Mathematical Methods for Artificial Intelligence Lab 4 - Boosting Alg.

Vytautas Kraujalis

2021-12-23

Table of Contents

[1 Gradient Boosting 1](#_Toc91257482)

[1.1 Required packages 1](#_Toc91257483)

[1.2 Reading Data 2](#_Toc91257484)

[1.3 Data Preparation 2](#_Toc91257485)

[1.4 Fitting Xgboost model 3](#_Toc91257486)

[1.4.1 Tuning learning rate 3](#_Toc91257487)

[1.4.2 Tuning max tree depth and minimum child weight 43](#_Toc91257488)

[1.4.3 Tuning row and column sampling 71](#_Toc91257489)

[1.4.4 Final tune for number of trees and learning rate with other optimal parameters 100](#_Toc91257490)

[1.4.5 Fitting model with final optimal parameters 173](#_Toc91257491)

[1.5 References 175](#_Toc91257492)

# 1 Gradient Boosting

## 1.1 Required packages

library(data.table)  
library(dplyr)  
library(tictoc)  
library(janitor)  
library(caret)  
library(xgboost)  
  
# Function for 4 class accuracy from confusion matrix  
classAcc <- function(confusionMatrix) {  
 class1 <- round(confusionMatrix$table[1, 1] / sum(confusionMatrix$table[, 1]) \* 100, 1)  
 class2 <- round(confusionMatrix$table[2, 2] / sum(confusionMatrix$table[, 2]) \* 100, 1)  
 class3 <- round(confusionMatrix$table[3, 3] / sum(confusionMatrix$table[, 3]) \* 100, 1)  
 class4 <- round(confusionMatrix$table[4, 4] / sum(confusionMatrix$table[, 4]) \* 100, 1)  
 acc <- c(class1, class2, class3, class4 )  
 names(acc) <- colnames(confusionMatrix$table)  
 return(acc)  
}  
  
# remove scientific notation  
options(scipen = 100)

## 1.2 Reading Data

set.seed(123)  
  
data\_original <- fread("activity.csv")  
data\_names <- read.table("names.txt") %>%   
 rename(column\_names = V1)

## 1.3 Data Preparation

data <- data\_original  
colnames(data) <- data\_names$column\_names  
  
data <- data %>%   
 clean\_names() %>%   
 mutate\_if(is.character, as.factor)  
data <- data %>%   
 select(-nearZeroVar(data))

# Correlation  
  
corr\_simple <- function(df,sig=0.5){  
 corr <- cor(df)  
 #prepare to drop duplicates and correlations of 1   
 corr[lower.tri(corr,diag=TRUE)] <- NA   
 #drop perfect correlations  
 corr[corr == 1] <- NA   
 #turn into a 3-column table  
 corr <- as.data.frame(as.table(corr))  
 #remove the NA values from above   
 corr <- na.omit(corr)   
 #select significant values   
 corr <- subset(corr, abs(Freq) > sig)   
 #sort by highest correlation  
 corr <- corr[order(-abs(corr$Freq)),]   
 return(corr)  
}  
  
correlation\_matrix = cor(data %>% select(-activity, -subject\_index))

data <- data %>%   
 select(-findCorrelation(correlation\_matrix, cutoff = 0.99)) %>%   
 mutate(subject\_index = as.factor(subject\_index))

## 1.4 Fitting Xgboost model

### 1.4.1 Tuning learning rate

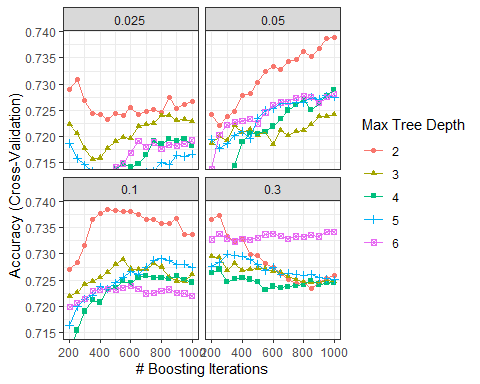
We’ll look for an optimal learning rate which we will use to find other optimal parameters.

tune\_grid <- expand.grid(  
 nrounds = seq(from = 200, to = 1000, by = 50),  
 eta = c(0.025, 0.05, 0.1, 0.3),  
 max\_depth = c(2, 3, 4, 5, 6),  
 gamma = 0,  
 colsample\_bytree = 1,  
 min\_child\_weight = 1,  
 subsample = 1  
)  
  
folds = groupKFold(data$subject\_index, k = 3)  
tune\_control <- trainControl(  
 index = folds,  
 method = "cv",  
 number = 3,  
 verboseIter = FALSE#, allowParallel = TRUE  
 )  
  
tic()  
xgb\_tune <- train(  
 activity ~ ., data = data %>% select(-subject\_index),  
 trControl = tune\_control,  
 tuneGrid = tune\_grid,  
 method = "xgbTree",  
 verbose = TRUE  
)

toc()

## 644.5 sec elapsed

# helper function for the plots  
tuneplot <- function(x, probs = .10) {  
 ggplot(x) +  
 coord\_cartesian(ylim = c(quantile(x$results$Accuracy, probs = probs), max(x$results$Accuracy))) +  
 theme\_bw()  
}  
  
tuneplot(xgb\_tune)



xgb\_tune$bestTune %>%   
 mutate\_all(round, digits = 5) %>%   
 t()

## 102  
## nrounds 1000.00  
## max\_depth 2.00  
## eta 0.05  
## gamma 0.00  
## colsample\_bytree 1.00  
## min\_child\_weight 1.00  
## subsample 1.00

Seems like the best accuracy is reached using 0.05 learning rate with 1000 number of trees and with max tree depth equal to 2. We will use this learning rate in our further tuning analysis.

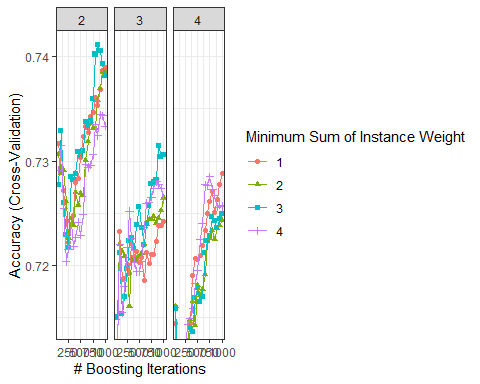
### 1.4.2 Tuning max tree depth and minimum child weight

tune\_grid2 <- expand.grid(  
 nrounds = seq(from = 50, to = 1000, by = 50),  
 eta = xgb\_tune$bestTune$eta,  
 max\_depth = if(xgb\_tune$bestTune$max\_depth == 2){  
 seq(xgb\_tune$bestTune$max\_depth, 4, 1)  
 }else{  
 seq(xgb\_tune$bestTune$max\_depth - 1, xgb\_tune$bestTune$max\_depth + 1, 1)  
 },  
 gamma = 0,  
 colsample\_bytree = 1,  
 min\_child\_weight = c(1, 2, 3, 4),  
 subsample = 1  
)  
  
tic()  
xgb\_tune2 <- train(  
 activity ~ ., data = data %>% select(-subject\_index),  
 trControl = tune\_control,  
 tuneGrid = tune\_grid2,  
 method = "xgbTree",  
 verbose = TRUE  
)

toc()

## 384.91 sec elapsed

tuneplot(xgb\_tune2)



xgb\_tune2$bestTune %>%   
 t()

## 57  
## nrounds 850.00  
## max\_depth 2.00  
## eta 0.05  
## gamma 0.00  
## colsample\_bytree 1.00  
## min\_child\_weight 3.00  
## subsample 1.00

Looks like a better accuracy can be obtained using 3 minimal child weight and with 2 max depth, same as before.

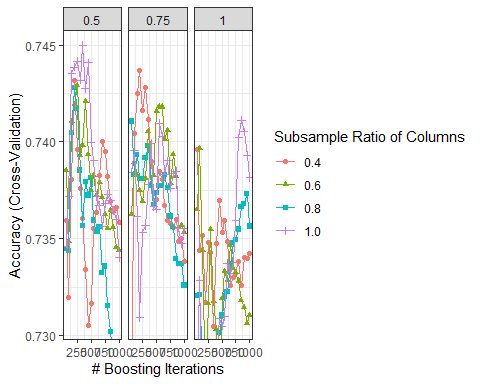
### 1.4.3 Tuning row and column sampling

tune\_grid3 <- expand.grid(  
 nrounds = seq(from = 50, to = 1000, by = 50),  
 eta = xgb\_tune$bestTune$eta,  
 max\_depth = xgb\_tune2$bestTune$max\_depth,  
 gamma = 0,  
 colsample\_bytree = c(0.4, 0.6, 0.8, 1.0),  
 min\_child\_weight = xgb\_tune2$bestTune$min\_child\_weight,  
 subsample = c(0.5, 0.75, 1.0)  
)  
  
tic()  
xgb\_tune3 <- train(  
 activity ~ ., data = data %>% select(-subject\_index),  
 trControl = tune\_control,  
 tuneGrid = tune\_grid3,  
 method = "xgbTree",  
 verbose = TRUE  
)

toc()

## 221.81 sec elapsed

tuneplot(xgb\_tune3)



xgb\_tune3$bestTune %>%   
 t()

## 187  
## nrounds 350.00  
## max\_depth 2.00  
## eta 0.05  
## gamma 0.00  
## colsample\_bytree 1.00  
## min\_child\_weight 3.00  
## subsample 0.50

Best accuracy is reached using 1 column sample and 0.5 row sample.

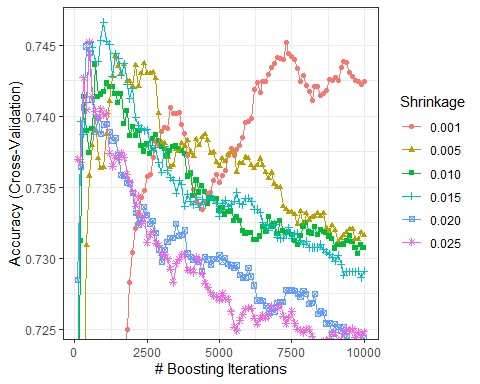
### 1.4.4 Final tune for number of trees and learning rate with other optimal parameters

tune\_grid4 <- expand.grid(  
 nrounds = seq(from = 100, to = 10000, by = 100),  
 eta = c(0.001, 0.005, 0.01, 0.015, 0.02, 0.025),  
 max\_depth = xgb\_tune2$bestTune$max\_depth,  
 gamma = 0,  
 colsample\_bytree = xgb\_tune3$bestTune$colsample\_bytree,  
 min\_child\_weight = xgb\_tune2$bestTune$min\_child\_weight,  
 subsample = xgb\_tune3$bestTune$subsample  
)  
  
tic()  
xgb\_tune4 <- train(  
 activity ~ ., data = data %>% select(-subject\_index),  
 trControl = tune\_control,  
 tuneGrid = tune\_grid4,  
 method = "xgbTree",  
 verbose = TRUE  
)

toc()

## 1155.64 sec elapsed

tuneplot(xgb\_tune4)



Final parameters:

xgb\_tune4$bestTune %>%   
 t()

## 310  
## nrounds 1000.000  
## max\_depth 2.000  
## eta 0.015  
## gamma 0.000  
## colsample\_bytree 1.000  
## min\_child\_weight 3.000  
## subsample 0.500

final\_grid <- expand.grid(  
 nrounds = xgb\_tune4$bestTune$nrounds,  
 eta = xgb\_tune4$bestTune$eta,  
 max\_depth = xgb\_tune4$bestTune$max\_depth,  
 gamma = 0,  
 colsample\_bytree = xgb\_tune4$bestTune$colsample\_bytree,  
 min\_child\_weight = xgb\_tune4$bestTune$min\_child\_weight,  
 subsample = xgb\_tune4$bestTune$subsample  
)

### 1.4.5 Fitting model with final optimal parameters

folds = groupKFold(data$subject\_index, k = 3)  
train\_control <- trainControl(  
 index = folds,  
 method = "cv",  
 number = 3,  
 verboseIter = FALSE,  
 classProbs = TRUE,  
 savePredictions='all'  
 )  
  
tic()  
xgb\_model <- train(  
 activity ~ ., data = data %>% select(-subject\_index),  
 trControl = train\_control,  
 tuneGrid = final\_grid,  
 method = "xgbTree",  
 verbose = TRUE  
)  
toc()

## 35.24 sec elapsed

xgb\_final <- xgb\_model$finalModel

data\_pred <- predict(xgb\_final, newdata = data %>% select(-subject\_index, -activity) %>% data.matrix())  
data\_prediction <- matrix(data\_pred, nrow = 4,  
 ncol = length(data\_pred)/4) %>%  
 t() %>%  
 data.frame() %>%  
 mutate(label = as.numeric(data$activity),  
 max\_prob = max.col(., "last"))  
  
confusion\_matrix <- confusionMatrix(factor(data\_prediction$max\_prob), factor(data\_prediction$label))  
confusion\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 1 2 3 4  
## 1 1074 102 18 0  
## 2 46 1013 23 0  
## 3 0 5 1079 0  
## 4 0 0 0 1120  
##   
## Overall Statistics  
##   
## Accuracy : 0.9567   
## 95% CI : (0.9503, 0.9625)   
## No Information Rate : 0.25   
## P-Value [Acc > NIR] : < 0.00000000000000022  
##   
## Kappa : 0.9423   
##   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: 1 Class: 2 Class: 3 Class: 4  
## Sensitivity 0.9589 0.9045 0.9634 1.00  
## Specificity 0.9643 0.9795 0.9985 1.00  
## Pos Pred Value 0.8995 0.9362 0.9954 1.00  
## Neg Pred Value 0.9860 0.9685 0.9879 1.00  
## Prevalence 0.2500 0.2500 0.2500 0.25  
## Detection Rate 0.2397 0.2261 0.2408 0.25  
## Detection Prevalence 0.2665 0.2415 0.2420 0.25  
## Balanced Accuracy 0.9616 0.9420 0.9810 1.00

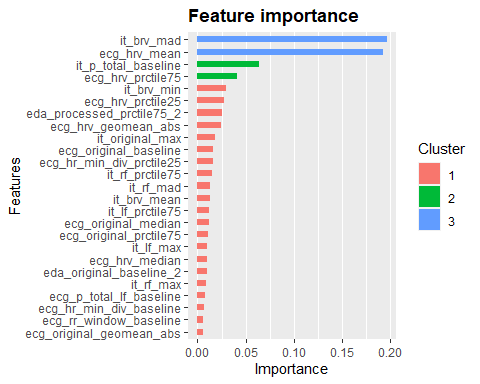
Final model has overall accuracy of 95.66%, while for each class accuracies are:

classAccuracies <- confusion\_matrix$byClass[,"Sensitivity"]\*100  
names(classAccuracies) <- levels(data$activity)  
classAccuracies

## emotional mental neural physical   
## 95.89286 90.44643 96.33929 100.00000

The model predicts physical class perfectly.

importance <- xgb.importance(model = xgb\_final)  
  
importance %>%   
 xgb.ggplot.importance(top\_n = 25, measure = NULL, rel\_to\_first = F)



According to feature importance and top 25 variables, there are 2 variables that stood up: one from ECG and one from TEB type of feature. Those 2 features are the most important according to the xgboost method.

## 1.5 References

<https://www.kaggle.com/pelkoja/visual-xgboost-tuning-with-caret> <https://www.r-bloggers.com/2019/10/explaining-predictions-boosted-trees-post-hoc-analysis-xgboost/>