Training Classification Model (Unadjusted SCM)

Brian K. Masinde

```
# Environment: Cleaning environment
rm(list = ls())

# Libraries: Load
library(rpart)
library(dplyr)
library(caret)
library(pROC) # For AUC calculation
library(data.table)
library(mlflow)
library(reticulate)
library(Matrix)
library(purrr) # useful for code optimization
library(themis)
library(doMC)
library(here)
```

Data inputs

```
# inputs
base_train <- read.csv(here("data", "base_train.csv"))
base_validation <- read.csv(here("data", "base_validation.csv"))

# Combining train and validation datasets to one
# Because we are going to use CV to train the models later
# naming it df_base_train2 to remain consistent with df naming
df_base_train2  <- rbind(base_train, base_validation)

cat("number of rows in combined train data:", nrow(df_base_train2), sep = " ")
```

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

```
# Training structural equation for wind speed
# wind_speed = f(track_min_dist, eps)


base_wind_model <- readRDS(here("unadjusted SCM/new base models",
                                "dec_base_wind_model_tuned.rds"))
```

```r
# Training structural equation for wind speed
# wind_speed = f(track_min_dist, eps)

base_rain_model <- readRDS(here("unadjusted SCM/new base models",
                                "dec_base_rain_model_tuned.rds"))
```

```r
# Interaction Terms: storm surge and landslide fraction variables are mediators (without moderation)
# Define wind and rain interaction variables
# Define wind and rain interaction variables
wind_fractions <- c("blue_ss_frac", "yellow_ss_frac", "orange_ss_frac", "red_ss_frac")
rain_fractions <- c("blue_ls_frac", "yellow_ls_frac", "orange_ls_frac", "red_ls_frac")

# Predict using model "base_wind_model"
# To get variable: wind_max_pred
df_base_train2[["wind_max_pred"]] <- predict(base_wind_model, newdata = df_base_train2)


# Predict using model "base_rain_model"
# To get variable: wind_max_pred
df_base_train2[["rain_total_pred"]] <- predict(base_rain_model, newdata = df_base_train2)


# Multiply wind fractions by wind_max_pred
for (col in wind_fractions) {
  print(col)
  new_col_name <- paste0("wind_", col)
  df_base_train2[[new_col_name]] <- df_base_train2[[col]] * df_base_train2[["wind_max_pred"]]
  # output for example wind_blue_ss_frac
}
```

```
## [1] "blue_ss_frac"
## [1] "yellow_ss_frac"
## [1] "orange_ss_frac"
## [1] "red_ss_frac"
```

```r
# Multiply rain fractions by rain_total_pred
for (col in rain_fractions) {
  new_col_name <- paste0("rain_", col)
  df_base_train2[[new_col_name]] <- df_base_train2[[col]] * df_base_train2[["rain_total_pred"]]
}
```

```r
# Ensure target variable is a factor
# Ensure the target variable is a factor with valid names

#df_base_train2$damage_binary <- as.factor(df_base_train2$damage_binary)

df_base_train2$damage_binary_2 <- factor(df_base_train2$damage_binary,
                                         levels = c("0", "1"),  # Your current levels
                                         labels = c("Damage_below_10", "Damage_above_10"))  # New valid l
```

Model training

```
# 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
train_control <- trainControl(
  method = "cv",
  number = n_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
)

# 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_pred +
        rain_total_pred +
        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 +
        wind_blue_ss_frac +
```

```
        wind_yellow_ss_frac +
        wind_orange_ss_frac +
        wind_red_ss_frac +
        rain_blue_ls_frac +
        rain_yellow_ls_frac +
        rain_orange_ls_frac +
        rain_red_ls_frac,
        data = df_base_train2,
        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
## 512.520   3.277  67.324
```

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

```
##   nrounds max_depth        eta  gamma colsample_bytree min_child_weight
## 1     100         6 0.02113197 3.9941        0.5383644               14
##   subsample
## 1 0.7606618
```

```
xgb_model$bestTune
```

```
##   nrounds max_depth        eta  gamma colsample_bytree min_child_weight
## 1     100         6 0.02113197 3.9941        0.5383644               14
##   subsample
## 1 0.7606618
```

```
# Training based on tuned parameters

# Combine Training and Validation datasets for final training

#final_training_df  <- rbind(df_base_train,
#                            df_base_validation)


# 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_pred +
            rain_total_pred +
            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 +
            wind_blue_ss_frac +
            wind_yellow_ss_frac +
            wind_orange_ss_frac +
            wind_red_ss_frac +
            rain_blue_ls_frac +
            rain_yellow_ls_frac +
            rain_orange_ls_frac +
            rain_red_ls_frac,
          data = df_base_train2, # USE TRAINING AND VALIDATION SETS COMBINED
          method = "xgbTree", # XGBoost method
          trControl = trainControl(method = "none"),  # No automatic validation
          tuneGrid = best_params_model # USE BEST PARAMETER
        )
```

```
# 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_train2)
```

```
levels(y_pred)
```

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

```
# using table function
conf_matrix <- confusionMatrix(y_pred,
                     df_base_train2$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            6763             325
##   Damage_above_10              27              69
##
##                Accuracy : 0.951
##                  95% CI : (0.9458, 0.9559)
##     No Information Rate : 0.9452
##     P-Value [Acc > NIR] : 0.01463
##
```

```
##                  Kappa : 0.2659
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.175127
##             Specificity : 0.996024
##          Pos Pred Value : 0.718750
##          Neg Pred Value : 0.954148
##              Prevalence : 0.054844
##          Detection Rate : 0.009605
##    Detection Prevalence : 0.013363
##       Balanced Accuracy : 0.585575
##
##        'Positive' Class : Damage_above_10
##
```

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

cat("test-set accuracy of minimal SCM model:", accuracy, sep = " ")
```

```
## test-set accuracy of minimal SCM model: 0.9510022
```

```
# 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: U-SCM - XGBOOST classification - CV (Training metir
```

```
## [1] "990346017975399620"
```

```
# 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 4118454882a0463d911~ 990346017975~ bald-mi~ masinde RUNNI~ 2025-07-24 14:52:44
## # 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", "R-undaj-scm-xgboost-classification")



# summarize results
conf_matrix <- confusionMatrix(y_pred,
                    df_base_train2$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 4118454882a0463d911~ 990346017975~ bald-mi~ masinde FINIS~ 2025-07-24 14:52:44
## # i 7 more variables: end_time <dttm>, artifact_uri <chr>,
## #   lifecycle_stage <chr>, run_id <chr>, metrics <list>, params <list>,
## #   tags <list>
```

Output

Save the trained classifier model

```
# Saving trained XGBOOST model
full_path <- here("unadjusted SCM/new base models")
saveRDS(damage_fit_class_full, file = file.path(full_path, paste0("damage_fit_class_full", ".rds")))
```

OLD CODE

```
# # setting seed for reproducibility
# set.seed(1234)
#
# tune_grid <- expand.grid(
#   nrounds = c(47,50, 60,70), # early stopping does not work, we still need to specify nrounds
#   max_depth = c(2, 3, 4, 6),
#   eta = c(0.09, 0.1, 0.11, 0.12),
#   gamma = c(0, 1, 2, 3, 4),
#   colsample_bytree = c(0.9, 1.0, 1.1),
#   min_child_weight = c(2, 3, 4),
#   subsample = c(0.5, 0.6, 0.7, 0.8)
# )
```