main.R

ardutta

Thu Dec 7 01:16:24 2017

```r
############### README  #########################

# The project has couple of files to modularise the functionalities
# 1. config.yml:  has the configurations needed
# 2. data.R:      all data related manipulations and reads
# 3. init.R:      loads all the libraries and sets some options
# 4. model.R:     actual ML code implementation
# 5. RA_sol.R:    main file which should be run and calls the above files
#
# Unzip and set working directory to the unzipped directory.
# set the training and test files paths in the config.yml
# Generates few plots,
# Prints AUC for few algorithms



############### LOAD THE MODULES AND LIBRARIES #######################

# some system constants
source("init.R", local = TRUE)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

## 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

## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift

##
## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':
##
##     nasa
```

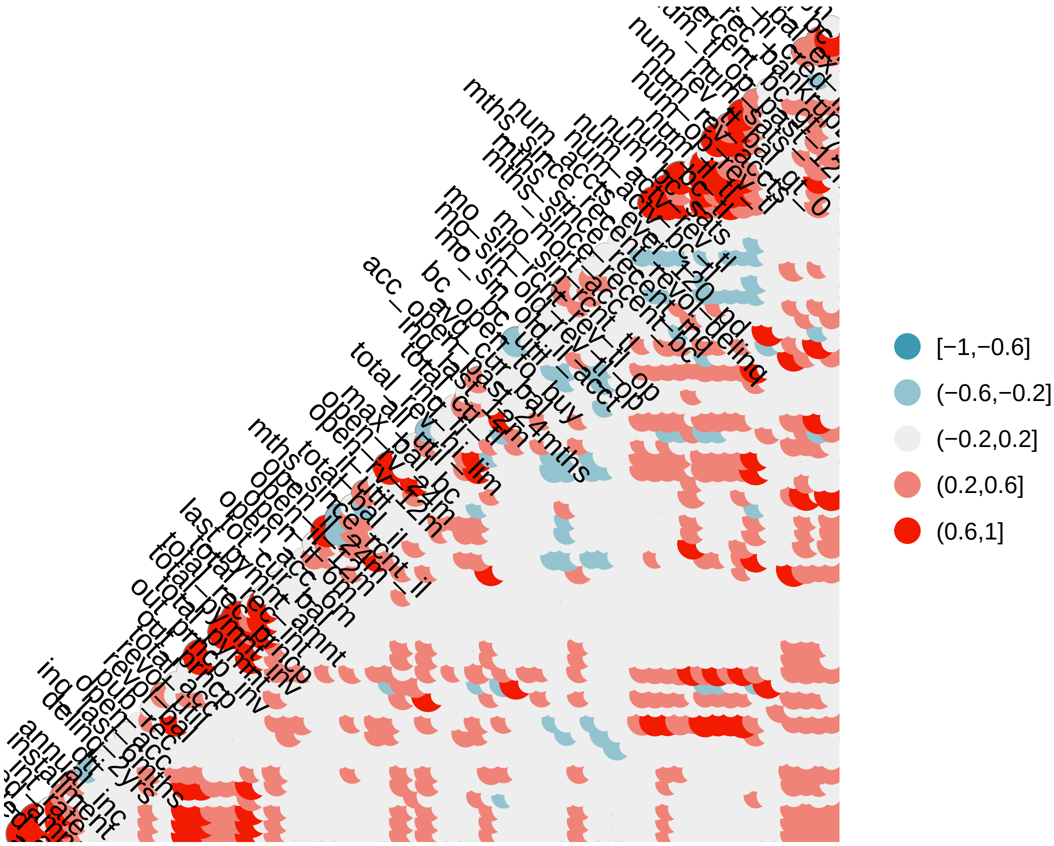
```
# load the data files. having all data manipulation tasks
source("data.R", local = TRUE)

# the actual model training funtions are in this module
source("model.R", local = TRUE)

# read the configuration file, once loaded all the configs can be accessed by
# using '$' on the following object
configurations <- get.configurations(path = "config.yml")

############### LOAD THE DATA  ##########################

# load the training data and the test data
train.df <-
  get.data(path = configurations$training, seperator = ",")
```

```
## [1] "Data loaded from RAcredit_train.csv"
```

```
test.df <- get.data(path = configurations$testing, seperator = ",")
```

```
## [1] "Data loaded from RAcredit_test.csv"
```

```
############### GENERIC TRANSFORM ##########################

# drop columns which are more than allowed.NA.levelpercentage of empty values
train.df.temp <-
  drop.columns(train.df, allowed.NA.level = configurations$NA.level)

# Drop highly correlated numerical features
drop <-
  drop.correlated.features(train.df.temp, threshold = configurations$corr.thresh)
```




```
train.df.temp <- train.df.temp[,!colnames(train.df.temp) %in% drop]

# generate a test dataframe with same columns
test.df.temp <-
  test.df[, colnames(test.df) %in% colnames(train.df.temp)]

################ CUSTOM TRANSFORM ###########################

# transform the cleaner data set for feature generation.
train.df.temp <- feature.generate(train.df.temp)
test.df.temp <- feature.generate(test.df.temp)
```

```
## Warning in feature.generate(test.df.temp): NAs introduced by coercion
```

```
# dropping the malformed rows
test.df.temp <- subset(test.df.temp,!is.na(annual_inc))

################ FEATURE SCALE ###########################

#Preprocessing and Imputing
preProcValues <-
  preProcess(train.df.temp, method = c("medianImpute", "center", "scale"))
train.df.temp <- predict(preProcValues, train.df.temp)
test.df.temp <- predict(preProcValues, test.df.temp)

# This drastically effects the training time, since, label-encoding factors, is better
# compared to one-hot encoding in terms of feature space expansion.
```

```r
# convert factors to numeric
character_vars <- lapply(train.df.temp, class) == "factor"
train.df.temp[, character_vars] <-
  lapply(train.df.temp[, character_vars], as.numeric)

character_vars <- lapply(test.df.temp, class) == "factor"
test.df.temp[, character_vars] <-
  lapply(test.df.temp[, character_vars], as.numeric)

# make target back to factor
train.df.temp$l_state <- as.factor(train.df.temp$l_state)
levels(train.df.temp$l_state) <- c("Default", "Fully.Paid")

test.df.temp$l_state <- as.factor(test.df.temp$l_state)
levels(test.df.temp$l_state) <- c("Default", "Fully.Paid")


############### CLASS IMBALANCE CHECK #########################

prop.table(table(train.df.temp$l_state)) * 100
```

```
##
##    Default Fully.Paid
##   2.648954  97.351046
```

```r
############### DATA PARTITION #########################

# drop NAs in target
train.df.temp <- subset(train.df.temp,!is.na(l_state))
train.df.temp <- train.df.temp[complete.cases(train.df.temp),]

# split train-test
tr_idx <-
  createDataPartition(train.df.temp$l_state, p = 0.8, list = FALSE)
training <- train.df.temp[tr_idx,]
testing <- train.df.temp[-tr_idx,]

# unblock to train on a smaller dataset
#training <- head(training, 50000)


# ADD WEIGHTS as one solution to the imbalance
# also try up/down sampling or SMOTE
# Create model weights
# Current implementaion allows either sample weights or UP sampling.
model_weights <- ifelse(training$l_state == "Default",
                        (1 / table(training$l_state)[1]) * 0.5,
                        (1 / table(training$l_state)[2]) * 0.5)

############### MODEL TRAINING #########################

print(paste0("Models training,, "))
```

```
## [1] "Models training,, "
```

```r
# LogReg Model ############
model_glm <-
  get.model.factory(data = training,
                    type = "glm",
                    model_weights = NA)
```

```
## Loading required package: plyr

## ------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following object is masked from 'package:purrr':
##
##     compact

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## Loading required package: mboost

## Loading required package: stabs

## This is mboost 2.8-0. See 'package?mboost' and 'news(package  = "mboost")'
## for a complete list of changes.

##
## Attaching package: 'mboost'

## The following object is masked from 'package:ggplot2':
##
##     %+%

## Aggregating results
## Fitting final model on full training set
## [1] "glm  algorithm takes 14.0062670707703 seconds"
```

```r
# GBM Model ############
model_gbm <- get.model.factory(data = training,
                               type = "gbm",
                               model_weights = NA)
```

```
## Loading required package: gbm

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines
```

```
## Loaded gbm 2.1.1

## Aggregating results
## Fitting final model on full training set
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.2948             nan     0.1000    0.0456
##      2        1.2205             nan     0.1000    0.0371
##      3        1.1596             nan     0.1000    0.0305
##      4        1.1090             nan     0.1000    0.0253
##      5        1.0668             nan     0.1000    0.0211
##      6        1.0315             nan     0.1000    0.0176
##      7        1.0022             nan     0.1000    0.0147
##      8        0.9775             nan     0.1000    0.0124
##      9        0.9566             nan     0.1000    0.0104
##     10        0.9392             nan     0.1000    0.0088
##     20        0.8502             nan     0.1000    0.0027
##     40        0.7793             nan     0.1000    0.0008
##     50        0.7613             nan     0.1000    0.0007
##
## [1] "gbm  algorithm takes 26.6270878314972 seconds"
```

```r
# extrem GB ############
model_xgb <- get.model.factory(data = training,
                               type = "xgb",
                               model_weights = NA)
```

```
## Loading required package: xgboost

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice

## Aggregating results
## Selecting tuning parameters
## Fitting nrounds = 50, max_depth = 1, eta = 0.4, gamma = 0, colsample_bytree = 0.6, min_child_weight
## [1] "xgb  algorithm takes 25.8777389526367 seconds"
```

```r
# RF Model ############
model_rf <- get.model.factory(data = training,
                              type = "rf",
                              model_weights = NA)
```

```
## Loading required package: e1071

## Loading required package: ranger

## Fitting mtry = 19 on full training set
## Growing trees.. Progress: 40%. Estimated remaining time: 46 seconds.
## Growing trees.. Progress: 80%. Estimated remaining time: 15 seconds.
## [1] "rf  algorithm takes 1.39695733388265 seconds"
```

```r
############### MODEL PERFORMANCE ##########################

# ONE  VIEW, report the AUC and plot the ROC curve for each #####
model_list <- list(
  gbm = model_gbm,
```

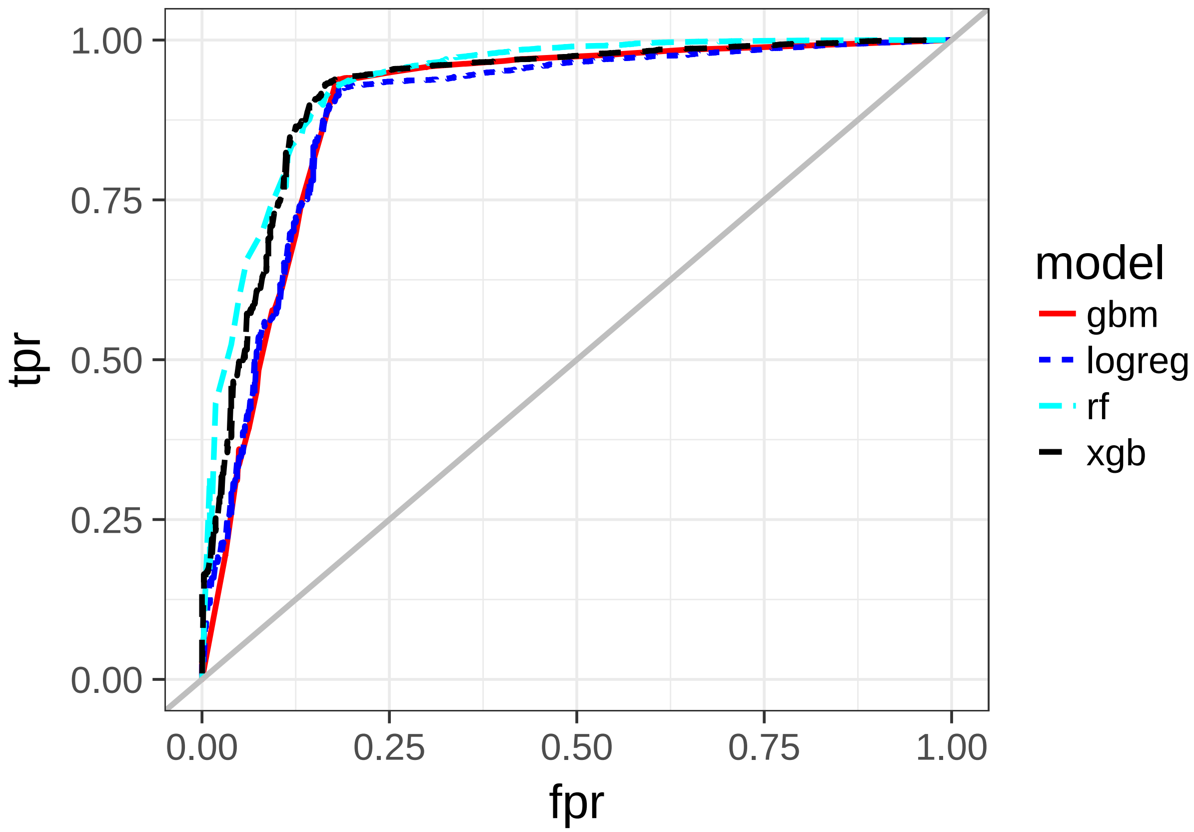
```r
  rf = model_rf,
  logreg = model_glm,
  xgb = model_xgb
)

# print the AUC scores
model_list_roc <- model_list %>%
  map(test_roc, data = testing)

model_list_roc %>%
  map(auc)
```

```
## $gbm
## Area under the curve: 0.8942
##
## $rf
## Area under the curve: 0.9318
##
## $logreg
## Area under the curve: 0.8908
##
## $xgb
## Area under the curve: 0.9185
```

```r
print(paste0("ROC Curves plots.. "))
```

```
## [1] "ROC Curves plots.. "
```

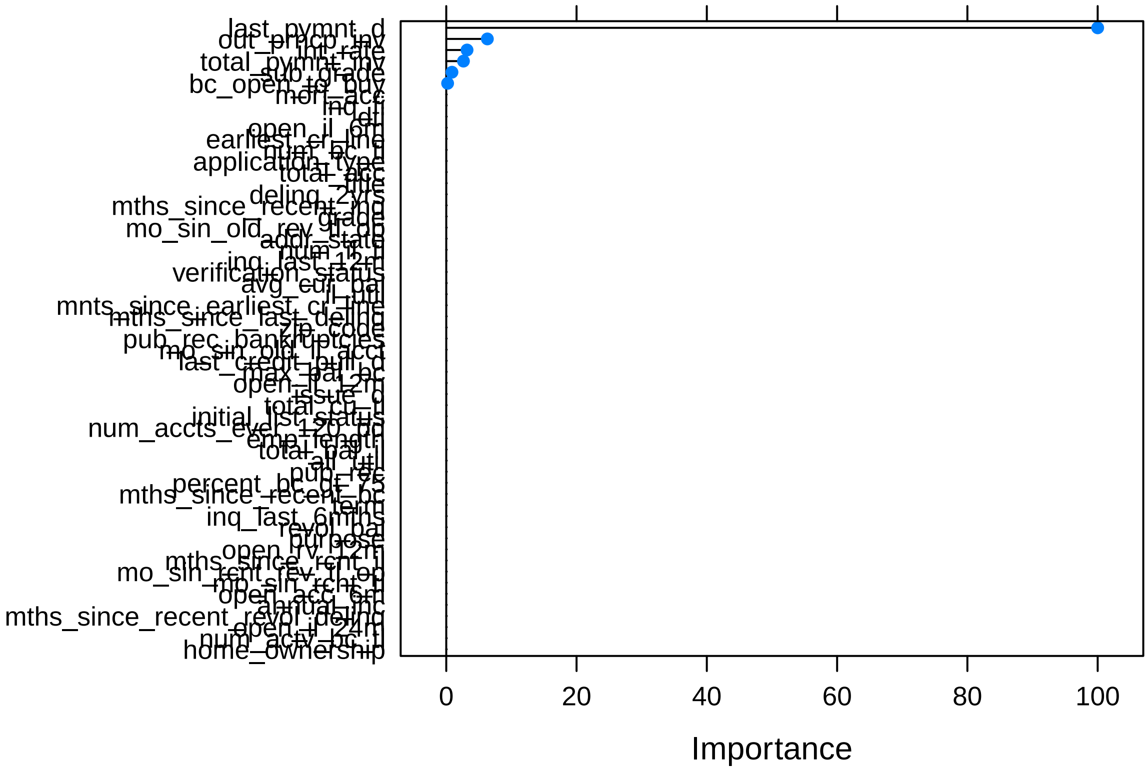
```r
# view the ROC
view.data(model_list_roc)
```




```r
print(paste0("Variable Importance plot... "))
```

```
## [1] "Variable Importance plot... "
```

```r
# plot the variable importance
plot(varImp(model_gbm, scale = TRUE))
```




```
############### Q-A #########################

#2. Do you face any problem and how would you solve them?
# The data is heavily unbalanced. One can us saple class weights to weight the under-
#represented class highly
# OR we can 'up' or 'down' sampling.

#6. What are the assumptions and limitations of your model?
# Did not check for corelation between the categorical columns.
# No exhaustive tuning for hyper parameters for the training.

#7. This model should be an easy implementation, if you would have a higher
#budget and more time, what could you provide in addition to this approach?
# Firstly, perform more of  features engineering and improve
# Try k-fold validations on larger ensembles.
# Try stacked ensemble or ANN based aproaches.
```