CS 486 Fall 2025 Homework 2

Timothy George
tmgeorge2@dons.usfca.edu

09/26/2025

I. Audit of Original Database

The dataset provided for this assignment originates from the Craig Venter Institute (JCV) biological study on single-cell transcriptomics. The source data corresponds to the e1 neuronal cluster, whose biological ground truth is documented in Aevermann et al. (2018). The task is a binary classification problem, where class label ‘1’ indicates samples belonging to the e1 cluster, and label ‘0’ indicates non-e1 cluster samples.

An overview of the statistics of the dataset is as follows:

- **Number of samples:** 871
- **Number of features:** 608
- **Label column:** ‘Label’ (binary, values 0 or 1)
- **Total columns:** 609 (608 features + 1 label)
- **Feature type:** All features are continuous numerical values representing gene expression measurements. There are no personally identifiable features in the dataset.
- **Missing values:** None (0 missing entries across all samples and features).

There are 572 samples in the negative decision class and 299 samples in the positive decision class. Since neither class makes up less than 10% of all class samples, both classes are adequately balanced.

With 871 samples and 608 features, there are 1.43x more samples than features. This is much lower than the standard of 10x more samples than features. It is important to note this may lead to overfitting due to the high dimensionality relative to the size of the dataset.

II. Creation of Training DB and Verification DB

In order to evaluate the Random Forest classifier on unseen data, the original dataset was partitioned into a Training Database and a Verification Database. Positive sample 21 and negative sample 370 were randomly selected from the original dataset and moved into the Verification Database. The remaining 869 samples form the Training Database. Both databases retain all 608 original gene expression features, with the last column ‘Label’ serving as the binary class indicator.

An overview of the training database and verification database statistics are as follows:

Training Database Statistics

- **Number of samples:** 869
- **Number of features:** 608
- **Class distribution:**
    - Negative class: 571 samples (65.72%)
    - Positive class: 298 samples (34.28%)

The Training Database preserves the original class proportions, ensuring that both classes remain adequately balanced after removing the two verification samples.

Verification Database Statistics

- **Number of samples:** 2
- **Number of features:** 608
- **Class distribution:**
    - Negative Class: 1 sample
    - Positive Class: 1 sample

This setup ensures that the Verification Database contains one representative sample from each class, enabling a straightforward test of the trained Random Forest model's predictive capability.

III. Software Tools Used

- R Documentation: https://www.r-project.org/other-docs.html
- ChatGPT: https://chatgpt.com/

IV. Experimental Methods and Setup

For this assignment, I decided to train Random Forest classifiers using the randomForest package in R. I decided to use the built-in out-of-bag (OOB) error estimation, which provides an unbiased estimate of model performance without requiring a separate validation split or cross-validation folds.

The experiment was designed to systematically explore the effect of three hyperparameters:

- NTREE (number of trees in the forest): the number of decision trees to grow
- MTRY: the number of variables randomly sampled as candidates at each split
- CUTOFF: classification threshold for assigning class probabilities to labels

The values of each hyperparameter to be tested via grid search are shown in Table 1 below:

NTREE	MTRY	CUTOFF
1000	$\frac{1}{2}\sqrt{608}$	0.3
2000	$\sqrt{608}$	0.5
5000	$2\sqrt{608}$	0.7

Table 1: Values of each hyperparameter to be iteratively tested

As shown above, three different values of NTREE will be selected: 1000, 2000, and 5000. For MTRY, I will use multiples of the square root of the total number of features (608), following the standard heuristic for Random Forest classification. For CUTOFF, I will test thresholds of 0.3, 0.5, and 0.7. The full grid includes the 27 possible parameter combinations from crossing NTREE, MTRY, and CUTOFF. Each model will be trained on the training database and evaluated using OOB error estimates. For the combination resulting in the lowest OOB error estimate, I will calculate the following accuracy statistics:

- Precision: fraction of positive predictions that are correct

  Precision = TP/(TP + FP)

- Recall: fraction of actual positives correctly predicted

  Recall = TP(TP + FN)

- F1 score: harmonic mean of precision and recall

  F1 = 2(Recall*Precision)/(Recall + Precision)

The main R functions and classes used were randomForest() for training and prediction, along with built-in methods for extracting OOB error rates, confusion matrices, class probabilities, and feature importance measures.

V: Results of RF Training and Accuracy Estimates

To initialize the data, the following lines of code to read and factor data from the training database were included at the top of the file:

```
e1data <- read.csv("trainingDB - e1_positive.csv", header=TRUE)
e1data$Label <- as.factor(e1data$Label)
```

Random Forest models were trained across the specified grid of hyperparameters, and performance was evaluated using out-of-bag (OOB) error estimation. This was done by iteratively calling randomForest() with every possible combination of hyperparameter values and storing the results of each trained model.

```
ntree_grid <- c(1000,2000,5000)
mtry_grid <- pmax(1, round(c(0.5, 1, 2) * sqrt(length(e1data))))
cutoff_grid <- list(0.3, 0.5, 0.7)

results <- data.frame(
  ntree = integer(),
  mtry = integer(),
  cutoff = numeric(),
  oob = numeric(),
  stringsAsFactors = FALSE
)
# Build and Train Model
for(n_tree in ntree_grid) {
  for(m_try in mtry_grid) {
    for(cut_off in cutoff_grid) {
      model1.rf <- randomForest(Label ~. , data = e1data,
                                ntree = n_tree,
                                mtry = m_try,
                                CUTOFF = cut_off,
                                do.trace=100)
      oob_err <- tail(model1.rf$err.rate[,"OOB"],1)
      results <- rbind(results, data.frame(ntree = n_tree, mtry =
m_try, cutoff = cut_off, oob = oob_err))
    }
  }
```

After comparing results across all parameter combinations, the optimal configuration with the lowest OOB error of 0.46% was found with NTREE = 1000, MTRY ≈ 12, and CUTOFF = 0.3.

In order to compute accuracy statistics, I recreated the run-time best model by calling randomForest() with the optimal NTREE, MTRY, and CUTOFF values and printed the resulting confusion matrix.

```
model1.rf <- randomForest(Label ~. , data = e1data,
                          ntree = best_values$ntree,
                          mtry = best_values$mtry,
                          importance = TRUE,
                          CUTOFF = best_values$cutoff,
                          do.trace=100)
print(model1.rf)
```

The corresponding confusion matrix for the best-trained model is shown in Table 2. This matrix summarizes the counts of true positives, true negatives, false positives, and false negatives, and it served as the basis for computing accuracy measures.

	Negative Prediction	Positive Prediction
Negative Ground Truth	568	3
Positive Ground Truth	1	297

Table 2: Confusion Matrix for Best-Trained RF (NTREE=1000, MTRY=12, CUTOFF=0.3)

From this matrix, I calculated and printed the precision, recall, and F1 score for the best-trained RF model.

```
TN <- model1.rf$confusion[1,1]   # True Negatives
FP <- model1.rf$confusion[1,2]   # False Positives
FN <- model1.rf$confusion[2,1]   # False Negatives
TP <- model1.rf$confusion[2,2]   # True Positives

precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
f1 <- 2 * (recall * precision) / (recall + precision)
```

```
cat("Precision:", round(precision, 4), "\n")
cat("Recall   :", round(recall, 4), "\n")
cat("F1 Score :", round(f1, 4), "\n")
```

This produced the following results:

```
> cat("Precision:", round(precision, 4), "\n")
Precision: 0.99000

> cat("Recall   :", round(recall, 4), "\n")
Recall   : 0.99660

> cat("F1 Score :", round(f1, 4), "\n")
F1 Score : 0.99330
```

This gives a precision of 0.99, a recall of 0.9966, and an F1 score of 0.9933.

VI: Feature Ranking

Feature importance was computed using the Mean Decrease in Accuracy (MDA) metric from the best-trained Random Forest model. Table 3 below summarizes the top 10 ranked features:

Feature	MDA
TESPA1	11.851642
KCNIP1	10.772331
SLC17A7	10.615117
NECAB1	9.725542
LINC00507	9.717754
ANKRD33B	9.628386
NPTX1	9.588385
SLIT3.2	9.552620
SLIT3	9.331406
ZNF536	9.325730

Table 3: Feature Ranking Based on MDA

These results align closely with the biological ground truth reported by Aevermann et al. The e1 excitatory neuron cluster was defined by selective expression of TESPA1, LINC00507, and SLC17A7, and by lack of expression of KCNIP1. Our Random Forest model independently surfaced all four of these marker genes within the top five ranked features. This strong agreement validates the RF classifier's ability to learn biologically meaningful distinctions rather than unrelated correlations.

Other highly ranked genes, such as NECAB1, ANKRD33B, and NPTX1, may represent secondary markers correlated with the e1 cluster, or they may capture related biological processes not highlighted in the ground truth definition. Their presence suggests that while TESPA1, LINC00507, and SLC17A7 are sufficient to define e1, RF leverages a broader set of features to maximize predictive performance.

The MDA scores also reveal a relatively tight cluster among the top 10 features (ranging from ~9.3 to ~11.9). This indicates that classification performance depends on a combination of markers rather than being dominated by a single feature. In practice, a

reduced model could likely achieve near-identical performance using the top 5–7 features.

VII: RF Run-Time Test

The final step was to evaluate the best-trained Random Forest model in prediction mode using the two samples from the Verification Database (one positive and one negative). The model produced both class decisions and class probabilities, shown in Table 4.

Sample ID	Ground Truth	Predicted Class	P(Class 0)	P(Class 1)
1	1	1	0.005	0.995
2	0	0	0.965	0.035

Table 4: Results of Random Forest Run-Time Test

Random Forest correctly classified both verification samples. The positive sample was predicted as class 1 with 99.5% probability, showing extremely high confidence. Similarly, the negative sample was predicted as class 0 with 96.5% probability, also with strong confidence.

These results reinforce the reliability of the model, as both predictions agree with the known ground truth labels. In addition, the high predicted probabilities provide further assurance that the classifier is not making borderline decisions, but rather confidently separating the two classes. Given this performance, we can conclude that the Random Forest generalizes well from the training database to unseen data, even when working with a small verification set.

VIII: References

- Research Paper: Cell type discovery using single-cell transcriptomics: implications for ontological representation
- An Introduction to RF with R and RStudio
- RF Experiments - Useful hints for R users for HM 2
- ChatGPT-5

Appendix I

To aid in developing my code, I asked GPT-5 to generate the following code for feature ranking:

```
importance_vals <- importance(model1.rf)
importance_df <- data.frame(Feature = rownames(importance_vals),
                            MDA = importance_vals[,
"MeanDecreaseAccuracy"])

importance_df <- importance_df[order(-importance_df$MDA), ]

print(head(importance_df, 10))

varImpPlot(model1.rf, n.var = 10)
```

For my prompt, I attached all of my R code up to this point, then asked “Attached is some R code. Generate code to rank the top 10 features from the run-time best model”. After verifying its solution, I can conclude that the tool was very helpful for this little code snippet.

Appendix II

```
library(randomForest)
library(caret)
library(e1071)

# Load Data
e1data <- read.csv("trainingDB - e1_positive.csv", header=TRUE)
e1data$Label <- as.factor(e1data$Label)


#Grid Search all values of NTREE, MTRY, CUTOFF
ntree_grid <- c(1000,2000,5000)
mtry_grid <- pmax(1, round(c(0.5, 1, 2) * sqrt(length(e1data))))
cutoff_grid <- list(0.3, 0.5, 0.7)

results <- data.frame(
  ntree = integer(),
  mtry = integer(),
  cutoff = numeric(),
  oob = numeric(),
  stringsAsFactors = FALSE
)

# Build and Train Model
for(n_tree in ntree_grid) {
  for(m_try in mtry_grid) {
    for(cut_off in cutoff_grid) {
      model1.rf <- randomForest(Label ~. , data = e1data,
                                ntree = n_tree,
                                mtry = m_try,
                                CUTOFF = cut_off,
                                do.trace=100)
      oob_err <- tail(model1.rf$err.rate[,"OOB"],1)
      results <- rbind(results, data.frame(ntree = n_tree, mtry =
m_try, cutoff = cut_off, oob = oob_err))
    }
  }
}
```

```r
# Get best hyperparameters
best_values <- results[which.min(results$oob), ]
print(best_values)

# Recreate run-time best model
model1.rf <- randomForest(Label ~. , data = e1data,
                          ntree = best_values$ntree,
                          mtry = best_values$mtry,
                          importance = TRUE,
                          CUTOFF = best_values$cutoff,
                          do.trace=100)
print(model1.rf)

# Calculate accuracy statistics
TN <- model1.rf$confusion[1,1]   # True Negatives
FP <- model1.rf$confusion[1,2]   # False Positives
FN <- model1.rf$confusion[2,1]   # False Negatives
TP <- model1.rf$confusion[2,2]   # True Positives

precision <- TP / (TP + FP)
recall <- TP / (TP + FN)
f1 <- 2 * (recall * precision) / (recall + precision)

cat("Precision:", round(precision, 4), "\n")
cat("Recall   :", round(recall, 4), "\n")
cat("F1 Score :", round(f1, 4), "\n")

# Rank top 10 features (Taken from ChatGPT)
importance_vals <- importance(model1.rf)
importance_df <- data.frame(Feature = rownames(importance_vals),
                            MDA = importance_vals[,
"MeanDecreaseAccuracy"])

importance_df <- importance_df[order(-importance_df$MDA), ]

print(head(importance_df, 10))

varImpPlot(model1.rf, n.var = 10)
```

```
#Save and verify trained model
saveRDS(model1.rf, "modelBEST.rds")
modelbest.rf <- readRDS("modelBEST.rds")
verification_data <- read.csv("verificationDB - e1_positive.csv",
header=TRUE)
verification_data$Label <- as.factor(verification_data$Label)
predictions <- predict(modelbest.rf, verification_data, type="prob")
print(predictions)
```