prob_Class_0 Prob_Class_1
179       179            1               1        0.208        0.792
825       825            0               0        0.974        0.026
>
```