Code Deliverables

Interpretable Predictions of Mortality in ICU Patients

by

Shu Yu Tew • Yee Voon Lee

Contents

1 Process/Project Management

1.1 Use of revision control system

When working in pairs, this usually means carrying out pair programming. Both partners may touch on similar chunks of code throughout the project, and changes made in one part of the code may result in incompatibility if both sides are working on the same part at the same time. This is where version control comes in real handy.

There are 2 types of version distributed control systems (DVCS), and centralized the control system. We will be using DVCS in our project as we can easily clone a copy of a repository locally. Bitbucket is used as a "central cloud" that allows us to store our code and other files required for the project, this can be shared and collaborate with other team member. Any changes made to the codes or files will be locally committed then pushed/pulled using Git.

1.2 Documented end-to-end process/ report generation

The end-to-end process of our project is very simple and straightforward. We started off with understanding the topic given, do as much research as possible alongside documenting our findings. After grasping the knowledge of the topic, we then start coming up with more solid plans for the entire project process. Once we came up with the solid plan, we start to execute the tasks in the plans. Throughout this process, there were many unforeseen events faced and we had to counter each one that comes by.

1.3 Backup procedures

We mainly use Google Drive to backup our files and code, as it is quite reliable and easily accessible anywhere. Codes and documentation will be uploaded/created in google drive and shared amongst both team members. The code files can then be downloaded anywhere; we mainly use it with other machines to run codes that require a long computing time.

2 Code Documentation

There are 7 total Rscript file (cleanCode.R, feature_creation.R, modelling.R, MICE.R, explainer_model.R, distribution.R and all _function.R). The sequence in which the R scripts should be executed are as listed below:

1. cleanCode.R
    - compile all patients' data into one csv file (3 separate csv files for different set of data – A, B and C. Users have to manually change working directory to folders, three different folders for 3 different dataset.)
    - Created binary variable for each of the parameters.
2. feature_creation
    - - take in output from cleanCode.R and continue data preprocessing.
    - Similar to cleanCode.R, users have to manually change the file names in the read.csv() code line to produce another 3 sets of csv files.
    - This is the Rscript file that produces the 3 csv files (SETA, SETB and SETC) provided.
    - Summary variables are created. (mean, max, min and standard deviation)
3. modelling or MICE (require to source all_function.R)
    - Both of these Rscripts are similar, the only difference is that MICE.R contains a chunk of code for predictive mean matching imputation and all the models trained in MICE.R is using SMOTE oversampling.
4. explainer_model
5. supporting Rscript:
    - distribution.R (to produce a pdf file with distributions of all variables in the dataset)

Below is the documentation for all functions created in the all_function.R file. These are the functions used throughout modelling process. Before using any of the functions below, this line of code need to be run first: 'source("all_function.R")'

Install_required_packages

Description
Install all packages required to run all the Rscript files.

Usage
install_required_packages()

Arguments
*no arguments required

Find_score

Description
Compute a score to evaluate the performance of the predictive model

Usage
find_score(table)

Arguments
table a 2x2 contingency table or matrix consisting numeric values

Examples:

predValid	0	1
0	869	119
1	3	9

Figure 1: find_score

find_uncertainty

Description
Compute a value to measure the uncertainty of the scores

Usage
find_score(score_table)

Arguments
score_table a dataframe of numerical values

initialize_RF

Description
Build a random forest model

Usage
Initialize_RF(input_data, test_set, nt = 100)

Arguments
input_data	a dataframe containing the variables and predictors for the model to be fitted
test_data	a dataframe containing the variables and predictors for test set
nt	number of trees to grow. A numerical value.

Return
A list containing the model score and a dataframe containing top 50 variables selected by the model.

repeat_RF

Description
Build a random forest model at n iteration. This function follows up with initialize_RF.

Usage
repeat_RF(input_data, test_set, nt = 100, score_table, variable_table, iter)

Arguments
input_data	a dataframe containing the variables and predictors for the model to be fitted
test_data	a dataframe containing the variables and predictors for test set
nt	number of trees to grow. A numerical value.
score_table	a dataframe containing score. This is the output of initialize_RF
variable_table	a dataframe with a list of important variables names. This is the output of initialize_RF
iter	a numerical value. The number of iterations to run the model. How many scores you want to get?

Return
A list containing the model score and a dataframe containing top 50 variables selected by the model.

repeat_model

Description

This function fit a number of classification and regression routines, imputed training and test set with selected imputation methods before oversampling based on users' choice.

Usage

repeat_RF(input_data, test_set, model, log_trans = T, standardize = T, oversample, iter)

Arguments

input_data	a dataframe containing the variables and predictors for the model to be fitted
test_data	a dataframe containing the variables and predictors for test set
model	a string specifying which classification or regression model to use. Possible inputs are: "Bayes_GLM" – Bayesian Generalized Linear Model "Decision Tree" "SVM" – Support Vector Machine
log_trans	A logical: should the data be log transformed?
standardize	A logical: should the data be standardized?
oversample	a string specifying which oversampling method to use. Possible inputs are: "ROSE" – Random OverSampling Example "SMOTE" – Synthetic Minority Oversampling TEchnique
iter	a numerical value. The number of iterations to run the model. How many scores you want to get?

Return

A list containing the model score and a dataframe containing up to top 50 variables selected the model.

3 Code Documentation for incomplete code

There are in total of 3 Rscript file (ui.R, server.R and app_modelling.R) that are created to deploy the standalone program.

1. ui.R
   This R script is created for the UI of the web app, it should always have a basic skeleton that looks like this 2. The (ui) object controls the layout and appearance of the web app. It contains also widgets such as textInput, actionButton, slideInput and etc.

```
ui <- fluidPage(
  titlePanel("title panel"),

  sidebarLayout(
    sidebarPanel("sidebar panel"),
    mainPanel("main panel")
  )
)
```

Figure 2: ui.R

2. server.R
   This R script is created as the server for the web app, without this file, the web app will not be able to run. The (server) function contains the instructions that your computer needs to build your app. It should always have a basic skeleton that looks like this 4.

```
# Define server logic ----
server <- function(input, output) {

}
```

Figure 3: server.R

3. app_modelling.R
   This R script should be sourced in the server.R file to carry out the prediction process, however it is incomplete. It contains a function that rbinds the user input data and training set, then impute necessary variables and finally put into the SVM model to predict for outcome. However there were some issues that could not be solved in time.

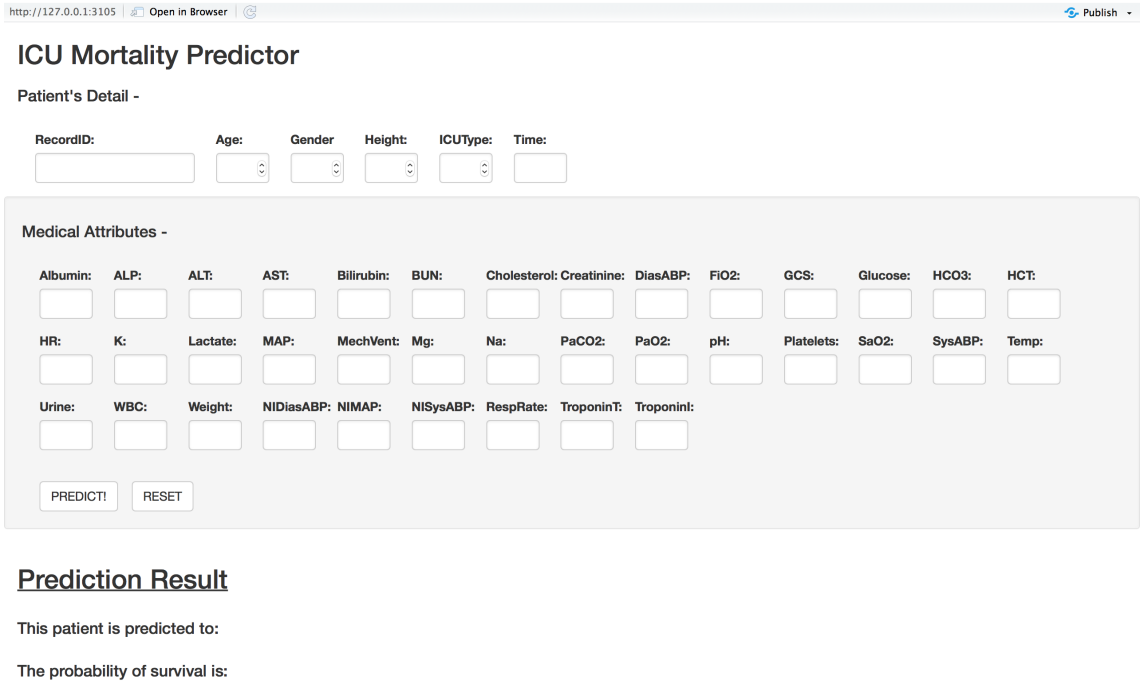

Figure 4: Interface of the program