Model Training: XGBOOST Classifier

Brain K. Masinde

```r
# Environment Cleaning: Clearing workspace
rm(list = ls())

# load packages
library(rpart)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
library(pROC) # For AUC calculation
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
library(mlflow)
library(reticulate)
library(Matrix)
library(purrr) # useful for code optimization
```

```
##
## Attaching package: 'purrr'

## The following object is masked from 'package:data.table':
##
##     transpose

## The following object is masked from 'package:caret':
##
##     lift
```

```r
library(themis)
```

```
## Loading required package: recipes

##
## Attaching package: 'recipes'

## The following object is masked from 'package:Matrix':
##
##     update

## The following object is masked from 'package:stats':
##
##     step
```

```r
library(doMC)
```

```
## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when

## Loading required package: iterators

## Loading required package: parallel
```

```r
library(here)
```

```
## here() starts at /Users/masinde/Projects/causal_fairness_Ph_IbF
```

Inputs

```r
base_train <- read.csv(here("data", "base_train.csv"))
base_validation <- read.csv(here("data", "base_validation.csv"))
base_test <- read.csv(here("data", "base_test.csv"))
```

```r
# Combining train and validation datasets to one
# Because we are going to use CV to train the models later
# naming it df_base_train to remain consistent with df naming
df_base_train  <- rbind(base_train, base_validation)

cat("number of rows in combined train data:", nrow(df_base_train), sep = " ")
```

```
## number of rows in combined train data: 7184
```

```r
# Ensure target variable is a factor
# Ensure the target variable is a factor with valid names
df_base_train$damage_binary_2 <- factor(df_base_train$damage_binary,
                                        levels = c("0", "1"),  # Your current levels
                                        labels = c("Damage_below_10", "Damage_above_10"))  # New valid l
```

Model Training using CV

```r
# Set up train control with custom seeds
n_folds <- 7
n_models <- 25  # adjust depending on search space size, affects seeds length


# Reproducibility: Defining seeds (a little bit complicated because of parallel processing)

#  Generate a reproducible list of seeds
set.seed(1234)

seeds_list <- vector(mode = "list", length = n_folds + 1)
for (i in 1:n_folds) {
  seeds_list[[i]] <- sample.int(1000000, n_models)  # one seed per model per fold
}
seeds_list[[n_folds + 1]] <- sample.int(1000000, 1)  # for final model



# Set up train control with 10-fold cross-validation
```

```r
train_control <- trainControl(
  method = "cv",
  number = n_folds, # number of cross-validation folds
  classProbs = TRUE,  # Needed for AUC calculation
  summaryFunction = twoClassSummary,
  sampling = "smote", # caret automatically identifies minority class
  search = "random", # random selection of the expanded grid
  seeds = seeds_list, # Ensures reproducibility
)

# Detect and register the number of available cores (use all but one)
num_cores <- parallel::detectCores() - 2
registerDoMC(cores = num_cores)  # Enable parallel processing


# Measure the time for a code block to run
system.time({
    # Train the model using grid search with 10-fold CV

    xgb_model <- train(damage_binary_2 ~ track_min_dist +
                       wind_max +
                       rain_total +
                       roof_strong_wall_strong +
                       roof_strong_wall_light +
                       roof_strong_wall_salv +
                       roof_light_wall_strong +
                       roof_light_wall_light +
                       roof_light_wall_salv +
                       roof_salv_wall_strong +
                       roof_salv_wall_light +
                       roof_salv_wall_salv +
                       blue_ss_frac +
                       yellow_ss_frac +
                       orange_ss_frac +
                       red_ss_frac +
                       blue_ls_frac +
                       yellow_ls_frac +
                       orange_ls_frac +
                       red_ls_frac,
        data = df_base_train,
        method = "xgbTree",
        trControl = train_control,
        tuneLength = n_models,  #  this replaces tuneGrid
        metric = "ROC" # "xgbTree" does not support other metrics for classification tasks (e.g., Kappa
    )
    Sys.sleep(2)  # This is just an example to simulate a delay
})
```

```
##    user  system elapsed
## 517.038   4.357  73.441
```

```r
# Print best parameters
print(xgb_model$bestTune)
```

```
##   nrounds max_depth       eta    gamma colsample_bytree min_child_weight
## 7     785         6 0.2356713 7.027949        0.6720632               10
##   subsample
## 7  0.540222
```

```r
# Extract the best parameters (remove AUC column)
best_params_model <- xgb_model$bestTune

damage_fit_class_full <- train(damage_binary_2 ~ track_min_dist +
                               wind_max +
                               rain_total +
                               roof_strong_wall_strong +
                               roof_strong_wall_light +
                               roof_strong_wall_salv +
                               roof_light_wall_strong +
                               roof_light_wall_light +
                               roof_light_wall_salv +
                               roof_salv_wall_strong +
                               roof_salv_wall_light +
                               roof_salv_wall_salv +
                               blue_ss_frac +
                               yellow_ss_frac +
                               orange_ss_frac +
                               red_ss_frac +
                               blue_ls_frac +
                               yellow_ls_frac +
                               orange_ls_frac +
                               red_ls_frac,
          data = df_base_train, # USE TRAINING AND VALIDATION SETS COMBINED
          method = "xgbTree", # XGBoost method
          trControl = trainControl(method = "none"),  # No automatic validation
          tuneGrid = best_params_model # USE BEST PARAMETER
        )
```

```r
# Sanity Check
# testing on the training datasets (training + validation)

## Outcome prediction on the final_training_df dataset
## default function predict returns class probabilities (has two columns)
y_pred <- predict(damage_fit_class_full,
                  newdata = df_base_train)


levels(y_pred)
```

```
## [1] "Damage_below_10" "Damage_above_10"
```

```r
# using table function
conf_matrix <- confusionMatrix(y_pred,
                     df_base_train$damage_binary_2, # remember to use damage_binary_2
                     positive = "Damage_above_10"
                     )
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Damage_below_10 Damage_above_10
##   Damage_below_10            6754             138
##   Damage_above_10              36             256
##
##                Accuracy : 0.9758
##                  95% CI : (0.972, 0.9792)
##     No Information Rate : 0.9452
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7339
##
##  Mcnemar's Test P-Value : 1.906e-14
##
##             Sensitivity : 0.64975
##             Specificity : 0.99470
##          Pos Pred Value : 0.87671
##          Neg Pred Value : 0.97998
##              Prevalence : 0.05484
##          Detection Rate : 0.03563
##    Detection Prevalence : 0.04065
##       Balanced Accuracy : 0.82222
##
##        'Positive' Class : Damage_above_10
##
```

```r
accuracy <- conf_matrix$overall['Accuracy']

cat("test-set accuracy of associational XGBOOST:", accuracy, sep = " ")
```

```
## test-set accuracy of associational XGBOOST: 0.9757795
```

Model - Mlflow

```r
# Logging the model and parameter using MLflow

# set tracking URI
mlflow_set_tracking_uri("http://127.0.0.1:5000")

# Ensure any active run is ended
suppressWarnings(try(mlflow_end_run(), silent = TRUE))

# set experiment
# Logging metrics for model training and the parameters used
mlflow_set_experiment(experiment_name = "Attempt 2: R ass-XGBOOST classification - CV (Training metircs)
```

```
## [1] "345587670307584911"
```

```r
# Ensure that MLflow has only one run. Start MLflow run once.
run_name <- paste("XGBoost Run", Sys.time())  # Unique name using current time


# Start MLflow run
mlflow_start_run(nested = FALSE)
```

```
## Warning: 'as_integer()' is deprecated as of rlang 0.4.0
## Please use 'vctrs::vec_cast()' instead.
## This warning is displayed once every 8 hours.
```

```
## # A tibble: 1 x 13
##   run_uuid             experiment_id run_name user_id status start_time
##   <chr>                <chr>         <chr>    <chr>   <chr>  <dttm>
## 1 8e8958d7ecb54f0cbf0~ 345587670307~ shiveri~ masinde RUNNI~ 2025-07-24 14:08:43
## # i 7 more variables: artifact_uri <chr>, lifecycle_stage <chr>, run_id <chr>,
## #   end_time <lgl>, metrics <lgl>, params <lgl>, tags <list>
```

```r
# Ensure the run ends even if an error occurs
#on.exit(mlflow_end_run(), add = TRUE)

# Extract the best parameters (remove AUC column)
best_params_model <- xgb_model$bestTune

# Log each of the best parameters in MLflow
for (param in names(best_params_model)) {
  mlflow_log_param(param, best_params_model[[param]])
}

# Log the model type as a parameter
mlflow_log_param("model_type", "ass-xgboost-classification")


# summarize results
conf_matrix <- confusionMatrix(y_pred,
                     df_base_train$damage_binary_2,
                     positive = "Damage_above_10"
                     )

# accuracy
accuracy  <- conf_matrix$overall['Accuracy']

# Positive class = 1, precision, recall, and F1
# Extract precision, recall, and F1 score
precision <- conf_matrix$byClass['Precision']
recall <- conf_matrix$byClass['Recall']
f1_score <- conf_matrix$byClass['F1']


# Log parameters and metrics
# mlflow_log_param("model_type", "scm-xgboost-classification")
mlflow_log_metric("accuracy", accuracy)
```

```
## Warning: `as_double()` is deprecated as of rlang 0.4.0
## Please use `vctrs::vec_cast()` instead.
## This warning is displayed once every 8 hours.
```

```r
mlflow_log_metric("F1", f1_score)
mlflow_log_metric("Precision", precision)
mlflow_log_metric("Recall", recall)


# Save model
#saveRDS(model, file = file.path(path_2_folder, "spam_clas_model.rds"))

# End MLflow run
mlflow_end_run()
```

```
## # A tibble: 1 x 13
##   run_uuid             experiment_id run_name user_id status start_time
##   <chr>                <chr>         <chr>    <chr>   <chr>  <dttm>
## 1 8e8958d7ecb54f0cbf0~ 345587670307~ shiveri~ masinde FINIS~ 2025-07-24 14:08:43
## # i 7 more variables: end_time <dttm>, artifact_uri <chr>,
## #   lifecycle_stage <chr>, run_id <chr>, metrics <list>, params <list>,
## #   tags <list>
```

Model Testing: Testing on out of sample test data.

```r
# Testing on test dataset ------------------------------------------------------------------------------
base_test$damage_binary_2 <- factor(base_test$damage_binary,
                                    levels = c("0", "1"),  # Your current levels
                                    labels = c("Damage_below_10", "Damage_above_10"))  # New valid l

# predict for damage_binary
# Make probability predictions for classification
y_preds_probs <- predict(damage_fit_class_full,
                         newdata = base_test, type = "prob")[,2]  # Probability of class 1

# AUC
# Compute AUC (better for classification)
auc_value <- auc(roc(base_test$damage_binary_2, y_preds_probs))
```

```
## Setting levels: control = Damage_below_10, case = Damage_above_10
```

```
## Setting direction: controls < cases
```

```r
auc_value
```

```
## Area under the curve: 0.936
```

```r
## assigning final class based on threshold
threshold = 0.3
y_pred <- ifelse(y_preds_probs > threshold, 1, 0)

y_pred  <- factor(y_pred, levels = c("0", "1"),  # Your current levels
                                       labels = c("Damage_below_10", "Damage_above_10"))  # New valid l
# using table function
conf_matrix <- confusionMatrix(as.factor(y_pred),
                     base_test$damage_binary_2,
                     positive = "Damage_above_10"
                     )
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Damage_below_10 Damage_above_10
##   Damage_below_10            1649              40
##   Damage_above_10              47              61
##
##                Accuracy : 0.9516
##                  95% CI : (0.9406, 0.961)
##     No Information Rate : 0.9438
##     P-Value [Acc > NIR] : 0.08107
##
##                   Kappa : 0.5581
##
##  Mcnemar's Test P-Value : 0.52005
##
##             Sensitivity : 0.60396
##             Specificity : 0.97229
##          Pos Pred Value : 0.56481
##          Neg Pred Value : 0.97632
##              Prevalence : 0.05620
##          Detection Rate : 0.03395
##    Detection Prevalence : 0.06010
##       Balanced Accuracy : 0.78812
##
##        'Positive' Class : Damage_above_10
##
```

```r
cat("Precision of Positive class: Damage above 10:", conf_matrix$byClass['Precision'], sep = " ")
```

```
## Precision of Positive class: Damage above 10: 0.5648148
```

```r
cat("Recall of postive class: Damage above 10:", conf_matrix$byClass['Recall'], sep = " ")
```

```
## Recall of postive class: Damage above 10: 0.6039604
```

```r
cat("F1 score of positive class: Damage above 10:", conf_matrix$byClass['F1'], sep = " ")
```

```
## F1 score of positive class: Damage above 10: 0.5837321
```

Logging Test metrics - Mlflow

```
# set tracking URI
mlflow_set_tracking_uri("http://127.0.0.1:5000")

# Ensure any active run is ended
suppressWarnings(try(mlflow_end_run(), silent = TRUE))

# set experiment
# Logging metrics for model training and the parameters used
mlflow_set_experiment(experiment_name = "Attempt 2: R ass- XGBOOST classification - CV (Test metircs)")
```

```
## [1] "244452174007912648"
```

```
# Ensure that MLflow has only one run. Start MLflow run once.
run_name <- paste("XGBoost Run", Sys.time())  # Unique name using current time


# Start MLflow run
mlflow_start_run(nested = FALSE)
```

```
## # A tibble: 1 x 13
##   run_uuid             experiment_id run_name user_id status start_time
##   <chr>                <chr>         <chr>    <chr>   <chr>  <dttm>
## 1 459c684fb3a44ba4ada~ 244452174007~ gentle-~ masinde RUNNI~ 2025-07-24 14:08:43
## # i 7 more variables: artifact_uri <chr>, lifecycle_stage <chr>, run_id <chr>,
## #   end_time <lgl>, metrics <lgl>, params <lgl>, tags <list>
```

```
# Ensure the run ends even if an error occurs
#on.exit(mlflow_end_run(), add = TRUE)

# Extract the best parameters (remove AUC column)
#best_params_model <- best_params %>% # Remove AUC column if present
#    select(-AUC)

parameters_used  <- xgb_model$bestTune

# Log each of the best parameters in MLflow
for (param in names(parameters_used)) {
  mlflow_log_param(param, parameters_used[[param]])
}

# Log the model type as a parameter
mlflow_log_param("model_type", "ass-xgboost-classification")

# predicting
#   Threshold
threshold = 0.3
y_preds_probs <- predict(damage_fit_class_full, newdata = base_test, type = "prob")[,2]  # Probability
y_pred <- ifelse(y_preds_probs > threshold, 1, 0)
```

```r
y_pred  <- factor(y_pred, levels = c("0", "1"),  # Your current levels
                                        labels = c("Damage_below_10", "Damage_above_10"))  # New valid l

# summarize results
conf_matrix <- confusionMatrix(as.factor(y_pred),
                     base_test$damage_binary_2,
                     positive = "Damage_above_10"
                     )

# accuracy
accuracy  <- conf_matrix$overall['Accuracy']

# Positive class = 1, precision, recall, and F1
# Extract precision, recall, and F1 score
precision <- conf_matrix$byClass['Precision']
recall <- conf_matrix$byClass['Recall']
f1_score <- conf_matrix$byClass['F1']
auc_value <- auc(roc(base_test$damage_binary_2, y_preds_probs))
```

```
## Setting levels: control = Damage_below_10, case = Damage_above_10

## Setting direction: controls < cases
```

```r
# Log parameters and metrics
# mlflow_log_param("model_type", "scm-xgboost-classification")
mlflow_log_metric("accuracy", accuracy)
mlflow_log_metric("F1", f1_score)
mlflow_log_metric("Precision", precision)
mlflow_log_metric("Recall", recall)
#mlflow_log_metric("AUC", auc_value)


# Save model
#saveRDS(model, file = file.path(path_2_folder, "spam_clas_model.rds"))

# End MLflow run
mlflow_end_run()
```

```
## # A tibble: 1 x 13
##   run_uuid             experiment_id run_name user_id status start_time
##   <chr>                <chr>         <chr>    <chr>   <chr>  <dttm>
## 1 459c684fb3a44ba4ada~ 244452174007~ gentle-~ masinde FINIS~ 2025-07-24 14:08:43
## # i 7 more variables: end_time <dttm>, artifact_uri <chr>,
## #   lifecycle_stage <chr>, run_id <chr>, metrics <list>, params <list>,
## #   tags <list>
```

Misc. Experiments

```r
# test set confusion matrix
conf_matrix
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Damage_below_10 Damage_above_10
##   Damage_below_10            1649              40
##   Damage_above_10              47              61
##
##                Accuracy : 0.9516
##                  95% CI : (0.9406, 0.961)
##     No Information Rate : 0.9438
##     P-Value [Acc > NIR] : 0.08107
##
##                   Kappa : 0.5581
##
##  Mcnemar's Test P-Value : 0.52005
##
##             Sensitivity : 0.60396
##             Specificity : 0.97229
##          Pos Pred Value : 0.56481
##          Neg Pred Value : 0.97632
##              Prevalence : 0.05620
##          Detection Rate : 0.03395
##    Detection Prevalence : 0.06010
##       Balanced Accuracy : 0.78812
##
##        'Positive' Class : Damage_above_10
##
```

```r
# get the false positves by regions (Luzon, Visayas, Mindanao)
# confusion matrix by regions
# Make sure the grouping variable is a factor
# Make sure island_groups is a factor
base_test$island_groups <- as.factor(base_test$island_groups)

# Loop through each group and generate a confusion matrix
for (grp in levels(base_test$island_groups)) {

  # Subset data for the current group
  group_indices <- base_test$island_groups == grp
  y_true_group <- base_test$damage_binary_2[group_indices]
  y_pred_group <- y_pred[group_indices]

  # Generate and print confusion matrix
  cat("Confusion Matrix for Island Group:", grp, "\n")
  print(confusionMatrix(y_pred_group, y_true_group, positive = "Damage_above_10"))
  cat("\n")
}
```

```
## Confusion Matrix for Island Group: Luzon
```

```
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Damage_below_10 Damage_above_10
##   Damage_below_10            1188              28
##   Damage_above_10              22              30
##
##                Accuracy : 0.9606
##                  95% CI : (0.9483, 0.9706)
##     No Information Rate : 0.9543
##     P-Value [Acc > NIR] : 0.1565
##
##                   Kappa : 0.5249
##
##  Mcnemar's Test P-Value : 0.4795
##
##             Sensitivity : 0.51724
##             Specificity : 0.98182
##          Pos Pred Value : 0.57692
##          Neg Pred Value : 0.97697
##              Prevalence : 0.04574
##          Detection Rate : 0.02366
##    Detection Prevalence : 0.04101
##       Balanced Accuracy : 0.74953
##
##        'Positive' Class : Damage_above_10
##
##
## Confusion Matrix for Island Group: Mindanao
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Damage_below_10 Damage_above_10
##   Damage_below_10             103               3
##   Damage_above_10               1               3
##
##                Accuracy : 0.9636
##                  95% CI : (0.9095, 0.99)
##     No Information Rate : 0.9455
##     P-Value [Acc > NIR] : 0.2775
##
##                   Kappa : 0.5817
##
##  Mcnemar's Test P-Value : 0.6171
##
##             Sensitivity : 0.50000
##             Specificity : 0.99038
##          Pos Pred Value : 0.75000
##          Neg Pred Value : 0.97170
##              Prevalence : 0.05455
##          Detection Rate : 0.02727
##    Detection Prevalence : 0.03636
##       Balanced Accuracy : 0.74519
##
```

```
##        'Positive' Class : Damage_above_10
##
##
## Confusion Matrix for Island Group: Visayas
## Confusion Matrix and Statistics
##
##                  Reference
## Prediction        Damage_below_10 Damage_above_10
##   Damage_below_10             358               9
##   Damage_above_10              24              28
##
##                Accuracy : 0.9212
##                  95% CI : (0.8912, 0.9452)
##     No Information Rate : 0.9117
##     P-Value [Acc > NIR] : 0.27848
##
##                   Kappa : 0.5866
##
##  Mcnemar's Test P-Value : 0.01481
##
##             Sensitivity : 0.75676
##             Specificity : 0.93717
##          Pos Pred Value : 0.53846
##          Neg Pred Value : 0.97548
##              Prevalence : 0.08831
##          Detection Rate : 0.06683
##    Detection Prevalence : 0.12411
##       Balanced Accuracy : 0.84696
##
##        'Positive' Class : Damage_above_10
##
```

Outputs

```r
saveRDS(damage_fit_class_full, here("associational XGBOOST", "ass_XGBOOST_class.rds"))
```

OLD CODE