Tutorial for the Analysis of Sokoban Logfiles using Machine Learning (2)

Using R to process extracted data from logfiles

Introduction

In this part, we are going to use R to manipulate the data. We will generate more features, reshape the data from long format to wide format, and finally feed it back to Python.

Actually, the package Pandas in Python is similiar with R. Most procedures we do in R can also be replicated with Pandas. The reason I change to R are here: First, R is more handy in data manipulation. As Pandas is a newly-developed package, Some sophisticated method which can be implemented in R easily may require lengthier code in Pandas.

Second, it is beneficial to learn a bit of R. Python is a programming language for general-purpose programming, while R, developed by statisticians, is especially for statistics by design. The package Psych in R provides almost all the tools useful for psychologists. The package mirt may be the best software for multidimensional IRT.

The last reason is that I'm more familiar with R than Pandas.

Though using R is highly recommended, you still can choose Pandas to do the job, and I believe except for the IRT procedures in the following R code may requires additional packages or softwares, all other procedures can be done by Pandas. You can get the official documentation of Pandas on its website.

Useful Packages

There are several packages useful in R which we'll use later. I will introduce those packages when I use them.

```
library(data.table)
library(dplyr)
library(stringi)
library(mirt)
library(foreign)
library(psych)
```

Generate more features

First of all, we read the csv file using the function "fread", provided by data.table, which enables fast reading of csv file.

```
dt = fread(file.path('working_data','extracted_data.csv'))
head(dt,3)
```

```
##        id box_completed dsteps duplicates first_move_time first_push_time
## 1: 190101             0    -27          0       10.300050       12.800100
## 2: 190101             1      0          0        3.339998        7.959993
## 3: 190101             3      0          0        9.620030       11.240070
##    guan_time num num_higher_than_sd optstep p_optimal rest_mean   rest_sd
## 1:  34.00059   1                  1      45 0.1777778 0.5875134 0.6097752
## 2:  14.58014   2                  2      17 0.9411765 0.4014366 0.9075092
## 3:  19.90026   3                  2      20 1.0500000 0.4517750 0.2989874
```

```
##    result_flag same_with_optimal steps
## 1:           3                 8    18
## 2:           1                16    17
## 3:           1                21    20
```

We can generate more features based on the exsisting.

```
dt[,success:= result_flag<=2]
dt= rename(dt, same_p_optimal = p_optimal)
dt[, p_dup:=duplicates/optstep]
dt[,f_m_ratio:= first_move_time/rest_mean]
dt[,fp_m_ratio:= first_push_time/rest_mean]
dt[,f_t_ratio:= first_move_time/guan_time]
dt[,fp_t_ratio:= first_push_time/guan_time]
dt[,p_higher:=num_higher_than_sd/steps]
dt[,maxbox := max(box_completed, na.rm = T), by = num]
dt[,comp_rate:= box_completed/maxbox]
dt[,dbox:=maxbox - box_completed]
dt[,c('maxbox', 'optstep'):=NULL] # When you manipulate two variables together,
#you have to use c() to concantenate them.
dt[,time_window := quantile(guan_time, 0.4), by = num]
dt[, suc_within := success & guan_time<=time_window]
dt[,time_window:=NULL] #setting them equal to NULL means dropping them
```

':=' is a useful feature provided by data.table. It creates or updates variables based on the righthand expression. For example, the fist line means create a new variable "success" to denote whether result_flag is lower than or equal to 2.

You can inspect the data after each change in the table above.

Now we average a set of variables across each Guan by each student and whether the guan is successful or not.

```
# the variable list needs to be averaged
mean_var = c("box_completed","dsteps",
             "first_move_time","first_push_time",
             "guan_time","same_p_optimal","rest_mean","rest_sd",
             "p_dup","f_m_ratio","fp_m_ratio","f_t_ratio",
             "fp_t_ratio","p_higher","comp_rate","dbox")
# select those successful Guans, and calculate the mean of those variables by each student.
sucdt = dt[success==T, lapply(.SD, mean, na.rm = T), by  = id,.SDcols = mean_var]
#.SD means subset of data.table. See the documentation of data.table.
# since comp_rate, dbox, box_completed are constant in successful Guans, we drop them.
```

After obtaining the mean, we transform some variables into the log form.

```
lntrans = c("first_move_time","first_push_time","guan_time",
            "f_m_ratio","fp_m_ratio","f_t_ratio","fp_t_ratio")
for (var in lntrans){
  lnvar = 'ln' %s+% var # %s+% is a function from stringi package,
  # which concatenates two strings together.
  sucdt[,(lnvar):=log(get(var))]
}
```

Then we do the same procedures to the failed Guans.

```
faildt = dt[success==F, lapply(.SD, mean, na.rm = T), by  = id,.SDcols = mean_var]
lntrans = c("first_move_time","first_push_time","guan_time",
```

```
            "f_m_ratio","fp_m_ratio","f_t_ratio","fp_t_ratio")
for (var in lntrans){
  lnvar = 'ln' %s+% var
  faildt[,(lnvar):=log(get(var))]
}
```

Then we merge these two subsets using 'id'.

```
ndt = merge(sucdt, faildt, by = 'id', all.x = T, all.y = T)
```

Using IRT to estimate ability

Every Guan can be treated as an item, the number of boxes completed by subjects is the score ranging from 0 to 3. Thus a polymotous IRT model can be implemented here to estimate subjects problem-solving ability. Morevoer, we can set a posterior time window, for example, the 30th percentile of the time subjects spent on each guan. Those who complete all boxes within the time window are coded as 1 and others are all coded as 0. In this way, each guan is a dichotomous item, which captures subjects' speed as well as intelligence.

Following code implements two IRT models above. If you cannot get the point of the code, please input it step-by-step (splitted by %>%) in the console and watch how each step influence the result.

```
ndt = dcast(dt, id~num, value.var = 'box_completed') %>% select(.,-id) %>%
  mirt(.,1,itemtype = 'graded', verbose = F) %>% fscores() %>% cbind(ndt,.) %>%
  rename(.,suc_theta = F1)
```

dcast is a function to reshape the data from long format to wide format. See ?dcast.

%>% is a function from dplyr, means piping the result of the left hand expression to the right-hand function. You can input ?%>% in the console to see the documentation of %>%.

mirt is for fitting IRT model, and "fscores" for calculating the ability from the model.

```
ndt = dt[,suc_within:=as.numeric(suc_within)] %>%
  dcast(., id~num, value.var = 'suc_within')  %>%
  select(.,-id) %>% mirt(.,1,itemtype = 'Rasch', verbose = F) %>%
   fscores %>% cbind(ndt,.) %>% rename(., suc_window = F1)
```

After these steps, we get our features all prepared.

Merge with y variable

Here we need to merge the features with y variables.

```
yvar = read.spss(file.path('raw_data','Grade 1+2-others-0313.sav'), to.data.frame = T) %>%
  # read.spss comes from the foreign package.
  select(., ID, contains('TM'), Raven) %>% data.table
# We only select math grades and Raven scores.
yvar[,Rperc := rank(Raven)/sum(!is.na(Raven))]
# calculate the percentile of Raven scores.
yvar[Rperc>1,Rperc:= NA] # those whose percentiles are larger than 1 is NA
fares = select(yvar, contains('TM')) %>% fa
# the function 'fa' from the package 'psych' is for exploratory factor analysis
yvar = fares$scores %>% cbind(yvar,.)
yvar[,Mperc := rank(MR1)/sum(!is.na(MR1))]
# MR1 is the latent variable estimated by EFA
yvar[Mperc>1,Mperc:= NA]
```

```r
ndt = merge(yvar, ndt, by.x = "ID", by.y = "id", all.x = T, all.y = T)

Mpolar = ndt[Mperc>=0.75] %>% mutate(., Mperc=1)
#mutate is from dplyr. We recode those top 25% as 1, and those bottom 25% as 0.
Mpolar = ndt[Mperc<=0.25] %>% mutate(., Mperc=0) %>% rbind(.,Mpolar) %>%
  select(., -(ID:MR1))
Mpolar = Mpolar[complete.cases(Mpolar),] # keep the complete cases only.
write.csv(Mpolar, file.path('working_data','Mpolar.csv'))

Rpolar = ndt[Rperc>=0.75] %>% mutate(., Rperc=1)
Rpolar = ndt[Rperc<=0.26] %>% mutate(., Rperc=0) %>% rbind(.,Rpolar) %>%
  select(., -(ID:Raven), -(MR1:Mperc))
Rpolar = Rpolar[complete.cases(Rpolar),]
write.csv(Rpolar, file.path('working_data','Rpolar.csv'))
```

Now we get two datasets, Mpolar containing the the top 25% and bottom 25% students in math, and Rpolar containing the the top 25% and bottom 25% students in Raven.

It's time for machine learning.