599_project_XGBoost

Group 2 - Qianzhi Bao, Rongxuan Qu, Weisen Xia, Kedan Xin

2023-04-23

```
spc_tbl <- read.csv("MICHD.csv")
```

```
str(spc_tbl)
```

```
## 'data.frame':    184877 obs. of  20 variables:
##  $ Urban           : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ PhysicalHealth  : num  0 0 0 0 0 ...
##  $ MentalHealth    : num  0 1.79 0 0 1.61 ...
##  $ PhysicalActivity: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Asthma          : int  3 3 3 3 1 3 3 3 3 3 ...
##  $ Arthritis       : int  0 0 0 0 1 0 0 1 0 1 ...
##  $ Sex             : int  0 0 1 0 0 1 1 1 1 0 ...
##  $ Race            : int  1 1 1 1 1 4 1 1 1 1 ...
##  $ Age             : int  2 3 5 5 6 3 6 5 5 4 ...
##  $ BMI             : num  25.6 29.5 31 35.4 25.7 ...
##  $ Education       : int  4 4 3 3 2 4 4 4 4 1 ...
##  $ Smoke           : int  4 4 4 4 4 1 3 4 4 4 ...
##  $ Alcohol         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ HHADULT         : num  0.693 0.693 0.693 0.693 0.693 ...
##  $ HIV             : int  1 0 0 0 0 0 0 0 0 1 ...
##  $ Fruit           : num  0.693 0.358 0.405 0.451 1.418 ...
##  $ Vegatable       : num  1.061 0.963 1.051 0.871 1.188 ...
##  $ Cholesterol     : int  0 0 1 0 1 0 1 1 1 0 ...
##  $ Stroke          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ MICHD           : int  0 0 0 0 0 0 1 0 0 0 ...
```

```r
library(xgboost)
library(dplyr)

# as.factor
cols_to_factor <- c("Urban", "PhysicalActivity", "Asthma", "Arthritis", "Sex", "Race", "Age",
                    "Education", "Alcohol", "Smoke", "HIV", "Cholesterol", "Stroke", "MICHD")
spc_tbl[cols_to_factor] <- lapply(spc_tbl[cols_to_factor], factor)

# model.matrix() onehot
encoded_data <- model.matrix(MICHD ~ ., data = spc_tbl)

## scale
encoded_data[,'BMI'] <- scale(encoded_data[,'BMI'])
encoded_data[,'HHADULT'] <- scale(encoded_data[,'HHADULT'])
encoded_data[,'Fruit'] <- scale(encoded_data[,'Fruit'])
encoded_data[,'Vegatable'] <- scale(encoded_data[,'Vegatable'])

# train and test
set.seed(123)
train_index <- sample(1:nrow(encoded_data), 0.7 * nrow(encoded_data))
train_data <- encoded_data[train_index, ]
test_data <- encoded_data[-train_index, ]



# features and label
train_label <- as.numeric(spc_tbl$MICHD[train_index] == 1)
train_features <- train_data
test_label <- as.numeric(spc_tbl$MICHD[-train_index] == 1)
test_features <- test_data

# set XGBoost, scale_pos_weight (weight)
## # cross-validation to select best model with best AUC
params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",  # maximize AUC
  max_depth = 6,
  eta = 0.3,
  subsample = 0.8,
  colsample_bytree = 0.8,
  scale_pos_weight = sum(train_label == 0) / sum(train_label == 1) # weight
)

# train

# cross-validation to select best model with best AUC
xgb_cv_result <- xgb.cv(
  params = params,
  data = train_features,
  label = train_label,
  nrounds = 100,
  nfold = 5,
  print_every_n = 10,
  early_stopping_rounds = 20,
  maximize = TRUE
)
```

```
## [1]  train-auc:0.798967+0.008917 test-auc:0.791799+0.010414
## Multiple eval metrics are present. Will use test_auc for early stopping.
## Will train until test_auc hasn't improved in 20 rounds.
##
## [11] train-auc:0.847367+0.001346 test-auc:0.825942+0.006683
## [21] train-auc:0.860150+0.001658 test-auc:0.824652+0.006473
## Stopping. Best iteration:
## [10] train-auc:0.845617+0.001409 test-auc:0.826109+0.006634
```

```
# Get the optimal number of rounds
best_nrounds <- xgb_cv_result$best_iteration
cat("Best number of rounds:", best_nrounds, "\n")
```

```
## Best number of rounds: 10
```

```
# Train the final model with the optimal number of rounds
xgb_model <- xgboost(
  params = params,
  data = train_features,
  label = train_label,
  nrounds = best_nrounds,
  print_every_n = 10
)
```

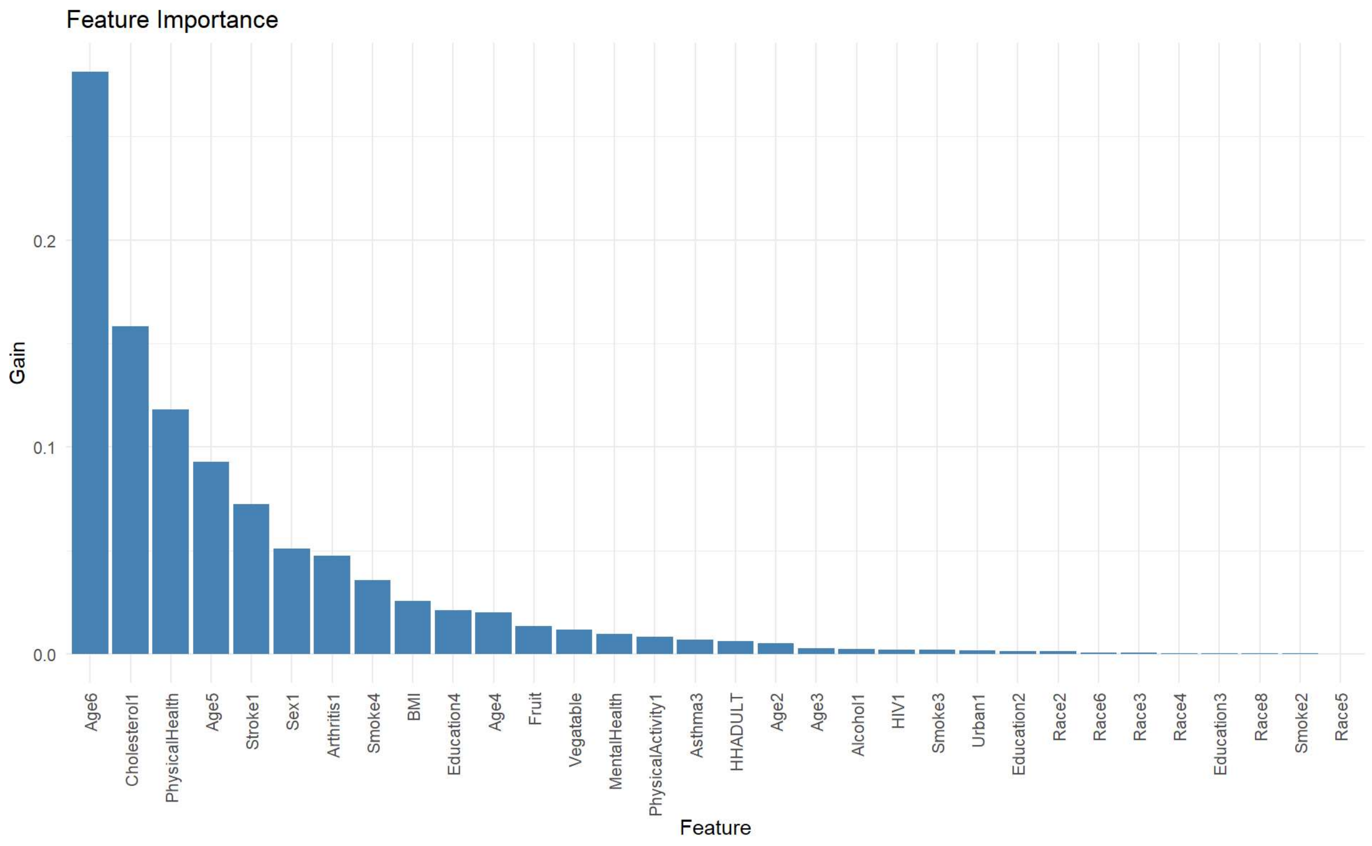
```
## [1]  train-auc:0.811008
## [10] train-auc:0.843997
```

```
# Extract feature importance
importance_matrix <- xgb.importance(feature_names = colnames(train_features), model = xgb_mod
el)

# Print and plot the feature importance
library(ggplot2)
print(importance_matrix)
```

```
##                Feature         Gain        Cover   Frequency
##  1:               Age6 2.812531e-01 1.211522e-01 0.028239203
##  2:       Cholesterol1 1.582731e-01 1.083747e-01 0.038205980
##  3:     PhysicalHealth 1.180160e-01 1.127786e-01 0.073089701
##  4:               Age5 9.283047e-02 4.445408e-02 0.013289037
##  5:            Stroke1 7.246905e-02 1.035406e-01 0.058139535
##  6:               Sex1 5.082193e-02 8.897892e-02 0.069767442
##  7:         Arthritis1 4.753298e-02 5.313897e-02 0.024916944
##  8:             Smoke4 3.556497e-02 5.582449e-02 0.039867110
##  9:                BMI 2.564581e-02 7.050077e-02 0.161129568
## 10:         Education4 2.098950e-02 4.437225e-02 0.041528239
## 11:               Age4 1.991028e-02 2.089263e-02 0.016611296
## 12:              Fruit 1.341767e-02 2.245900e-02 0.106312292
## 13:          Vegatable 1.162494e-02 2.492855e-02 0.089700997
## 14:       MentalHealth 9.611257e-03 2.145402e-02 0.041528239
## 15: PhysicalActivity1 8.125326e-03 1.724402e-02 0.021594684
## 16:            Asthma3 6.852381e-03 1.546390e-02 0.026578073
## 17:            HHADULT 6.104801e-03 1.678867e-02 0.028239203
## 18:               Age2 5.264169e-03 1.044400e-02 0.014950166
## 19:               Age3 2.629620e-03 1.103027e-02 0.011627907
## 20:           Alcohol1 2.397059e-03 1.282860e-02 0.016611296
## 21:               HIV1 2.109092e-03 4.769436e-03 0.008305648
## 22:             Smoke3 1.975867e-03 7.428940e-03 0.014950166
## 23:             Urban1 1.523692e-03 8.531566e-04 0.011627907
## 24:         Education2 1.262907e-03 1.530936e-03 0.008305648
## 25:              Race2 1.236998e-03 2.536946e-03 0.008305648
## 26:              Race6 6.346876e-04 1.546920e-03 0.006644518
## 27:              Race3 5.256794e-04 6.183708e-04 0.001661130
## 28:              Race4 4.674990e-04 1.565836e-03 0.004983389
## 29:         Education3 3.582567e-04 4.190558e-04 0.006644518
## 30:              Race8 3.490674e-04 6.043911e-04 0.003322259
## 31:             Smoke2 1.942835e-04 1.455705e-03 0.001661130
## 32:              Race5 2.755644e-05 2.116459e-05 0.001661130
##                Feature         Gain        Cover   Frequency
```

```
ggplot(importance_matrix, aes(x = reorder(Feature, -Gain), y = Gain)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1)) +
  labs(title = "Feature Importance", x = "Feature", y = "Gain")
```



```
## Compute Accuracy,  TPR,  FPR,  Test AUC

# predict probs
set.seed(123)
predicted_probs <- predict(xgb_model, test_features)



library(ROCR)

# Calculate AUC
pred_obj <- prediction(predicted_probs, test_label)
perf_obj <- performance(pred_obj, measure = "tpr", x.measure = "fpr")
auc_obj <- performance(pred_obj, measure = "auc")
auc <- as.numeric(auc_obj@y.values)
cat("Test AUC:", auc, "\n")
```

```
## Test AUC: 0.8330303
```

```
# Find the best threshold according to Youden's Index (J statistic)
cutoffs <- data.frame(cut = perf_obj@alpha.values[[1]],
                      tpr = perf_obj@y.values[[1]],
                      fpr = perf_obj@x.values[[1]])
cutoffs$optimal <- cutoffs$tpr - cutoffs$fpr
best_threshold <- cutoffs[which.max(cutoffs$optimal), "cut"]
cat("Best Threshold:", best_threshold, "\n")
```

```
## Best Threshold: 0.4630981
```

```
# Use the best threshold to predict
predicted_values <- ifelse(predicted_probs > best_threshold, 1, 0)

# TPR
true_positives <- sum(predicted_values == 1 & test_label == 1)
false_negatives <- sum(predicted_values == 0 & test_label == 1)
tpr <- true_positives / (true_positives + false_negatives)
print(paste("True Positive Rate (TPR) with best threshold:", tpr))
```

```
## [1] "True Positive Rate (TPR) with best threshold: 0.833083083083083"
```

```
# Accuracy
true_negatives <- sum(predicted_values == 0 & test_label == 0)
false_positives <- sum(predicted_values == 1 & test_label == 0)
accuracy <- (true_positives + true_negatives) / length(test_label)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.6980564
```

```
# FPR
fpr <- false_positives / (false_positives + true_negatives)
cat("False Positive Rate (FPR):", fpr, "\n")
```

```
## False Positive Rate (FPR): 0.3124271
```

```
set.seed(123)
# Function to train and evaluate model with selected features
## According to Accuracy
train_and_evaluate <- function(selected_features) {
  # Select the top features
  train_selected <- train_features[, colnames(train_features) %in% selected_features]
  test_selected <- test_features[, colnames(test_features) %in% selected_features]

  # Train the model with selected features
  xgb_model_selected <- xgboost(
    params = params,
    data = train_selected,
    label = train_label,
    nrounds = best_nrounds,
    print_every_n = 10,
  )

  # Predict the test set
  preds_selected <- predict(xgb_model_selected, test_selected)

  # Calculate the accuracy
  preds_class_selected <- as.numeric(preds_selected > best_threshold)
  accuracy_selected <- sum(preds_class_selected == test_label) / length(test_label)

  return(accuracy_selected)
}

# Sort features by importance
sorted_features <- importance_matrix$Feature[order(importance_matrix$Gain, decreasing = TRU
E)]

# Initialize variables to store the best accuracy and corresponding features
best_accuracy <- 0
best_features <- c()

# Iterate through the sorted features and train models with the top k features
for (k in 2:(length(sorted_features)-1)) {
  selected_features <- sorted_features[1:k]
  accuracy_selected <- train_and_evaluate(selected_features)

  # If the accuracy is higher than the best accuracy so far, update the best accuracy and bes
t features
  if (accuracy_selected > best_accuracy) {
    best_accuracy <- accuracy_selected
    best_features <- selected_features
  }
}
```

```
## [1]  train-auc:0.664204
## [10] train-auc:0.728547
## [1]  train-auc:0.735415
## [10] train-auc:0.777991
## [1]  train-auc:0.723866
## [10] train-auc:0.791732
## [1]  train-auc:0.779952
## [10] train-auc:0.804244
## [1]  train-auc:0.744999
## [10] train-auc:0.813201
## [1]  train-auc:0.799885
## [10] train-auc:0.822434
## [1]  train-auc:0.803227
## [10] train-auc:0.827957
## [1]  train-auc:0.757369
## [10] train-auc:0.832901
## [1]  train-auc:0.816403
## [10] train-auc:0.835327
## [1]  train-auc:0.815863
## [10] train-auc:0.836918
## [1]  train-auc:0.811111
## [10] train-auc:0.838990
## [1]  train-auc:0.799721
## [10] train-auc:0.840090
## [1]  train-auc:0.811553
## [10] train-auc:0.840777
## [1]  train-auc:0.789042
## [10] train-auc:0.841338
## [1]  train-auc:0.806682
## [10] train-auc:0.841366
## [1]  train-auc:0.809407
## [10] train-auc:0.841882
## [1]  train-auc:0.803848
## [10] train-auc:0.841756
## [1]  train-auc:0.802559
## [10] train-auc:0.842348
## [1]  train-auc:0.809278
## [10] train-auc:0.842013
## [1]  train-auc:0.783142
## [10] train-auc:0.842378
## [1]  train-auc:0.803863
## [10] train-auc:0.842932
## [1]  train-auc:0.793783
## [10] train-auc:0.843525
## [1]  train-auc:0.790222
## [10] train-auc:0.842916
## [1]  train-auc:0.798689
## [10] train-auc:0.843636
## [1]  train-auc:0.810921
## [10] train-auc:0.842726
## [1]  train-auc:0.810609
## [10] train-auc:0.843297
## [1]  train-auc:0.818076
## [10] train-auc:0.842608
## [1]  train-auc:0.804501
## [10] train-auc:0.843275
## [1]  train-auc:0.818228
## [10] train-auc:0.842869
## [1]  train-auc:0.789232
## [10] train-auc:0.842857
```

```
# Print the best features and best accuracy
cat("Best Features:\n")
```

```
## Best Features:
```

```
print(best_features)
```

```
##  [1] "Age6"          "Cholesterol1"  "PhysicalHealth" "Age5"
##  [5] "Stroke1"       "Sex1"          "Arthritis1"     "Smoke4"
##  [9] "BMI"           "Education4"    "Age4"           "Fruit"
## [13] "Vegatable"
```

```
cat("Best Accuracy:", best_accuracy, "\n")
```

```
## Best Accuracy: 0.7022393
```

```
# Train the final model with the best features
train_best <- train_features[, best_features]
test_best <- test_features[, best_features]
xgb_model_best <- xgboost(
  params = params,
  data = train_best,
  label = train_label,
  nrounds = best_nrounds,
  print_every_n = 10
)
```

```
## [1]  train-auc:0.760642
## [10] train-auc:0.839854
## Compute Accuracy,  TPR,  FPR,  Test AUC
set.seed(123)
# Predict the test set
preds_best <- predict(xgb_model_best, test_best)

# Convert predictions to binary class labels
preds_class_best <- as.numeric(preds_best > best_threshold)

# Calculate TP, FP, TN, and FN
TP <- sum(test_label == 1 & preds_class_best == 1)
FP <- sum(test_label == 0 & preds_class_best == 1)
TN <- sum(test_label == 0 & preds_class_best == 0)
FN <- sum(test_label == 1 & preds_class_best == 0)

# Calculate TPR, FPR, and Accuracy
TPR <- TP / (TP + FN)
FPR <- FP / (FP + TN)
Accuracy <- (TP + TN) / (TP + FP + TN + FN)

cat("True Positive Rate (TPR):", TPR, "\n")
```

```
## True Positive Rate (TPR): 0.8090591
```

```
cat("False Positive Rate (FPR):", FPR, "\n")
```

```
## False Positive Rate (FPR): 0.303509
```

```
cat("Accuracy:", Accuracy, "\n")
```

```
## Accuracy: 0.7046012
```

```
# Calculate Test AUC
library(pROC)
test_auc <- roc(test_label, preds_best, direction = "<", auc = TRUE)
cat("Test AUC:", test_auc$auc, "\n")
```

```
## Test AUC: 0.8293006
```