User Manual for Random Forest-Based Bacterial Morphology Prediction Software

# I. Function Overview

This pipeline annotates every protein in each bacterial genome with Pfam domains using pfam\_scan (Pfam‑A v33.0), concatenates the domains to build a domain‑abundance matrix, and couples the matrix with observed phenotypes to train a Random Forest (randomForest package) for microbial group discrimination, feature selection and visualisation. The workflow targets binary or multiclass classification and reports domain‑level feature importance and cross‑validation error curves.

# II. Input File Format

## 1. Grouping File (e.g., 3750group3-1.txt)

(1) Row names are sample IDs;

(2) Must include a “Type” column (category/group label, which must be of factor type), and may include a “Group” column (used to distinguish training set (group1) and validation set (group2)). Example:

SampleID Type Group  
Species1 T1 group1  
Species2 T2 group1  
Species3 T1 group2  
...

## 2. Abundance Matrix (e.g., train\_matrix\_fixed.txt)

(1) Row names are features/domains (e.g., bacteria or genes);

(2) Column names are sample IDs;

(3) Each cell contains an abundance value. Example:

Species1 Species2 Species3 ...  
feature1 2 3 0  
feature2 5 1 1  
...

# III. Main Steps and Commands

## 1. Environment Preparation

# Install (if missing) and load required packages  
pkgs <- c('randomForest', 'ggplot2', 'pheatmap')  
to\_install <- setdiff(pkgs, rownames(installed.packages()))  
if (length(to\_install)) install.packages(to\_install, dependencies = TRUE)  
lapply(pkgs, library, character.only = TRUE)

## 2. Data Reading and Processing

# Please manually decompress the train\_matrix\_fixed.zip file first. The file was too large for GitHub to upload, so we uploaded it as a compressed file.

# Read the grouping and abundance matrix  
design = read.table("3750group3-1.txt", header = T, row.names = 1)  
design$Type = as.factor(design$Type)  
otu\_table = read.table("train\_matrix\_fixed.txt", header = T, row.names = 1)  
otu\_table = t(otu\_table) # Ensure samples are in rows, features in columns

## 3. Select Training Set

design\_sub = subset(design, Group %in% c("group1"))  
# Ensure sample IDs match  
idx = rownames(design\_sub) %in% colnames(otu\_table)  
design\_sub = design\_sub[idx,]  
otu\_sub = otu\_table[, rownames(design\_sub)]

## 4. Random Forest Model Training

# You can manually change the value inside set.seed(3) for each run; performing several such adjustments effectively supplies your own random seeds.

set.seed(3)  
rf = randomForest(t(otu\_sub), design\_sub$Type, importance=TRUE, proximity=TRUE, ntree=1000)  
save(rf, file = 'bestmodel\_group3\_1.RData')

## 5. Feature Selection and Cross-Validation

set.seed(827)  
result = rfcv(t(otu\_sub), design\_sub$Type, cv.fold=5)  
save(result, file = 'best\_rfcv\_group3\_1.RData')  
error\_data <- as.data.frame(result$error.cv)  
write.table(error\_data, file = 'best\_error\_group3\_1.txt', sep = '\t', row.names = T, quote = F, col.names = NA)  
with(result, plot(n.var, error.cv, log="x", type="o", lwd=1))

## 6. Training Set Prediction and Result Export

train.p = predict(rf, type = "response")  
df = data.frame(observed = design\_sub$Type, predict = train.p)  
write.table(df, file = "train\_predict\_group3\_1.txt", quote = F, sep = '\t', row.names = T, col.names = T)

## 7. Feature Importance Analysis and Visualization

imp = as.data.frame(rf$importance)  
imp = imp[order(imp[,1], decreasing = T), ]  
write.table(imp, file = "best\_importance\_class\_group3\_1.txt", quote = F, sep = '\t', row.names = T, col.names = T)  
  
# Visualization (bar chart, Top 10)  
varImpPlot(rf, main = "Top 10 - Feature importance", n.var = 10)  
  
# Beautification using ggplot2  
imp = read.table("best\_importance\_class\_group3\_1.txt", header=T, row.names= 1, sep="\t")  
imp = head(imp, n=23)  
imp = imp[order(imp[,3]), ]  
imp$Domain = factor(rownames(imp), levels=rownames(imp))  
p = ggplot(data = imp, mapping = aes(x=Domain, y=MeanDecreaseAccuracy, fill=Domain)) +   
 geom\_bar(stat="identity") + coord\_flip() + theme\_bw()  
ggsave(p, filename = "imp\_shape.pdf", width = 16, height = 9)

## 8. Validation Set Prediction and Evaluation

design\_test = subset(design, Group %in% c("group2"))  
idx = rownames(design\_test) %in% colnames(otu\_table)  
design\_test = design\_test[idx, ]  
otu\_sub = otu\_table[, rownames(design\_test)]  
otutab\_t = as.data.frame(t(otu\_sub))  
otutab\_t$Type = design[rownames(otutab\_t), ]$Type  
  
otutab.pred = predict(rf, t(otu\_sub))  
pre\_tab = table(observed=otutab\_t[,"Type"], predicted=otutab.pred)  
print(pre\_tab)  
predict = data.frame(Type = otutab\_t[,"Type"], predicted=otutab.pred)  
write.table("SampleID\t", file="RF\_prediction\_binary.txt", append=F, quote=F, eol="", row.names=F, col.names=F)  
write.table(predict, file = "RF\_prediction\_binary.txt", append=T, quote = F, row.names = T, col.names = F, sep = "\t")

# IV. Common Issues and Precautions

1. Sample IDs must be strictly consistent: Sample IDs in the grouping table and abundance matrix must match, otherwise data cannot be properly associated.  
2. Grouping variables must be of factor type: Otherwise, errors may occur during model training or evaluation.  
3. Output file overwriting: Pay attention to file overwriting and naming to avoid accidental overwriting.  
4. R environment: It is recommended to use R ≥ 4.0.0, and ensure that the required R packages (randomForest, ggplot2, pheatmap) are installed in advance.

# V. Result Interpretation

1. train\_predict\_group3\_1.txt: Comparison of training set predictions and true results  
2. best\_importance\_class\_group3\_1.txt: Feature importance ranking  
3. imp\_shape.pdf: Feature importance beautified chart  
4. RF\_prediction\_binary.txt: Validation set prediction results  
5. best\_error\_group3\_1.txt: Table of feature count vs. error rate

# VI. Hyperparameters Used in Random Forest Model Training

rf = randomForest(t(otu\_sub), design\_sub$Type, importance=TRUE, proximity=TRUE, ntree = 1000)

- ntree = 1000  
 Specifies that the number of trees generated in the random forest is 1000.  
 In the principle of random forests, each tree has a certain degree of "randomness." If the number of trees is too small, the model results are prone to fluctuation and lack stability. Increasing the number of trees makes the model more stable, reduces randomness, and improves generalization ability, but also increases computation time and slows down the analysis process. In practice, usually 500–2000 trees can ensure model stability; 1000 is a moderate value, which ensures the stability of importance indicators without significantly slowing down the process, making it a reliable choice.  
  
- importance = TRUE  
 Enables feature importance evaluation. This parameter does not affect the model itself but outputs the contribution of each feature to model classification (e.g., MeanDecreaseAccuracy).  
  
- proximity = TRUE  
 Calculates the proximity matrix between samples, which can be used for subsequent visualization or sample clustering analysis. This parameter also does not affect the model's predictive performance.  
  
- Other hyperparameters (e.g., mtry, nodesize, etc.)  
 Default values are used, i.e., mtry is the number of features selected at each split (by default, the square root of the total number of features), and nodesize is the minimum number of samples in each leaf node of a tree (default is 1 for classification tasks).  
 Using default values facilitates process standardization, helps unify experimental workflows, and makes results comparable across different batches or operators. It also avoids tedious parameter tuning, especially in exploratory analysis or when data volume is small, thus improving efficiency.

# VII. Model Validation Type

Cross-validation is used for feature selection and validation:

result = rfcv(t(otu\_sub), design\_sub$Type, cv.fold=5)

1. 5-fold cross-validation (cv.fold=5) is used.  
2. The training set is randomly divided into 5 subsets; each time, 4 subsets are used for training and 1 subset for validation. This is repeated 5 times to obtain cross-validation error rates (error.cv) for different numbers of features.  
3. The optimal number of features (where the error rate is lowest) is selected based on cross-validation results, thereby avoiding overfitting.