# EvoDA: Supervised Learning for Discriminating Models of Trait Evolution

# Introduction

Phylogenetic comparative methods (PCMs) have revolutionized how we study trait variation across species. Central to these methods is the adoption of a probabilistic model of trait evolution that parameterizes the processes that shape trait change over time across a phylogeny of species. Methods for selecting these models have traditionally relied on conventional approaches, such as AIC-based model selection, to weigh evidence in favor of candidate models. Here we employ supervised learning algorithms that are widely applicable to a wide breadth of experimental and evolutionary conditions, offering promising new advances in the quest to understand biodiversity.

In this tutorial, we introduce **Evolutionary Discriminant Analysis (EvoDA)**, a supervised learning framework designed to predict the evolutionary processes shaping trait variation. EvoDA leverages training and testing datasets simulated under known models to build classifier algorithms that demonstrate higher predictive accuracy, even when analyzing messy trait data. This guide walks you through each step of the process using a set of fungal phylogeny case studies.

|  |
| --- |
| 💡 Tip: Tutorial Manual and where find it: |
| See the file <https://github.com/radamsRHA/EvoDA/> for detailed instructions, associated R packages, and datasets used in this workflow. |

# Tutorial Outline

Jump to the section you need:

* [Installing Dependencies](#installing-dependencies)
* [Quick Example using TraitTrainR](#quick-example-using-traittrainr)
* [Training Data: Fungal Phylogeny - Case Study II](#Xb5b1c57df4e367f4a879406e9e46da9073e14ae)
* [Testing Data: Fungal Phylogeny](#testing-data)
* [Rejection Sampling](#rejection-sampling-coming-soon)

# Installing Dependencies

Before using EvoDA, install the required R packages. This includes the EvoDA helper package TraitTrainR:

|  |
| --- |
| 💡 Note: R version: TraitTrainR was written in R 4.4.0 (“Puppy cup”) and we recommend that version or later for installing TraitTrainR |
| 💡 Note: example training and testing datasets are provided in: <https://github.com/radamsRHA/EvoDA/tree/main/EmpiricalExample> |

install.packages("devtools")  
library(devtools)  
install\_github("radamsRHA/TraitTrainR")  
  
install.packages("phytools") # version >= 2.1-1  
install.packages("geiger") # version >= 2.0.11  
  
library(TraitTrainR)  
library(phytools)  
library(geiger)

# TraitTrainR Evolutionary Models Overview

Before exploring the example, it’s helpful to understand the evolutionary models and their associated parameters available in TraitTrainR, which will be used to generate training and testing datasets in this example. The table below summarizes the parameters, evolutionary processes, and distributions that will be used for simulations. These distributions align with default assumptions in fitContinuous from the geiger package. However, in your own analyses, you are not restricted to these settings (see <https://github.com/radamsRHA/TraitTrainR/> R package for details)

**Table 1. Trait models, parameters, evolutionary processes, and sampling distributions used in TraitTrainR.**

| Model | Parameters | Evolutionary Process | Distributions |
| --- | --- | --- | --- |
| BM | z0; σ² | Random-walk model of trait change (Felsenstein 1973) | σ² ~ Exp(1); z0 ~ N(0, 1) |
| OU | z0; σ²; α | Stabilizing selection (Butler and King 2004) | σ² ~ Exp(1); z0 ~ N(0,1); α ~ U(exp(-500), exp(1)) |
| EB | z0; σ²; a | Adaptive radiation: rates decline over time (Harmon et al. 2010) | σ² ~ Exp(1); z0 ~ N(0, 1); a ~ U(-5/depth, -1e-6) |
| Lambda | z0; σ²; λ | Phylogenetic signal scaling (Pagel 1999) | σ² ~ Exp(1); z0 ~ N(0, 1); λ ~ U(exp(-500), 1) |
| Delta | z0; σ²; δ | Time-dependent evolutionary rate (Pagel 1999) | σ² ~ Exp(1); z0 ~ N(0, 1); δ ~ U(exp(-500), 3) |
| Kappa | z0; σ²; κ | Branch length scaling in phylogeny (Pagel 1999) | σ² ~ Exp(1); z0 ~ N(0, 1); κ ~ U(exp(-500), 1) |
| Trend | z0; σ²; slope | Linear change in trait evolution over time (Pennell et al. 2014) | σ² ~ Exp(1); z0 ~ N(0, 1); slope ~ U(-100, 100) |

# 🚴 QUICK START CODE: simulate under Brownian Motion and Ornstein–Uhlenbeck models (see next section for more detailed information):

To demonstrate TraitTrainR functionality, here is an example using three basic models (BM, OU, EB) and a simple tree with three species. The following code will provide a an object MySimulationResults that includes a set of training datasets for input into EvoDA algorithms (described in following sections).

|  |
| --- |
| 💡 Why start simple? |
| This quick example serves as a sandbox to understand the structure of simulations in TraitTrainR. It ensures you’re comfortable before scaling up to realistic phylogenies and complex parameterizations. |

library(TraitTrainR)  
library(phytools)  
library(geiger)  
  
MyTree <- read.tree(text = "((A:1, B:1):1, C:2);")   
plot(MyTree)  
  
list.SimulationModelSettings <- list()  
NumberOfReps <- 5  
list.Rmatrix <- lapply(1:NumberOfReps, function(i) matrix(1, 1, 1))  
  
list.SimulationModelSettings[[1]] <- list(string.SimulationModel = "BM",   
 vector.Sig2 = rexp(NumberOfReps, rate = 1),   
 vector.AncestralState = rep(1, NumberOfReps),   
 list.Rmatrix = list.Rmatrix)  
  
list.SimulationModelSettings[[2]] <- list(string.SimulationModel = "OU",   
 vector.Sig2 = rexp(NumberOfReps, rate = 1),   
 vector.AncestralState = rnorm(NumberOfReps),  
 vector.Alpha = runif(NumberOfReps, min = exp(-500), max = exp(1)),  
 list.Rmatrix = list.Rmatrix)  
  
list.SimulationModelSettings[[3]] <- list(string.SimulationModel = "EB",   
 vector.Sig2 = rexp(NumberOfReps, rate = 1),   
 vector.AncestralState = rnorm(NumberOfReps),   
 vector.A = runif(NumberOfReps, min = log(10^-5)/310, max = -0.000001),  
 list.Rmatrix = list.Rmatrix)  
  
MySimulationResults <- TraitTrain(handle.Phylogeny = MyTree,  
 list.SimulationModelSettings = list.SimulationModelSettings,  
 logical.PIC = TRUE, logical.PROJECT = TRUE)

Now that we’ve explored a basic example using a toy tree, let’s take things to the next level.

We’re now ready to dive into the real application: a **fungal phylogeny**. This case study will provide the training data needed to power EvoDA’s supervised learning approach for building classifer algorithms with discriminant functinons.

# 🔍 Case Study II: Training and Testing data with Fungal phylogeny

In this case study, we use a realistic fungal phylogeny composed of 18 species, including major taxa like *Aspergillus*, *Neurospora*, *Saccharomyces*, and *Cryptococcus*. This deep and diverse tree reflects a broad evolutionary timespan, ideal for exploring evolutionary model discrimination. Here, we generate trait data under three models—Brownian Motion (BM), Ornstein-Uhlenbeck (OU), and Early Burst (EB)—and divide the workflow into training and testing phases to support EvoDA analysis. The following code details the workflow, from training to testing and visualizing the results of EvoDA analyses.

### 🧪 First, load all dependancies that can be used from start to finish

# Load all dependencies for workflow, from simulation to training, testing, and visualizing output  
rm(list = ls()) # clear all objects  
library(TraitTrainR) # for simulating training/testing data  
library(ape) # for general tree manipulations  
library(geiger) # for simulation of traits  
library(phytools) #for general tree manipulations  
library(caret) # for plotting manipulations  
library(corrplot) # for plotting purposes  
library(MASS) # for LDA and QDA  
library(mda) # for FDA and MDA  
library(klaR) # for RDA  
library(ggplot2) # for visualization purposes  
library(dplyr) # data manipulation   
library(tidyr)# data manipulation

### 🍄 Load Fungal Phylogeny

#############################  
# get fungal phylogeny here #  
#############################  
handle.TargetTree <- read.tree(text = "((((A.fumigatus:138.308934,A.nidulans:138.308934):191.332495,((M.oryzae:175.091151,(N.discreta:14.736545,(N.tetrasperma:5.443496,N.crassa:5.443496):9.293049):160.354606):33.404309,F.graminearum:208.49546):121.145969):232.724291,((((N.castellii:145.870505,(S.bayanus:60.615136,(((S.paradoxus:25.739496,S.cerevisiae:25.739496):12.037587,S.mikatae:37.777084):13.200718,S.kudriavzevii:50.977802):9.637334):85.255369):21.321276,C.glabrata:167.191781):29.950882,L.kluyveri:197.142664):111.428464,(C.albicans:135.842772,C.parapsilosis:135.842772):172.728356):253.794591):281.634281,C.neoformans:844);")

### 🔧 Define simulation parameters for training with TraitTrain

#######################  
# Simulation Settings #  
#######################  
numeric.MeasurementError <- 0 # simulate without any trait measurement error  
list.SimulationModelSettings <- list() # define an empty model list  
numeric.NumberTrainingReps <- 10^4 # same number of replicates for all models in list.SimulationModelSettings. Adjust this to increase accuracy, here we only use 10,000 replicates  
list.Rmatrix <- list(); for (i in 1:numeric.NumberTrainingReps){list.Rmatrix[[i]] <- matrix(1, nrow = 1, ncol = 1)} # three traits. Different rates for different traits can be specified here.   
  
######################  
# First model is BM #  
######################  
list.SimulationModelSettings[[1]] <- list(string.SimulationModel = "BM",   
 vector.Sig2 = rexp(n = numeric.NumberTrainingReps, rate = 1),   
 vector.AncestralState = rnorm(n = numeric.NumberTrainingReps),  
 list.Rmatrix = list.Rmatrix)  
#######################  
# Second model is OU #  
#######################  
list.SimulationModelSettings[[2]] <- list(string.SimulationModel = "OU",   
 vector.Sig2 = rexp(n = numeric.NumberTrainingReps, rate = 1),   
 vector.AncestralState = rnorm(n = numeric.NumberTrainingReps),  
 vector.Alpha = runif(n = numeric.NumberTrainingReps, min = exp(-500), max = exp(1)),  
 list.Rmatrix = list.Rmatrix)  
######################  
# third model is EB #  
######################  
list.SimulationModelSettings[[3]] <- list(string.SimulationModel = "EB",   
 vector.Sig2 = rexp(n = numeric.NumberTrainingReps, rate = 1),   
 vector.AncestralState = rnorm(n = numeric.NumberTrainingReps),   
 vector.A = runif(n = numeric.NumberTrainingReps, min = log(10^-5)/vcv(handle.TargetTree)[1,1], max = -0.000001),  
 list.Rmatrix = list.Rmatrix)

### 🧬 Simulate training datasets!

####################  
# SIMULATE TRAITS! #  
####################  
handle.RESULTS\_TRAIN <- TraitTrain(handle.Phylogeny = handle.TargetTree,  
 list.SimulationModelSettings = list.SimulationModelSettings,  
 logical.PIC = T, logical.PROJECT = T, numeric.MeasurementError = numeric.MeasurementError)

### 🧪 Testing Data Simulation

### 🔧 Define simulation parameters for testing. We need a new out-of-training set that is completely independent of the training data.

#############################  
# Simulation Model Settings #  
#############################  
list.SimulationModelSettings <- list() # define an empty model list  
numeric.NumberTrainingReps <- 10^3 # same number of replicates for all models in list.SimulationModelSettings  
list.Rmatrix <- list(); for (i in 1:numeric.NumberTrainingReps){list.Rmatrix[[i]] <- matrix(1, nrow = 1, ncol = 1)} # three traits. Different rates for different traits can be specified here.   
  
######################  
# First model is BM #  
######################  
list.SimulationModelSettings[[1]] <- list(string.SimulationModel = "BM",   
 vector.Sig2 = rexp(n = numeric.NumberTrainingReps, rate = 1),   
 vector.AncestralState = rnorm(n = numeric.NumberTrainingReps),  
 list.Rmatrix = list.Rmatrix)  
#######################  
# Second model is OU #  
#######################  
list.SimulationModelSettings[[2]] <- list(string.SimulationModel = "OU",   
 vector.Sig2 = rexp(n = numeric.NumberTrainingReps, rate = 1),   
 vector.AncestralState = rnorm(n = numeric.NumberTrainingReps),  
 vector.Alpha = runif(n = numeric.NumberTrainingReps, min = exp(-500), max = exp(1)),  
 list.Rmatrix = list.Rmatrix)  
######################  
# third model is EB #  
######################  
list.SimulationModelSettings[[3]] <- list(string.SimulationModel = "EB",   
 vector.Sig2 = rexp(n = numeric.NumberTrainingReps, rate = 1),   
 vector.AncestralState = rnorm(n = numeric.NumberTrainingReps),   
 vector.A = runif(n = numeric.NumberTrainingReps, min = log(10^-5)/vcv(handle.TargetTree)[1,1], max = -0.000001),  
 list.Rmatrix = list.Rmatrix)

### 🧪 Simulate testing traits

####################  
# SIMULATE TRAITS! #  
####################  
handle.RESULTS\_TEST <- TraitTrain(handle.Phylogeny = handle.TargetTree,   
 list.SimulationModelSettings = list.SimulationModelSettings,   
 logical.PIC = T, logical.PROJECT = T, numeric.MeasurementError = numeric.MeasurementError)

### 📊 Prepare training and testing datasets for EvoDA

|  |
| --- |
| 🧠 Why PICs? |
| PICs remove phylogenetic autocorrelation from trait data, allowing you to assess trait variation independently of shared ancestry. This is can help improve accuracy between empirical and simulated data. |

DATA\_TRAIN <- handle.RESULTS\_TRAIN$RESULTS\_PIC  
DATA\_TRAIN$SimulationModelNumber <- factor(DATA\_TRAIN$SimulationModelNumber)  
  
DATA\_TEST <- handle.RESULTS\_TEST$RESULTS\_PIC  
DATA\_TEST$SimulationModelNumber <- factor(DATA\_TEST$SimulationModelNumber)

### 📊 Train the EvoDA algorithms!

##########################  
# Train EvoDA algorithms #  
##########################  
MODEL\_LDA <- lda(SimulationModelNumber ~., DATA\_TRAIN, method = "mle") #LDA  
MODEL\_QDA <- qda(SimulationModelNumber ~., DATA\_TRAIN, method = "mle") #QDA   
MODEL\_FDA <- fda(SimulationModelNumber ~., DATA\_TRAIN, method = bruto) #FDA  
MODEL\_MDA <- mda(SimulationModelNumber ~., DATA\_TRAIN, method = bruto) # MDA  
MODEL\_RDA <- train(SimulationModelNumber ~ ., data = DATA\_TRAIN, method = "rda", trControl = trainControl(method = "cv", number = 10), tuneGrid = data.frame(gamma = 0, lambda = c(0, 0.2, 0.4, 0.6, 0.8, 1)), shuffle = T) # RDA with grid search

### 🤖 Evaluate EvoDA preformance: predict trait models using testing dataset.

####################  
# Make predictions #  
####################  
PREDICT\_LDA <- factor(predict(MODEL\_LDA, DATA\_TEST)$class)  
PREDICT\_QDA <- factor(predict(MODEL\_QDA, DATA\_TEST)$class)  
PREDICT\_FDA <- factor(predict(MODEL\_FDA, DATA\_TEST))  
PREDICT\_MDA <- factor(predict(MODEL\_MDA, DATA\_TEST))  
PREDICT\_RDA <- factor(predict(MODEL\_RDA, DATA\_TEST))

### 🤖 Plot confusion matrix for predictive accuracy

##############  
# compute CM #  
##############  
matrix.CONFUSED <- t(confusionMatrix(data = PREDICT\_FDA, reference = DATA\_TEST$SimulationModelNumber)$table)  
matrix.CONFUSED <- matrix.CONFUSED/1000; colnames(matrix.CONFUSED) <- row.names(matrix.CONFUSED) <- c("BM", "OU", "EB"); matrix.CONFUSED <- matrix.CONFUSED \*100  
  
corrplot(matrix.CONFUSED, type = "full", order = "original", method = "shade",  
 col = COL1("Reds", 10), add = FALSE, diag = TRUE, addCoef.col = "black",   
 tl.col = "black", tl.srt = 45, is.corr = FALSE,   
 number.cex = 2, addgrid.col = NULL, col.lim = c(0, 100), number.digits = 2, main = "AIC", cl.pos = "n")

### 🎯 Taking it a step further with Rejection Sampling:

Now that we’ve generated both training and testing datasets under three evolutionary models (BM, OU, and EB), it’s time to bring **real data** into play.

To make our simulations biologically meaningful, we **filter** them using **rejection sampling** based on empirical metrics recovered from a **Fungal gene expression data**.

This ensures that simulated traits (used to train EvoDA) exhibit **variation comparable to empirical gene expression traits**, making downstream model classification more robust and interpretable.

|  |
| --- |
| 🧠 What is Rejection Sampling? |
| Rejection sampling is a filtering method where only data points that meet specific empirical criteria are retained. In our case, we compute the standard deviation of **phylogenetically independent contrasts (PICs)** from real gene expression data and use that standard deviation as a benchmark to select comparable simulations. |

First, download the gene\_expression\_tpm\_matrix\_updated\_Standard.LogNorm.tsv provided in the following directory: <https://github.com/radamsRHA/EvoDA/tree/main/EmpiricalExample>. This dataset can be sourced from the original publications:

Dimayacyac, J.R., Wu, S., Jiang, D. and Pennell, M., 2023. Evaluating the performance of widely used phylogenetic models for gene expression evolution. Genome Biology and Evolution. 15. Cope AL, O’Meara BC, Gilchrist MA. 2020. Gene expression of functionally-related genes coevolves across fungal species: detecting coevolution of gene expression using phylogenetic comparative methods. BMC Genomics. 21:1–17.

#######################  
# load input GEX data #  
#######################  
handle.GEX\_Data <- as\_tibble(read.table(file = '~/Desktop/gene\_expression\_tpm\_matrix\_updated\_Standard.LogNorm.tsv', header = T))  
  
###############################################  
# filter the GEX data based on missing values #  
###############################################  
handle.GEX\_Data <- handle.GEX\_Data %>% drop\_na() # remove any genes with NA  
  
############################  
# CHECK OVERLAP OF SPECIES #  
############################  
vector.KeepNames <- c(handle.TargetTree$tip.label, colnames(handle.GEX\_Data)[colnames(handle.GEX\_Data) != "Protein"])  
vector.KeepNames <- names(table(vector.KeepNames))[table(vector.KeepNames) ==2]  
  
#############################################  
# only keep species in both tree and traits #  
#############################################  
handle.GEX\_Data <- handle.GEX\_Data[,vector.KeepNames]  
handle.TargetTree <- drop.tip(handle.TargetTree,handle.TargetTree$tip.label[-match(vector.KeepNames, handle.TargetTree$tip.label)]) # match taxa names to phylogeny   
  
################  
# compute PICs #  
################  
handle.GEX\_PICs <- apply(t(handle.GEX\_Data), MARGIN = 2, FUN = function(x) pic(x = x, phy = handle.TargetTree)) # compute PICs using target trees  
handle.GEX\_PICs <- t(handle.GEX\_PICs)  
  
#####################################  
# Rename columns to match DATA\_TEST #  
#####################################  
colnames(handle.GEX\_PICs) <- colnames(DATA\_TEST)[-ncol(DATA\_TEST)]   
handle.GEX\_PICs <- as.data.frame(handle.GEX\_PICs) # convert to data.frame

📌 *Next step*: Use handle.GEX\_PICs to define empirical variability bounds, and apply rejection sampling to both training and testing simulations to ensure they fall within biologically plausible ranges.

### 🧠 EvoDA Model Training: Filtering Training Data Using Empirical GEX PICs

We can use the simulated training and testing datasets generated above and add a crucial step that bridges our empirical and simulated data: rejection sampling. This method helps align the variability of simulated traits with that observed in real gene expression (GEX) data, ensuring our discriminant models learn from biologically plausible inputs.

In this section, we compute **phylogenetically independent contrasts (PICs)** for each simulated trait in the training and testing data, calculate the expected standard deviation, and use that to filter our training data.

############################  
# Extract training dataset #  
############################  
handle.TRAINING\_DATA <- handle.RESULTS\_TRAIN$RESULTS\_PIC  
  
###########################################  
# let's get mean of means and sd of means #  
###########################################  
vector.MeanPIC\_GEX <- apply(X = handle.GEX\_PICs, FUN = sd, MARGIN = 1)  
vector.MeanPIC\_SIM <- apply(X = handle.TRAINING\_DATA[,-ncol(handle.TRAINING\_DATA)], FUN = sd, MARGIN = 1)  
  
##########################################################  
# Define upper and lower biological bounds (mean ± 3 SD) #  
##########################################################  
numeric.MeanPIC\_GEX <- mean(vector.MeanPIC\_GEX)  
numeric.SDPIC\_GEX <- sd(vector.MeanPIC\_GEX)  
numeric.UPPER <- numeric.MeanPIC\_GEX + (3\*numeric.SDPIC\_GEX)  
numeric.LOWER <- numeric.MeanPIC\_GEX - (3\*numeric.SDPIC\_GEX)  
  
#################################################  
# let's try to filter based on empirical bounds #  
#################################################  
handle.FILTERED\_TRAINING <- handle.TRAINING\_DATA  
handle.FILTERED\_TRAINING <- cbind(handle.FILTERED\_TRAINING, MeanPIC = vector.MeanPIC\_SIM)  
handle.FILTERED\_TRAINING <- handle.FILTERED\_TRAINING %>% filter(MeanPIC >= numeric.LOWER, MeanPIC <= numeric.UPPER)  
handle.FILTERED\_TRAINING <- handle.FILTERED\_TRAINING[,-ncol(handle.FILTERED\_TRAINING)]  
handle.FILTERED\_TRAINING$SimulationModelNumber <- as.factor(handle.FILTERED\_TRAINING$SimulationModelNumber)

|  |
| --- |
| 🧠 Why Filter Simulations? |
| This step ensures your training data resembles the variability seen in real gene expression, preventing the models from learning on unrealistic traits. |
| You can add more training/testing simulations on top, but repeating the process |

## 🧠 EvoDA Model Training with Filtered Data

We now train five discriminant models: LDA, QDA, FDA, MDA, and RDA using the filtered dataset. RDA is cross-validated with a grid of lambda values for optimal regularization.

#############################################  
# Train models using filtered training data #  
#############################################  
MODEL\_LDA <- lda(SimulationModelNumber ~., handle.FILTERED\_TRAINING, method = "mle")  
MODEL\_QDA <- qda(SimulationModelNumber ~., handle.FILTERED\_TRAINING, method = "mle")  
MODEL\_FDA <- fda(SimulationModelNumber ~., handle.FILTERED\_TRAINING, method = bruto)  
MODEL\_MDA <- mda(SimulationModelNumber ~., handle.FILTERED\_TRAINING, method = bruto)  
MODEL\_RDA <- train(SimulationModelNumber ~ ., data = handle.FILTERED\_TRAINING, method = "rda", trControl = trainControl(method = "cv", number = 10), tuneGrid = data.frame(gamma = 0, lambda = c(0, 0.2, 0.4, 0.6, 0.8, 1)), shuffle = T)

### 🤖 Predict Evolutionary Models on empirical gene expression data

#######################################################  
# Make predictions for empirical gene expression data #  
#######################################################  
PREDICT\_LDA <- factor(predict(MODEL\_LDA, handle.GEX\_PICs)$class)  
PREDICT\_QDA <- factor(predict(MODEL\_QDA, handle.GEX\_PICs)$class)  
PREDICT\_FDA <- factor(predict(MODEL\_FDA, handle.GEX\_PICs))  
PREDICT\_MDA <- factor(predict(MODEL\_MDA, handle.GEX\_PICs))  
PREDICT\_RDA <- factor(predict(MODEL\_RDA, handle.GEX\_PICs))

## 🔍 Training Set Evaluation (Confussion Matrices)

### 🧪 Test Models on Filtered Simulated Data

Once the training models are fitted using biologically plausible simulations (filtered via rejection sampling), we proceed to test them on a new filtered test dataset. This ensures that both training and testing reflect the variability observed in empirical gene expression profiles across fungal species.

We use the same rejection thresholds (±3 SD from the mean PIC) previously computed from the empirical gene expression data.

|  |
| --- |
| 🧠 Why filter again? |
| Reapplying the rejection criteria ensures that the testing data conforms to the same empirical trait variance constraints as the training data. This strengthens the generalizability of model performance on real-world biological data. |

############################  
# Extract training dataset #  
############################  
handle.TESTING\_DATA <- handle.RESULTS\_TEST$RESULTS\_PIC  
  
###############################  
# compute vector of MEAN PICs #  
###############################  
vector.MeanPIC\_SIM\_TEST <- apply(X = handle.TESTING\_DATA[,-ncol(handle.TESTING\_DATA)], FUN = sd, MARGIN = 1)  
  
########################################  
# let's try to filter the testing data #  
########################################  
handle.FILTERED\_TEST <- handle.TESTING\_DATA  
handle.FILTERED\_TEST <- cbind(handle.FILTERED\_TEST, MeanPIC = vector.MeanPIC\_SIM\_TEST)  
handle.FILTERED\_TEST <- handle.FILTERED\_TEST %>% filter(MeanPIC >= numeric.LOWER, MeanPIC <= numeric.UPPER)  
handle.FILTERED\_TEST <- handle.FILTERED\_TEST[,-ncol(handle.FILTERED\_TEST)]  
handle.FILTERED\_TEST$SimulationModelNumber <- as.factor(handle.FILTERED\_TEST$SimulationModelNumber)

### 🤖 Predict Evolutionary Models on Test Data.

💡 Tip: Note: accuracy will be poor due to the low number of training and testing datasets used in this example. In a real world scenario, we recommend at least 100,000 training replicates (post-filtering) and 1000 testing replicates (post-filtering) |

####################  
# Make predictions #  
####################  
PREDICT\_LDA <- factor(predict(MODEL\_LDA, handle.FILTERED\_TEST)$class)  
PREDICT\_QDA <- factor(predict(MODEL\_QDA, handle.FILTERED\_TEST)$class)  
PREDICT\_FDA <- factor(predict(MODEL\_FDA, handle.FILTERED\_TEST))  
PREDICT\_MDA <- factor(predict(MODEL\_MDA, handle.FILTERED\_TEST))  
PREDICT\_RDA <- factor(predict(MODEL\_RDA, handle.FILTERED\_TEST))  
  
table(handle.FILTERED\_TEST$SimulationModelNumber) # get table of results  
mean(PREDICT\_FDA == handle.FILTERED\_TEST$SimulationModelNumber) # accuracy may be high or low, but we need more training and testing replicates in a real analysis