Binary Classification with CARET package in R

Abstract

This R script can be easily reused to perfom binary classification by CARET package in R. Machine Learning algorithms' plots, results and resampling plots are exported as output files. Few guided adjustments are required to easily reuse the script with any dataset.

Contents

List of Figures

List of Tables

1 Introduction: Script overview

This R script can be easily reused to perfom binary classification by CARET package in R. Machine Learning algorithms' plots, results and resampling plots are exported. Few guided adjustments are required to easily reuse the script with any dataset. The sample dataset is Heart Disease UCI.

The R script consists of 5 main chunks of code which are summarized in the table below:

Chunk	**Description**	**Line of code**
Input actions	This chunck loads libraries, imports the dataset, creates folders to save plots of interest, definines a function to plot confusion matrices, and requires few adjustments by the user to make the script suitable for any dataset.	1 - 131
ML algorithms with CARET	This chunk implements 9 Machine Learning algorithms using CARET package. It splits data into training and testing, fits and trains models, exports models plots for variables' importance and confusion matrices with performance indicators as image file.	134 - 828
Models' resampling	This chunk resamples models, saves models' performances indicator in a dataframe which is exported as excel file, stores models' timings in order to study how script's total elapsed time is distributed.	832 - 925
Models' performances plots	This chunk creates a bar plot for each and every indicator taken into account in order to compare models' performances.	928 - 1078
Models' time elapsed plot	This chunk creates a bar chart to visualize how script's total elapsed time is distributed.	1080 - 1110
Main information storage	This chunk exports main algorithms' information in a *txt* file.	1113 - 1289

Table 1: Summary of code chunks

Following sections report details about main above mentioned code chunks.

Output files are organized as follows:

1. *figs* folder: it contains image files for confusion matrices, ROC curves, variables' importance and models' performances indicators;
2. *Console Output.txt* file: it contains total elapsed time and models' elapsed time information. It also provides models' details about classification performances and parameters' tuning;
3. *Models' Results.xlsx* file: it tables models' perfomance indicators which are taken into account;
4. *Time Elapsed.png* file: it plots how total elapsed time is distributed among ML algorithms.

2 Input actions

The first part of Input actioncs chunk sets the working directory, creates folders where to save exported plots and loads require libraries as shown in figure 1 below.

The user has to customize its own working directory on line 2 of the script.

```
1  #Set Working Directory
2  setwd("C:/Users/utente/Documents/R/GitHub_Projects/Binary_Classification_with_CARET")
3
4  library(tictoc)
5  tic("Since Star_Time")
6
7  dir.create("figs") #save plots
8  dir.create("figs/Confusion Matrices") #save Confusion Matrices' plots
9  dir.create("figs/ROC Curves") # Save ROC Curves plots
10 dir.create("figs/Models' Performances") # Save Models' Performances plots
11 dir.create("figs/Variables' Importance") # Save Variables' Importance plots
12
13 #Import packages
14 library(caret)
15 library(tidyverse)
16 library(readxl)
17 library(writexl)
18 library(dplyr)
19 library(ggplot2)
20 library(forcats)
21 library(usefun)
22 library(MLeval)
23 library(plotROC)
```

Figure 1: Set working directory and load libraries

The second part imports the case study's dataset, defines a percentage to split data into training and testing datasets, renames the target variable's column, renames target variable's outcomes as positive and negative, and sets values for the number of variables to plot and the length of the parameters' tuning grid. Related code is shown in figure 2. The user has to:

1. import its own preprocessed dataset by modifying line 25;
2. define a proportion by modifying the number on line 29;
3. rename target variable by changing the number of the column on line 32;
4. replace target variable's outcomes on lines 36 - 37;
5. set up number of variables to display in variables' importance plots on line 45;
6. set up number of tunelength for algorithms on line 48;

Lines of code in figure 3 coherce character colums as factors and remaining ones as numeric then they scale and center numeric variables. The loop verifies mean and standard deviation of numeric variables. If you have just one character colum, keep the code as it is.
If you have more than one character column, comment line 53 and uncomment line 54.
If the users needs, it can uncomment line 42 and replace the whole dataset with a sample of observation to test preliminary how long the script will take to run or test sampling methods for unbalanced datasets.

```r
#Import Dataset
data <- read_excel("heart disease.xlsx")
data <- as.data.frame(data)

#Define proportion for training dataset
proportion <- 0.7

#Rename Target Variable
names(data)[14] <- "Y"

#Rename Target Variable's categories as positive or negative
data <- as.data.frame(data %>%
                      mutate(Y=case_when((Y == 1) ~ "Positive",
                                         (Y == 0) ~ "Negative")))

table(data$Y) #Verify if your dataset is balanced or not

#Subsampling
#data <- sample_n(data, round(0.25*nrow(data),digits = 0))

#Set number of variables you want to be displayed in Variables' Importance plots
N <- ncol(data) - 1

#Set Tune Length for for tuning parameters
TuneLength <- 5
```

Figure 2: Input actions by the user

```r
#Coherce character columns to factors
char_col <- which(unlist(lapply(data, is.character)))
char_col
data[,char_col] <- as.factor(data[,char_col])
#data[,char_col] <- lapply(data[,char_col], factor)

#Coherce remaining columns to numeric
data[,-char_col] <- sapply(data[,-char_col], as.numeric)
data <- as.data.frame(data)

str(data)

#Centering and scaling
numeric_col <- which(unlist(lapply(data, is.numeric)))
numeric_col

data <- data %>%
        mutate_if(is.numeric, scale)

for (i in numeric_col) { #Verify if all numeric columns are standardized
  print(
    paste(
      names(data)[i],"has mean", round(mean(data[,i]),digits = 2),
      "and sd", sd(data[,i])

    )
  )
}
```

Figure 3: Coherce data and data centering - scaling

Lastly, figure 4 shows the function which generates confusion matrices' plots with related performance indicators.

```r
##################### Plot Confusion Matrix Function ########################
#############################################################################

# cm = confusionMatrix(...)

draw_CM <- function(cm,Title) {

  layout(matrix(c(1,1,2)))
  par(mar=c(2,2,2,2))
  plot(c(100, 345), c(300, 450),
       type = "n", xlab="", ylab="",
       xaxt='n', yaxt='n')
  title(Title, cex.main = 2.0)

  # create the matrix
  rect(150, 430, 240, 370, col='#72F069')
  text(195, 435, 'Negative', cex=1.8)
  rect(250, 430, 340, 370, col='#E84767')
  text(295, 435, 'Positive', cex=1.8)
  text(125, 370, 'Predicted', cex=1.9, srt=90, font=2)
  text(245, 450, 'Actual', cex=1.9, font=2)
  rect(150, 305, 240, 365, col='#E84767')
  rect(250, 305, 340, 365, col='#72F069')
  text(140, 400, 'Negative', cex=1.8, srt=90)
  text(140, 335, 'Positive', cex=1.8, srt=90)

  # add in the cm results
  res <- as.numeric(cm$table)
  text(195, 400, res[1], cex=1.8, font=2, col='black')
  text(195, 335, res[2], cex=1.8, font=2, col='black')
  text(295, 400, res[3], cex=1.8, font=2, col='black')
  text(295, 335, res[4], cex=1.8, font=2, col='black')

  # add in the specifics
  plot(c(100, 0), c(100, 0), type = "n", xlab="", ylab="",
       main = "DETAILS", xaxt='n', yaxt='n')
  text(10, 85, names(cm$byClass[1]), cex=1.6, font=2)
  text(10, 70, round(as.numeric(cm$byClass[1]), 5), cex=1.6)
  text(30, 85, names(cm$byClass[2]), cex=1.6, font=2)
  text(30, 70, round(as.numeric(cm$byClass[2]), 5), cex=1.6)
  text(50, 85, names(cm$byClass[5]), cex=1.6, font=2)
  text(50, 70, round(as.numeric(cm$byClass[5]), 5), cex=1.6)
  text(70, 85, names(cm$byClass[6]), cex=1.6, font=2)
  text(70, 70, round(as.numeric(cm$byClass[6]), 5), cex=1.6)
  text(90, 85, names(cm$byClass[7]), cex=1.6, font=2)
  text(90, 70, round(as.numeric(cm$byClass[7]), 5), cex=1.6)

  # add in the accuracy information
  text(30, 35, names(cm$overall[1]), cex=1.6, font=2)
  text(30, 20, round(as.numeric(cm$overall[1]), 3), cex=1.6)
  text(70, 35, names(cm$overall[2]), cex=1.6, font=2)
  text(70, 20, round(as.numeric(cm$overall[2]), 3), cex=1.6)
}
```

Figure 4: Confusion Matrix function

3 ML algorithms with CARET

This chunk of code implements 9 algorithms of supervised learning. ML algorithms are tabled below with related lines of code in the R script:

Algorithm	Lines of code
Naive Bayes	173 - 248
Linear SVM	250 - 321
Radial SVM	324 - 392
Polynomial SVM	395 - 466
Neural Network	468 - 537
Decision Tree (CART)	540 - 609
K-Nearest Neighbors	613 - 681
Logistic Regression	683 - 755
Random Forest	756 - 829

Table 2: Machine Learning algorithms for binary classification

Figure 5 reports the lines of code related to data partitioning and set up of the *trainControl()* function. If the user is dealing with an unbalanced dataset, it can uncomment line 165 to use a sampling method. CARET provides 4 sampling methods for among which 2 methods require a related library the user can load by uncommenting line 157 or 158 according to the choice it makes. The user can also customize the *trainControl()* parameters.

```
134 ############################## ML Algorithms ###################################
135 ################################################################################
136
137 #Data Splitting
138 set.seed(120496)
139
140 inTrain <- createDataPartition(y = data$Y,
141                                p = proportion,
142                                list = FALSE)
143 training <- data[inTrain,]
144 dim(training)
145 table(training$Y)
146 testing  <- data[-inTrain,]
147 dim(testing)
148 table(testing$Y)
149
150 # Class proportion in training dataset for target variable
151 table(training$Y)
152 prop.table(table(training$Y))
153 #################
154
155 #setup trainControl
156
157 #library(themis) #for SMOTE sampling for unbalanced datasets
158 #library(ROSE)   #for ROSE sampling for unbalanced datasets
159
160
161 fitControl <- trainControl(method = 'cv',
162                            number = 5,
163                            verboseIter = FALSE,
164                            classProbs = TRUE,
165                            #sampling = "down",
166                            savePredictions = TRUE,
167                            search = "grid")
```

Figure 5: Data partitioning and fit control

As reported in table 2, lines 173 - 829 implement 9 algorithms for binary classification.
This guide reports an explanation about only the Naive Bayes algorithm being rationale and used fuctions equal and repetitive for each and every ML algorithm.

Figure 6 imports the library and defines a grid the user can customize in lines 175 - 177.
As a default option the code runs with the *tuneLegnth* option.
If the user prefers setting the grid manually, it can comment line 186 and uncomment line 185.
The chunk code also measures the algorithm's time elapsed by using *tic()/toc()* functions then it exports measurements on *Console Output.txt* file.

```
172                          ###### Naive Bayes
173 library(naivebayes)
174
175 grid_NB <-  expand.grid(laplace = seq(0,0.25,0.05),
176                          usekernel = c(TRUE,FALSE),
177                          adjust = seq(0.90,1.15,0.05))
178
179 tic("Naive Bayes")
180
181 #setup trainControl
182 NB <- train(Y ~ .,
183             data = training,
184             method = "naive_bayes",
185             trControl= fitControl,
186             #tuneGrid = grid_NB,
187             tuneLength = TuneLength)
188
189
190
191 ELT <- toc()
192
193 sink("./Console output.txt",append = TRUE)
194 print(paste(ELT$msg,(ELT$toc - ELT$tic), "sec elapsed"))
195 print(paste(ELT$msg,(ELT$toc - ELT$tic)/60, "min elapsed"))
196 print(paste(ELT$msg,(ELT$toc - ELT$tic)/3600, "hours elapsed"))
197 print_empty_line(html.output = FALSE)
198 print_empty_line(html.output = FALSE)
199 sink()
```

Figure 6: Algorithms' train control

Figure 7 outputs on console the algorithm's details by lines 201 - 205.
Lines 207 - 215 plots algorithm's ROC curve and exports it as image file the it also store the AUC value. Lines 217 - 288 plots and exports as image file the variables' importance chart.

Lastly, figure 8 compotes predictions on the testing dataset and outputs confusion matrix on console.
Confusion matrix is also exported as image file.

```r
NB
summary(NB)
NB$finalModel
NB$results
NB$bestTune

#ROC and AUC
x <- evalm(NB)
x$roc
ggsave(paste("ROC curve of ",NB$modelInfo$label,".png"),
       path = "./figs/ROC Curves")
x$stdres
y<- as.data.frame(x$stdres)
auc_NB <- y[13,1]
auc_NB

#Variables' Importance
VarImp_NB <- varImp(NB, scale = TRUE)
VarImp_NB
CM <- plot(VarImp_NB,
           main = paste("Variables' Importance in", NB$modelInfo$label),
           type = "b", cex=1.3,top = N)
CM

png(filename = paste("./figs/Variables' Importance/",NB$modelInfo$label,".png"),
    width = 490, height = 580, units = "px", pointsize = 15)
CM
dev.off()
```

Figure 7: Algorithms' details and ROC/AUC

```r
#Prediction
prediction_NB  <- predict(NB,testing)
prediction_NB

#Confusion matrix
confusion_matrix_NB <- confusionMatrix(prediction_NB,
                                       testing$Y,
                                       positive = "Positive")
confusion_matrix_NB


draw_CM(confusion_matrix_NB,paste("Confusion Matrix in", NB$modelInfo$label))


png(filename = paste("./figs/Confusion Matrices/", NB$modelInfo$label,".png"),
    width = 900, height = 900, units = "px", pointsize = 25)
draw_CM(confusion_matrix_NB,paste("Confusion Matrix in", NB$modelInfo$label))
dev.off()
```

Figure 8: Algorithms' prediction