Sex Classification Using Univariate Features

Sex Classifiacation

```
library(tidyverse)
```

Import Libraries

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(dplyr)
library(tibble)
library(cli)
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
library(caTools)
library(ROCR)
library(doParallel)
```

```
## 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
```

```
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
```

```
## Loaded glmnet 4.1-7
```

```
seed <- sample.int(99999, 1)
set.seed(seed)
```

Generating Seeds

```
dir.data <- "./Data"
dir.processing <- "./Data_Processing"
dir.analysis <- "./Analysis"
dir.output <- paste0(dir.analysis, "/Sex/Output")
```

Directory Names

Load Avarage Features This datasets contain avarage of fuctional connectivity, acf, pacf, standard deviation and mean of fmri time series across 4 scans.

```
load(paste0(dir.processing, "/Output/mean_all.RData"))
load(paste0(dir.processing, "/Output/sd_all.RData"))
load(paste0(dir.processing, "/Output/subject_id_all.RData"))
load(paste0(dir.processing, "/Output/pacf1_all.RData"))
load(paste0(dir.processing, "/Output/pacf2_all.RData"))
load(paste0(dir.processing, "/Output/pacf3_all.RData"))
load(paste0(dir.processing, "/Output/acf2_all.RData"))
load(paste0(dir.processing, "/Output/acf3_all.RData"))
load(paste0(dir.processing, "/Output/acf4_all.RData"))
load(paste0(dir.processing, "/Output/yycor_all.RData"))
```

Load Subject Id Each sample of our dataset has a subject ID. We load different subsets od those for different scenarios. 1. If we want consider all samples :

```
load(paste0(dir.processing, "/Output/subject_id_all.RData"))
subject_id <- as.numeric(subject_id_all)
```

2. In case of twins if we want to include only the first twin

```
load(paste0(dir.processing, "/Output/first_twin_subject_id.RData"))
subject_id <- as.numeric(subject_id_all)
```

2. If we want to include only the second twin

```
load(paste0(dir.processing, "/Output/second_twin_subject_id.RData"))
subject_id <- as.numeric(subject_id_all)
```

```
feature <- c()
feature <- cbind(
  feature, timeseries_mean_arr[1, , ],
  timeseries_sd_arr[1, , ],
  timeseries_pacf1_arr[1, , ], timeseries_pacf2_arr[1, , ],
  timeseries_pacf3_arr[1, , ],
  timeseries_acf2_arr[1, , ], timeseries_acf3_arr[1, , ], timeseries_acf4_arr[1, , ]
)
```

Create feature matrix Containing Mean, Sd, ACF, PACF

```
unrestricted_elvisha <- read.csv(paste0(
  dir.data,
  "/Elvisha/dem/unrestricted_elvisha9_3_9_2021_10_37_53.csv"
))

restricted_elvisha <- read.csv(paste0(
  dir.data,
  "/Elvisha/dem/RESTRICTED_elvisha9_3_9_2021_10_38_4.csv"
))
```

Load Dependent Variable

```
elvisa_gender <- unrestricted_elvisha %>%
  filter(Subject %in% subject_id) %>%
  select(Subject, Gender)

## M = 1 , F = 0

elvisa_gender$Gender[elvisa_gender$Gender == "M"] <- 1
elvisa_gender$Gender[elvisa_gender$Gender == "F"] <- 0

label <- as.numeric(elvisa_gender$Gender)
```

Create Label

Transform Different Properties Dataset Here we define Parent_ID as a unique identifier composed of Father_ID and Mother_ID.

```
transformed_elvisha <- restricted_elvisha %>%
  mutate(twin_status = (ZygositySR != "NotTwin")) %>%
  filter(Subject %in% subject_id_all) %>%
  select(Subject, twin_status, Mother_ID, Father_ID) %>%
  mutate(Parent_ID = Mother_ID * (10^5) + Father_ID) %>%
  select(Subject, Parent_ID)
```

```
feature <- as.matrix(feature)
mydata <- cbind(subject_id, label, feature)
```

Join ID, Label and Featues

Filter feature Dataset Here We filter the feature matrix and transformed elvisha dataframe to only contain samples from filtered Ids.

```
transformed_elvisha <- transformed_elvisha %>%
  mutate(index = 1:dim(transformed_elvisha)[1])
minus_index <- transformed_elvisha %>%
  filter(!(Subject %in% second_twin_subject_id))
minus_index <- minus_index$index
transformed_elvisha <- transformed_elvisha[-minus_index, ]
mydata <- mydata[-minus_index, ]
```

Split fuctions For Different Scinarios

1. Split Without any consideration

```
split_index <- function(splitratio) {
  return(sample.split(mydata[, 2], SplitRatio = splitratio))
}
```

2. Parent Id Split :

Here we split dataset such that samples having same Parent is either all in train set or in test set. This ay no two samples in train and test set have same parent and train and test set are completely seperated.

```
split_index <- function(splitratio) {
  parent_split <- sample.split(unique(transformed_elvisha$Parent_ID), SplitRatio = splitratio)
  transformed_elvisha <- transformed_elvisha %>%
    mutate(split = Parent_ID %in% unique(transformed_elvisha$Parent_ID)[parent_split == TRUE])
  return(transformed_elvisha$split)
}
```

```
split <- split_index(splitratio = 0.8)
train_data <- subset(mydata, split == "TRUE")
test_data <- subset(mydata, split == "FALSE")

print(paste0(
  "Train data percentage= ",
  dim(train_data)[1] * 100 / (dim(train_data)[1] + dim(test_data)[1])
))
```

Split Dataset between Train and Test

```
## [1] "Train data percentage= 79.874213836478"
```

```
feature_train <- train_data[, 3:dim(train_data)[2]]
label_train <- train_data[, 2]
feature_test <- test_data[, 3:dim(train_data)[2]]
label_test <- test_data[, 2]
```

Mixing Parameter The objective function will look like:

$$f = -\frac{\log L}{n} + \lambda * penalty$$

where,

$$penalty = \frac{(1-\alpha)}{2}\|\beta\|_2^2 + \alpha\|\beta\|_1$$

Here, α is called the mix parameter and $0 \leq \alpha \leq 1$

```
mix <- 0.5
```

Create CV GLMnet Object We have used the R function called "cv.glmnet" to find the optimum value of the smoothing parameter λ

```
c1 <- makeCluster(7)
registerDoParallel(c1)
cvglmnet <- cv.glmnet(feature_train,
  label_train,
  family = binomial(link = "logit"),
```

```
  gamma = mix,
  parallel = TRUE
)
stopCluster(c1)
```

Create GLMnet from optimal Parameters Using this optimal value we have used the "glmnet" function to get the coefficients β and the predicted classes for each individual.

```
myglmnet <- glmnet(feature_train,
  label_train,
  family = binomial(link = "logit"),
  alpha = mix,
  lambda = cvglmnet$lambda.1se
)
```

```
probs_train <- predict(myglmnet, feature_train, type = "response")
ROCPred_train <- prediction(probs_train, label_train)
ROCPer_train <- performance(ROCPred_train,
  measure = "tpr",
  x.measure = "fpr"
)

auc_train <- performance(ROCPred_train, measure = "auc")
auc_train <- auc_train@y.values[[1]]
auc_train
```

Calculate Train AUC

```
## [1] 0.9905139
```

```
probs_test <- predict(myglmnet, feature_test, type = "response")
ROCPred_test <- prediction(probs_test, label_test)
ROCPer_test <- performance(ROCPred_test,
  measure = "tpr",
  x.measure = "fpr"
)

auc_test <- performance(ROCPred_test, measure = "auc")
auc_test <- auc_test@y.values[[1]]
auc_test
```

Calculate Test AUC

```
## [1] 0.9311373
```

```
predict_train <- ifelse(probs_train > 0.5, 1, 0)
missing_classerr_train <- mean(predict_train != label_train)
accuracy_train <- 1 - missing_classerr_train
accuracy_train
```

Calculate Train Accuracy

```
## [1] 0.9527559
```

```
predict_train <- ifelse(probs_train > 0.5, 1, 0)
missing_classerr_train <- mean(predict_train != label_train)
accuracy_train <- 1 - missing_classerr_train
accuracy_train
```

Calculate Test Accuracy

```
## [1] 0.9527559
```

Cognition Prediction

Data preperation is very similar to sex prediction.

```
elvisa_cognition <- unrestricted_elvisha %>%
  filter(Subject %in% subject_id) %>% # only subjects in feature
  select(Subject, CogTotalComp_AgeAdj, Age, Gender)

## M = 1 , F = 0

elvisa_cognition$Gender[elvisa_cognition$Gender == "M"] <- 1
elvisa_cognition$Gender[elvisa_cognition$Gender == "F"] <- 0
elvisa_cognition$Gender <- as.numeric(elvisa_cognition$Gender)

elvisa_cognition$Age <- as.factor(elvisa_cognition$Age)
elvisa_cognition$Age <- as.numeric(elvisa_cognition$Age)

elvisa_cognition <- elvisa_cognition %>% mutate(
  combined = (10 * Age) + Gender
)
elvisa_cognition$combined <- as.factor(elvisa_cognition$combined)
elvisa_cognition$combined <- as.numeric(elvisa_cognition$combined)
```

Create Cognitivs Score Variable, Age and Gender column

```
nonnanindex <- !is.na(elvisa_cognition[, 2])
```

removing subjects that do not have cognition scores

Transform Different Properties Dataset Here we define Parent_ID as a unique identifier composed of Father_ID and Mother_ID.

```
transformed_elvisha <- restricted_elvisha %>%
  mutate(twin_status = (ZygositySR != "NotTwin")) %>%
  filter(Subject %in% subject_id_all) %>%
  select(Subject, twin_status, Mother_ID, Father_ID) %>%
  mutate(Parent_ID = Mother_ID * (10^5) + Father_ID) %>%
  select(Subject, Parent_ID) %>%
  slice(which(nonnanindex))
```

```
label <- as.numeric(elvisa_cognition[, 2])
new_subject_id <- as.numeric(elvisa_cognition$Subject)
```

```
Combined <- as.numeric(elvisa_cognition$Combined)
Combined <- Combined[nonnanindex]

Gender <- as.numeric(elvisa_cognition$Gender)
Gender <- Gender[nonnanindex]

Age <- as.numeric(elvisa_cognition$Age)
Age <- Gender[nonnanindex]

label <- label[nonnanindex]
new_subject_id <- new_subject_id[nonnanindex]
feature <- feature[nonnanindex, ]
```

Create label, age, gender, and combined vecotor

```
feature <- as.matrix(feature)
mydata <- cbind(new_subject_id, label, feature)
```

Join ID, Label and Featues

Filter feature Dataset Here We filter the feature matrix and transformed elvisha dataframe to only contain samples from filtered Ids.

```
transformed_elvisha <- transformed_elvisha %>%
  mutate(index = 1:dim(transformed_elvisha)[1])
minus_index <- transformed_elvisha %>%
  filter(!(Subject %in% second_twin_subject_id))
minus_index <- minus_index$index
transformed_elvisha <- transformed_elvisha[-minus_index, ]
mydata <- mydata[-minus_index, ]
```

Split fuctions For Different Scinarios

1. Split Without any consideration

```
split_index <- function(splitratio) {
  return(sample.split(mydata[, 2], SplitRatio = splitratio))
}
```

2. Split By Age :

```
split_index <- function(splitratio) {
  return(sample.split(Age, SplitRatio = splitratio))
}
```

3. Split by Gender

```
split_index <- function(splitratio) {
  return(sample.split(Gender, SplitRatio = splitratio))
}
```

4. Split by Age and Gneder Combined

```
split_index <- function(splitratio) {
  return(sample.split(Combined, SplitRatio = splitratio))
}
```

2. Parent Id Split :

Here we split dataset such that samples having same Parent is either all in train set or in test set. This ay no two samples in train and test set have same parent and train and test set are completely seperated.

```
split_index <- function(splitratio) {
  parent_split <- sample.split(unique(transformed_elvisha$Parent_ID), SplitRatio = splitratio)
  transformed_elvisha <- transformed_elvisha %>%
    mutate(split = Parent_ID %in% unique(transformed_elvisha$Parent_ID)[parent_split == TRUE])
  return(transformed_elvisha$split)
}
```

```
split <- split_index(splitratio = 0.8)
train_data <- subset(mydata, split == "TRUE")
test_data <- subset(mydata, split == "FALSE")

print(paste0(
  "Train data percentage= ",
  dim(train_data)[1] * 100 / (dim(train_data)[1] + dim(test_data)[1])
))
```

Split Dataset between Train and Test

```
## [1] "Train data percentage= 80.4071246819338"
```

```
feature_train <- train_data[, 3:dim(train_data)[2]]
label_train <- train_data[, 2]
feature_test <- test_data[, 3:dim(train_data)[2]]
label_test <- test_data[, 2]
```

Train CV Glmnet Here we tain glmnet using optimal λ value from cross validataion

```
c1 <- makeCluster(8)
registerDoParallel(c1)
cvglmnet <- cv.glmnet(feature_train, label_train,
  gamma = mix, family = gaussian(), parallel = TRUE
)
stopCluster(c1)

myglmnet <- glmnet(feature_train, label_train,
  alpha = mix, lambda = cvglmnet$lambda.min, family = gaussian()
)
```

```
exv_var_Score <- function(y_pred, y_true) {
  exv_var_Score <- 1 - var(y_true - y_pred) / var(y_true)
  return(exv_var_Score)
}
```

Explained variance score

```
pred_train <- predict(myglmnet, feature_train)
print(exv_var_Score(pred_train, label_train))
```

Train R^2

```
##           s0
## s0 0.3572763
```

```
pred_test <- predict(myglmnet, feature_test)
print(exv_var_Score(pred_test, label_test))
```

```
##            s0
## s0 0.04297494
```