testing_train_func.R

mac

Wed May 1 19:48:28 2019

```r
library(dplyr)
library(ROSE)
library(caret)
```

Below create three functions: `modelfit` uses caret::train() with no trControl() argument. `modelfit2` uses caret::train() with only trControl(classProbs=TRUE) `modelfit3` uses caret::train() with trControl(), with all of the defaults set manually and `number` set to a large number

```r
modelfit <- function(data){

  train <- data$train
  test <- data$test
  # fit the model on the training set
  fit <- train(
    Y ~ .,
    data = train,
    method="svmPoly"
  )

  # predict on the test set
  yhat = predict(fit, newdata = select(test, -Y))

  # evaluate test accuracy
  conf <- caret::confusionMatrix(yhat, test$Y)
  result <- c(conf$overall[1], conf$byClass[1:2]) #<-can change threshold if you want
  result["auc"] <- auc(test$Y, yhat)

  return(result)

}

modelfit2 <- function(data){

  train <- data$train
  test <- data$test
  # fit the model on the training set
  fit <- train(
    Y ~ .,
    data = train,
    method="svmPoly",
    trControl = trainControl(
      # method = "boot",
      # number = 25,
      # repeats = NA,
      # search = "grid",
      # p = 0.75,
      # initialWindow = NULL,
      # horizon = 1,
      # fixedWindow = TRUE,
      # skip = 0,
      # verboseIter = FALSE,
      # returnData = TRUE,
      # returnResamp = "final",
      # savePredictions = "none",
      classProbs = TRUE
    )
  )

  # predict on the test set
  yhat = predict(fit, newdata = select(test, -Y))

  # evaluate test accuracy
  conf <- caret::confusionMatrix(yhat, test$Y)
  result <- c(conf$overall[1], conf$byClass[1:2]) #<-can change threshold if you want
  result["auc"] <- auc(test$Y, yhat)

  return(result)

}

modelfit3 <- function(data){

  train <- data$train
  test <- data$test
  # fit the model on the training set
  fit <- train(
    Y ~ .,
    data = train,
    method="svmPoly",
    trControl = trainControl(
      method = "boot",
      number = 500,
      repeats = NA,
      search = "grid",
      p = 0.75,
      initialWindow = NULL,
      horizon = 1,
      fixedWindow = TRUE,
      skip = 0,
      verboseIter = FALSE,
      returnData = TRUE,
      returnResamp = "final",
      savePredictions = "none",
      classProbs = TRUE
    )
  )

  # predict on the test set
  yhat = predict(fit, newdata = select(test, -Y))

  # evaluate test accuracy
  conf <- caret::confusionMatrix(yhat, test$Y)
  result <- c(conf$overall[1], conf$byClass[1:2]) #<-can change threshold if you want
  result["auc"] <- auc(test$Y, yhat)

  return(result)

}
```

Additionally a function for train test splitting. I noticed you'd struggled to access the data object, we should file an issue on github

```
train_test <- function(dataset, folder){

  # issue here with namespace and "data" variable reference
  data <- dataset[-folder, ]
  mytest <- dataset[folder, ]

  # generate a balanced training set
  train <- ovun.sample(
    Y ~ .,
    data=data,
    method = "both",
    p = 0.6,
    seed = 1342
  )$data

  train <- as_tibble(train) %>%
    droplevels()
  test <- as_tibble(mytest) %>%
    droplevels()

  return(list(train = train, test = test))

}
```

Read in and preprocess the dataset here. Steps are: 1. Filter correct conditions for threshold and windows 2. Complete.cases() 3. Filter out high stress controls 4. Calculate Z scores within participants

```
df_time_ml <- read.csv(here('data', 'Preprocessing_data_outputs', 'Paper', 'data_out.csv'))%>%
  # use params threshold = 250, windows = 2
  filter(threshold == 250 & winds == 2)%>%    # filter dataset
  select(-winds, - threshold, -index) %>%
  filter(complete.cases(.)) %>%               # complete cases
  filter(!((Stress>=5) & (Y == "Control")))   # filter out high stress controls

df_time_ml <- df_time_ml %>%
  mutate(ID = as.factor(ID)) %>%
  group_by(ID) %>%                            # group by ID
  mutate_at(vars(SDNN:HRVi), scale) %>%       # scale each var within group
  ungroup() %>%
  select(Y:HRVi) %>%
  filter(complete.cases(.))                   # complete cases
```

Lastly, fit the models, using the same seed and `nfolds` as you'd specified.

```
# fit model with regular train()
set.seed(9)
nfolds <- 4
subdata<-createFolds(df_time_ml$Y, nfolds)
t1_z <- train_test(df_time_ml, subdata$Fold1) %>%
  modelfit()
t2_z <- train_test(df_time_ml, subdata$Fold2) %>%
  modelfit()
t3_z <- train_test(df_time_ml, subdata$Fold3) %>%
  modelfit()
t4_z <- train_test(df_time_ml, subdata$Fold4) %>%
  modelfit()
correct_result <- colMeans(rbind(t1_z, t2_z, t3_z, t4_z))
print(correct_result)
```

```
##    Accuracy Sensitivity Specificity         auc
##   0.6348286   0.6267806   0.7000000   0.6633903
```

```
# Accuracy Sensitivity Specificity         auc
# 0.6348286   0.6267806   0.7000000   0.6633903
```

The above are the optimal results reproduced exactly.

```
# fit model with train() and empty trControl()
set.seed(9)
nfolds <- 4
subdata<-createFolds(df_time_ml$Y, nfolds)
t1_z <- train_test(df_time_ml, subdata$Fold1) %>%
  modelfit2()
```

```
## maximum number of iterations reached 1.166296e-05 -1.164447e-05
```

```
t2_z <- train_test(df_time_ml, subdata$Fold2) %>%
  modelfit2()
```

```
## maximum number of iterations reached 0.0004040453 -0.0004034993
```

```
t3_z <- train_test(df_time_ml, subdata$Fold3) %>%
  modelfit2()
```

```
## maximum number of iterations reached -0.0002470484 0.0002469383maximum number of iterations reached -
```

```
t4_z <- train_test(df_time_ml, subdata$Fold4) %>%
  modelfit2()
```

```
## maximum number of iterations reached 8.947355e-05 -8.936378e-05maximum number of iterations reached 5
```

```
trcontrol_result <- colMeans(rbind(t1_z, t2_z, t3_z, t4_z))
print(trcontrol_result)
```

```
##    Accuracy Sensitivity Specificity         auc
##   0.5864415   0.5712251   0.7000000   0.6356125
```

```
# Accuracy Sensitivity Specificity         auc
# 0.5864415   0.5712251   0.7000000   0.6356125
```

These results are suboptimal. Notice that in this case the maximum iterations warning comes up.

```
# fit model with train() and trControl with defaults and unlimited `number` argument
set.seed(9)
```

```
nfolds <- 4
subdata<-createFolds(df_time_ml$Y, nfolds)
t1_z <- train_test(df_time_ml, subdata$Fold1) %>%
  modelfit3()
```

```
## maximum number of iterations reached 1.166296e-05 -1.164447e-05maximum number of iterations reached -
```

```
t2_z <- train_test(df_time_ml, subdata$Fold2) %>%
  modelfit3()
```

```
## maximum number of iterations reached 0.0004040453 -0.0004034993maximum number of iterations reached 1
```

```
t3_z <- train_test(df_time_ml, subdata$Fold3) %>%
  modelfit3()
```

```
## maximum number of iterations reached -0.0002470484 0.0002469383maximum number of iterations reached -
```

```
t4_z <- train_test(df_time_ml, subdata$Fold4) %>%
  modelfit3()
```

```
## maximum number of iterations reached 8.947355e-05 -8.936378e-05maximum number of iterations reached 5
```

```
trcontrol_large_n <- colMeans(rbind(t1_z, t2_z, t3_z, t4_z))
print(trcontrol_large_n)
```

```
##    Accuracy Sensitivity Specificity         auc
##   0.5781250   0.5584046   0.7000000   0.6292023
```

Again, suboptimal results, and even with `number` set to a very large value, we still get the maximum iterations warning.