Mathematical Methods for Artificial Intelligence Lab 4 - Random Forest and Boosting method

Vytautas Kraujalis

2021-12-18

Table of Contents

[1 Random Forest 2](#_Toc91258707)

[1.1 Required packages 2](#_Toc91258708)

[1.2 Parallel processing 2](#_Toc91258709)

[1.3 Reading Data 2](#_Toc91258710)

[1.4 Data Preparation 2](#_Toc91258711)

[1.5 EDA, first look at the dataset 3](#_Toc91258712)

[1.6 Fitting Random Forest Model 6](#_Toc91258713)

[1.6.1 Tune with OOB 6](#_Toc91258714)

[1.6.2 Random search 7](#_Toc91258715)

[1.6.3 Grid Search 9](#_Toc91258716)

[1.6.4 Tune number of trees 11](#_Toc91258717)

[1.6.5 Optimal Random Forest model 13](#_Toc91258718)

[2 Gradient Boosting 20](#_Toc91258719)

[2.1 Required packages 20](#_Toc91258720)

[2.2 Reading Data 20](#_Toc91258721)

[2.3 Data Preparation 20](#_Toc91258722)

[2.4 Fitting Xgboost model 21](#_Toc91258723)

[2.4.1 Tuning learning rate 21](#_Toc91258724)

[2.4.2 Tuning max tree depth and minimum child weight 23](#_Toc91258725)

[2.4.3 Tuning row and column sampling 24](#_Toc91258726)

[2.4.4 Final tune for number of trees and learning rate with other optimal parameters 26](#_Toc91258727)

[2.4.5 Fitting model with final optimal parameters 27](#_Toc91258728)

[3 Conclusion 30](#_Toc91258729)

[3 References 31](#_Toc91258730)

# 1 Random Forest

## 1.1 Required packages

library(data.table)  
library(SmartEDA)  
library(dplyr)  
library(ggplot2)  
library(caret)  
library(randomForest)  
library(janitor)  
library(tictoc)  
library(tidyr)  
library(tibble)  
library(ggrepel)  
  
# Function for 4 class accuracy from confusion matrix  
classAcc <- function(confusionMatrix) {  
 class1 <- round(confusionMatrix[1, 1] / sum(confusionMatrix[, 1]) \* 100, 1)  
 class2 <- round(confusionMatrix[2, 2] / sum(confusionMatrix[, 2]) \* 100, 1)  
 class3 <- round(confusionMatrix[3, 3] / sum(confusionMatrix[, 3]) \* 100, 1)  
 class4 <- round(confusionMatrix[4, 4] / sum(confusionMatrix[, 4]) \* 100, 1)  
 acc <- c(class1, class2, class3, class4 )  
 names(acc) <- colnames(confusionMatrix)  
 return(acc)  
}

## 1.2 Parallel processing

library(parallel)   
no\_cores <- detectCores() - 1  
library(doParallel)  
cl <- makePSOCKcluster(no\_cores)  
registerDoParallel(cl)

## 1.3 Reading Data

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

## 1.4 Data Preparation

length(data\_names$column\_names)

## [1] 535

n\_distinct(data\_names$column\_names)

## [1] 417

There are 535 provided column names, but only 417 are distinct, it means we have some duplicated names, we need to make them unique. To do that, for duplicated names we’ll add a unique ID at the end:

data <- data\_original  
colnames(data) <- data\_names$column\_names  
  
data <- data %>%   
 clean\_names()  
  
n\_distinct(colnames(data))

## [1] 535

## 1.5 EDA, first look at the dataset

ExpData(data,type = 1)

## Descriptions Value  
## 1 Sample size (nrow) 4480  
## 2 No. of variables (ncol) 535  
## 3 No. of numeric/interger variables 534  
## 4 No. of factor variables 0  
## 5 No. of text variables 1  
## 6 No. of logical variables 0  
## 7 No. of identifier variables 0  
## 8 No. of date variables 0  
## 9 No. of zero variance variables (uniform) 4  
## 10 %. of variables having complete cases 100% (535)  
## 11 %. of variables having >0% and <50% missing cases 0% (0)  
## 12 %. of variables having >=50% and <90% missing cases 0% (0)  
## 13 %. of variables having >=90% missing cases 0% (0)

We have a dataset of 4480 observations with 535 variables, only 1 variable has text format. All variables have no missing values. We can see that there are 4 variables with zero variance, we’ll remove those later.

Let’s look at the response variable:

data %>%   
 group\_by(activity) %>%   
 summarise(n = n()) %>%   
 mutate(n\_prop = round(n / sum(n) \* 100, 2))

## # A tibble: 4 x 3  
## activity n n\_prop  
## <chr> <int> <dbl>  
## 1 emotional 1120 25  
## 2 mental 1120 25  
## 3 neural 1120 25  
## 4 physical 1120 25

We have perfectly balanced response variable with 4 classes.

We’ll change the response variable to factor type.

data <- data %>%   
 mutate\_if(is.character, as.factor)

data %>%   
 select(nearZeroVar(data)) %>%   
 summary()

## ecg\_p\_vfl\_kurtosis ecg\_p\_lf\_kurtosis it\_vlf\_kurtosis it\_lf\_kurtosis\_2  
## Min. :1.5 Min. :1 Min. :1.5 Min. :1   
## 1st Qu.:1.5 1st Qu.:1 1st Qu.:1.5 1st Qu.:1   
## Median :1.5 Median :1 Median :1.5 Median :1   
## Mean :1.5 Mean :1 Mean :1.5 Mean :1   
## 3rd Qu.:1.5 3rd Qu.:1 3rd Qu.:1.5 3rd Qu.:1   
## Max. :1.5 Max. :1 Max. :1.5 Max. :1

As mentioned previously, we have 4 variables with zero variance, we will remove those columns.

data <- data %>%   
 select(-nearZeroVar(data))

We should look at the correlation between variables

# 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))  
  
length(findCorrelation(correlation\_matrix, cutoff = 0.99))

## [1] 309

length(findCorrelation(correlation\_matrix, cutoff = 0.95))

## [1] 369

length(findCorrelation(correlation\_matrix, cutoff = 0.9))

## [1] 389

We have 309 variables with correlation greater than 0.99, we will remove those variables.

data <- data %>%   
 select(-findCorrelation(correlation\_matrix, cutoff = 0.99))

We’ll convert subject index column to factor type:

data %>%   
 select(subject\_index) %>%   
 table()

## .  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20   
## 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112   
## 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40   
## 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112 112

data <- data %>%   
 mutate(subject\_index = as.factor(subject\_index))

Let’s look how obersvations are distributed across subjects and activity types:

data %>%   
 group\_by(subject\_index, activity) %>%   
 summarise(n = n()) %>%   
 pivot\_wider(names\_from = activity, values\_from = n)

## # A tibble: 40 x 5  
## # Groups: subject\_index [40]  
## subject\_index emotional mental neural physical  
## <fct> <int> <int> <int> <int>  
## 1 1 28 28 28 28  
## 2 2 28 28 28 28  
## 3 3 28 28 28 28  
## 4 4 28 28 28 28  
## 5 5 28 28 28 28  
## 6 6 28 28 28 28  
## 7 7 28 28 28 28  
## 8 8 28 28 28 28  
## 9 9 28 28 28 28  
## 10 10 28 28 28 28  
## # ... with 30 more rows

Each person has 28 observations of each activity. We are not going to use Out-of-Bag score for tuning parameters as this would be misleading. Same person with the same activity could be splited into different sets and the result based on OOB could be misleading. We are not going to have this problem with Cross Validation, as we can specify folds to be grouped according to subjects.

## 1.6 Fitting Random Forest Model

### 1.6.1 Tune with OOB

Let’s look at the optimal mtry value based on OOB and number of trees = 500

tic()  
png(file = "Tune\_oob.png", width = 1200, height = 850)  
rf\_tune <- tuneRF(data %>%   
 select(-subject\_index, -activity),  
 data$activity,   
 mtryStart = 2,   
 ntreeTry = 500,  
 stepFactor = 3,  
 improve = 0.001,  
 trace = TRUE,  
 plot = TRUE)

## mtry = 2 OOB error = 1.27%   
## Searching left ...  
## mtry = 1 OOB error = 2.48%   
## -0.9473684 0.001   
## Searching right ...  
## mtry = 6 OOB error = 1.38%   
## -0.0877193 0.001

dev.off()

## png   
## 2

toc()

## 53.71 sec elapsed

Chart, line chart

Description automatically generated

Based on OOB the optimal mtry was found to be 2. We are not going to use this value as mentioned previously.

### 1.6.2 Random search

folds = groupKFold(data$subject\_index, k = 10)  
fitControl <- trainControl(## 10-fold CV  
 index = folds,  
 method = "cv",  
 number = 10,  
 classProbs = TRUE,  
 savePredictions='all',  
 verboseIter = TRUE,  
 allowParallel = TRUE,  
 search = "random")  
  
tic()  
rf\_random <- train(  
 activity ~ .,   
 data = data %>% select(-subject\_index),   
 method = "rf",   
 metric = "Accuracy",   
 tuneLength = 10,   
 trControl = fitControl)

## Aggregating results  
## Selecting tuning parameters  
## Fitting mtry = 2 on full training set

toc()

## 344.86 sec elapsed

print(rf\_random)

## Random Forest   
##   
## 4480 samples  
## 220 predictor  
## 4 classes: 'emotional', 'mental', 'neural', 'physical'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 4256, 4256, 4032, 3920, 3584, 4368, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.7234375 0.6312500  
## 52 0.7196875 0.6262500  
## 61 0.7194866 0.6259821  
## 67 0.7191518 0.6255357  
## 75 0.7177009 0.6236012  
## 82 0.7203348 0.6271131  
## 101 0.7155357 0.6207143  
## 118 0.7163616 0.6218155  
## 164 0.7118304 0.6157738  
## 188 0.7123437 0.6164583  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 2.

png(file = "Tune\_randomSearch.png", width = 1200, height = 850)  
plot(rf\_random)  
dev.off()

## png   
## 2

Chart, line chart

Description automatically generated

Based on random search, the optimal mtry was found to be 2. We are going to also use a grid search, to look if an optimal mtry could be closer to sqrt(m) value.

### 1.6.3 Grid Search

folds = groupKFold(data$subject\_index, k = 10)  
fitControl <- trainControl(## 10-fold CV  
 index = folds,  
 method = "cv",  
 number = 10,  
 classProbs = TRUE,  
 savePredictions='all',  
 verboseIter = TRUE,  
 allowParallel = TRUE,  
 search = "grid")  
  
rf\_grid = expand.grid(.mtry = c(seq(2,15,2), seq(rf\_random$bestTune$mtry - 4, rf\_random$bestTune$mtry + 4, 2)))  
  
tic()  
rf\_fit\_grid\_cv <- train(  
 activity ~ .,   
 data = data %>% select(-subject\_index),   
 method = "rf",   
 metric = "Accuracy",  
 tuneGrid = rf\_grid,  
 trControl = fitControl)

## Aggregating results  
## Selecting tuning parameters  
## Fitting mtry = 10 on full training set

toc()

## 188.34 sec elapsed

print(rf\_fit\_grid\_cv)

## Random Forest   
##   
## 4480 samples  
## 220 predictor  
## 4 classes: 'emotional', 'mental', 'neural', 'physical'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 4144, 3584, 4032, 4144, 4032, 3584, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## -2 0.7037285 0.6049713  
## 0 0.7038938 0.6051918  
## 2 0.7071181 0.6094907  
## 4 0.7101438 0.6135251  
## 6 0.7241237 0.6321649  
## 8 0.7255622 0.6340829  
## 10 0.7281415 0.6375220  
## 12 0.7229332 0.6305776  
## 14 0.7230985 0.6307981  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 10.

png(file = "Tune\_gridSearch.png", width = 1200, height = 850)  
plot(rf\_fit\_grid\_cv)  
dev.off()

## png   
## 2

Chart, line chart

Description automatically generated

Using grid search we found an optimal value of mtry to be 10. We are going to use this value in our further analysis.

### 1.6.4 Tune number of trees

folds = groupKFold(data$subject\_index, k = 10)  
fitControl <- trainControl(## 10-fold CV  
 index = folds,  
 method = "cv",  
 number = 10,  
 classProbs = TRUE,  
 savePredictions='all',  
 verboseIter = TRUE,  
 allowParallel = TRUE,  
 search = "grid")  
  
rf\_grid = expand.grid(.mtry = rf\_fit\_grid\_cv$bestTune$mtry)  
  
modellist <- list()  
for (ntree in c(350, 500, 750, 1000, 1500)) {  
 fit <- train(  
 activity ~ .,   
 data = data %>% select(-subject\_index),   
 method = "rf",   
 metric = "Accuracy",  
 tuneGrid = rf\_grid,  
 trControl = fitControl,  
 ntree = ntree)  
 key <- toString(ntree)  
 modellist[[key]] <- fit  
}

## Aggregating results  
## Fitting final model on full training set  
## Aggregating results  
## Fitting final model on full training set  
## Aggregating results  
## Fitting final model on full training set  
## Aggregating results  
## Fitting final model on full training set  
## Aggregating results  
## Fitting final model on full training set

# compare results  
results <- resamples(modellist)  
summary(results)

##   
## Call:  
## summary.resamples(object = results)  
##   
## Models: 350, 500, 750, 1000, 1500   
## Number of resamples: 10   
##   
## Accuracy   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## 350 0.5848214 0.6928013 0.7444196 0.7280432 0.7814732 0.8616071 0  
## 500 0.6160714 0.6994978 0.7493304 0.7355134 0.7836310 0.8616071 0  
## 750 0.5848214 0.6964286 0.7491071 0.7307440 0.7853423 0.8616071 0  
## 1000 0.5982143 0.6947545 0.7419643 0.7286756 0.7779018 0.8392857 0  
## 1500 0.5892857 0.6944754 0.7431548 0.7275074 0.7681920 0.8571429 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## 350 0.4464286 0.5904018 0.6592262 0.6373909 0.7086310 0.8154762 0  
## 500 0.4880952 0.5993304 0.6657738 0.6473512 0.7115079 0.8154762 0  
## 750 0.4464286 0.5952381 0.6654762 0.6409921 0.7137897 0.8154762 0  
## 1000 0.4642857 0.5930060 0.6559524 0.6382341 0.7038690 0.7857143 0  
## 1500 0.4523810 0.5926339 0.6575397 0.6366766 0.6909226 0.8095238 0

png(file = "Tune\_ntree.png", width = 1200, height = 850)  
dotplot(results)  
dev.off()

## png   
## 2

mtry\_optimal <- rf\_fit\_grid\_cv$bestTune$mtry  
  
ntree\_optimal <- as.integer(names(which(summary(results)$statistics$Accuracy[,"Mean"] == max(summary(results)$statistics$Accuracy[,"Mean"]))))

Chart

Description automatically generated

Optimal number of trees were found to be 500.

### 1.6.5 Optimal Random Forest model

Fit a random forest model with best parameters (CV)

folds = groupKFold(data$subject\_index, k = 10)  
fitControl <- trainControl(## 10-fold CV  
 index = folds,  
 method = "cv",  
 number = 10,  
 classProbs = TRUE,  
 savePredictions='all',  
 verboseIter = TRUE,  
 allowParallel = TRUE)  
  
tic()  
rf\_fit\_cv <- train(  
 activity ~ .,   
 data = data %>% select(-subject\_index),   
 method = "rf",   
 tuneGrid = expand.grid(.mtry = mtry\_optimal),  
 ntree = ntree\_optimal,  
 trControl = fitControl,  
 importance = TRUE)

## Aggregating results  
## Fitting final model on full training set

toc()

## 115.7 sec elapsed

rf\_final <- rf\_fit\_cv$finalModel

print(rf\_final)

##   
## Call:  
## randomForest(x = x, y = y, ntree = ..1, mtry = min(param$mtry, ncol(x)), importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 10  
##   
## OOB estimate of error rate: 1.65%  
## Confusion matrix:  
## emotional mental neural physical class.error  
## emotional 1108 12 0 0 0.010714286  
## mental 19 1101 0 0 0.016964286  
## neural 17 22 1081 0 0.034821429  
## physical 1 3 0 1116 0.003571429

confusion\_matrix <- rf\_final$confusion[,1:4]  
classAcc(confusion\_matrix)

## emotional mental neural physical   
## 96.8 96.7 100.0 100.0

Final model has accuracy of 98.35% with quite good accuracy for each class.

rf\_margin <- margin(rf\_final, data$activity)  
df = data.frame(margin = as.numeric(rf\_margin),   
 label = names(rf\_margin),   
 index = 1:length(rf\_margin))  
df %>%   
 ggplot(aes(x = index,y = margin,col = label)) +   
 geom\_point() +  
 theme(text = element\_text(size = 14))

Chart, scatter chart

Description automatically generated

ggsave(filename = "Margin\_ScatterPlot.png", width = 14, height = 7, units = "in", bg = "white")  
  
df %>%   
 ggplot(aes(x = margin, fill = label)) +   
 geom\_histogram() +   
 facet\_wrap(~label) +  
 theme(text = element\_text(size = 14))

Chart

Description automatically generated

ggsave(filename = "Margin\_BarPlot.png", width = 14, height = 7, units = "in", bg = "white")

We clearly see, that only a small number of observations were misclassified (margin < 0). We also clearly see that majority of Neutral and Physical activity observations were classified correctly with almost perfect voting score for the correct class. One could also identify, that for classes Emotional and Mental, most of the correct guesses were made with ~0.5 - ~0.75 majority votes. To increase the model effectiveness, one should find more features to distinguish Emotional and Mental classes.

Let’s explore model performance for each subject:

# get a table of actual and predicted values for each subject  
df <- data.frame(  
 subject\_id = data$subject\_index,  
 activity = rf\_final$y,  
 activity\_prediction = rf\_final$predicted  
)  
  
# list of tables by each subject  
df\_subject <- split(df, df$subject\_id)  
  
# list of confusion matrices by each subject  
df\_subject <- lapply(df\_subject, function(x){confusionMatrix(x$activity\_prediction, x$activity)$table})  
  
# list of class and overall accuracies by each subject  
df\_subject <-lapply(df\_subject, function(x){  
 AccClasses = classAcc(x)  
 AccOverall = round(sum(diag(x)) / sum(colSums(x)) \* 100, 2)  
 acc = c(AccClasses, AccOverall)  
 names(acc) = c(names(AccClasses), "OVERALL")  
 return(acc)  
})  
  
df\_colnames <- names(df\_subject[[1]])  
  
# a table of class and overall accuracies by each subject  
df\_subject <- do.call(rbind.data.frame, df\_subject) %>%   
 rownames\_to\_column("subject\_id") %>%   
 mutate(subject\_id = as.numeric(subject\_id))  
colnames(df\_subject) <- c("subject\_id", df\_colnames)  
  
# Plot of overall  
df\_subject %>%  
 arrange(OVERALL) %>%  
 mutate(subject\_id = factor(subject\_id, levels = subject\_id)) %>%   
 ggplot( aes(x = subject\_id, y = OVERALL)) +  
 geom\_point( size = 4, color = "orange") +  
 ylab("Overall Accuracy (%)") +  
 coord\_flip() +  
 theme\_bw() +  
 lims(y = c(min(df\_subject$OVERALL) - 0.5, 100)) +  
 geom\_label\_repel(aes(label = paste0("id: ", subject\_id)),  
 box.padding = 0.35,   
 point.padding = 0.5,  
 segment.color = 'grey50') +  
 theme(axis.text.y=element\_blank(),  
 axis.title.y=element\_blank(),  
 text = element\_text(size = 14))

Chart

Description automatically generated

We can see that we don’t have any exceptional subject. All of them are closely to each other.

ggsave(filename = "OverallAccBySubject.png", width = 14, height = 7, units = "in", bg = "white")  
  
df\_subject %>%   
 pivot\_longer(cols = c(-subject\_id), names\_to = "Type", values\_to = "Accuracy") %>%   
 group\_by(Type, Accuracy\_Rounded = round(Accuracy)) %>%   
 summarise(n = n()) %>%   
 pivot\_wider(names\_from = Type, values\_from = n, values\_fill = 0) %>%   
 arrange(desc(Accuracy\_Rounded))

## # A tibble: 9 x 6  
## Accuracy\_Rounded emotional mental neural OVERALL physical  
## <dbl> <int> <int> <int> <int> <int>  
## 1 100 31 28 19 11 36  
## 2 99 0 0 0 9 0  
## 3 98 0 0 0 5 0  
## 4 97 0 0 0 8 0  
## 5 96 7 7 11 7 4  
## 6 93 1 3 6 0 0  
## 7 89 1 2 1 0 0  
## 8 86 0 0 2 0 0  
## 9 82 0 0 1 0 0

df <- importance(rf\_final) %>%   
 as.data.frame() %>%   
 rownames\_to\_column("feature") %>%   
 mutate(  
 feature\_type = case\_when(  
 substr(feature, 0, 2) == "ec" ~ "ECG",  
 substr(feature, 0, 2) == "it" ~ "TEB",  
 substr(feature, 0, 2) == "ed" ~ "EDA",  
 TRUE ~ "ERROR"  
 )  
 ) %>%   
 mutate(feature = paste0(feature\_type, " - ", feature))  
  
df %>%   
 arrange(MeanDecreaseAccuracy) %>%  
 tail(25) %>%   
 mutate(feature = factor(feature, levels = feature)) %>%   
 ggplot(aes(MeanDecreaseAccuracy, feature)) +  
 geom\_point() +  
 scale\_x\_continuous(limits=c(0,NA), expand=expansion(c(0,0.04))) +  
 theme\_bw() +  
 theme(panel.grid.minor=element\_blank(),  
 panel.grid.major.x=element\_blank(),  
 panel.grid.major.y=element\_line(),  
 axis.title=element\_blank(),  
 text = element\_text(size = 14)) +  
 labs(title = "Mean decrease in accuracy")

A screenshot of a computer

Description automatically generated with medium confidence

At first glance, looking at the top 25 features by mean decrease in accuracy we cannot see any few exceptional features.

ggsave(filename = "VariableImportance.png", width = 14, height = 7, units = "in", bg = "white")  
  
temp <- df %>%   
 arrange(MeanDecreaseAccuracy) %>%  
 group\_by(feature\_type) %>%   
 summarise(n\_all = n())  
  
df %>%   
 arrange(MeanDecreaseAccuracy) %>%  
 tail(25) %>%   
 group\_by(feature\_type) %>%   
 summarise(n = n()) %>%   
 left\_join(temp, by = "feature\_type") %>%   
 mutate(n\_prop = n / n\_all \* 100)

## # A tibble: 3 x 4  
## feature\_type n n\_all n\_prop  
## <chr> <int> <int> <dbl>  
## 1 ECG 5 91 5.49  
## 2 EDA 16 68 23.5   
## 3 TEB 4 61 6.56

While TEB and ECG features are top 3, based on top 25 variables, the most importance is coming from EDA type of feature. From top 25 variables it takes up to 16 variables which is 23.5% out of total EDA variables in the dataset (68).

# 2 Gradient Boosting

## 2.1 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)

## 2.2 Reading Data

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

## 2.3 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))

## 2.4 Fitting Xgboost model

### 2.4.1 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)

Chart, scatter chart

Description automatically generated

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.

### 2.4.2 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)

Chart, scatter chart

Description automatically generated

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.

### 2.4.3 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)

Chart, bar chart, histogram

Description automatically generated

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.

### 2.4.4 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)

Chart, scatter chart

Description automatically generated

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  
)

### 2.4.5 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)

Chart, bar chart

Description automatically generated

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.

# 3 Conclusion

We see some interesting results, random forest with 500 number of trees and 10 variables per split (mtry) gave us overall accuracy of 98.35% with neural and physical class being classified perfectly (100%) while emotional and mental classes had 96.8% and 96.7% accuracies.

Using xgboost, overall accuracy fell down to 95.66% and only physical class being classified perfectly, while emotional, mental and neural classes have 95.89%, 90.45% and 96.34% accuracies. Seems like model struggled with mental class much more than a random forest.

One key difference between the 2 models is that xgboost method identifies 2 features (it\_brv\_mad and ecg\_hrv\_mean) that are the most important and those 2 variables could be “clustered” into special cluster. Random forest model did not find any exceptional features and those 2 features mentioned, are not even present in the top 25 according to the mean decrease in accuracy. Both methods consider “it\_p\_total\_baseline” feauture as very important, it take 3rd place in xgboost method and 1st place in the random forest model.

# 3 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/>