~~~
---
title: "DS Classification Exercise"
author: "Daisy Shi"
output: html_document
date: "2022-09-09"
---



#step 1 - clean and prepare data
#step 2 - build models
#step 3 - generate predictions
#step 4 - compare models
#step 5 - submit output work

```{r, message=FALSE, warning=FALSE}
library(knitr)
library(plyr)
library(corrplot)
library(gridExtra)
library(scales)
library(randomForest)
library(psych)
library(tidyverse)
library(tidymodels)
library(discrim)
library(baguette)
library(bonsai)

```
~~~

~~~~
# step 1 - clean and prepare data

```{r}
#  read train and test data
train<- read.csv("C:\\Users\\SSE6\\Downloads\\exercise_40_train.csv")
test<- read.csv("C:\\Users\\SSE6\\Downloads\\exercise_40_test.csv")


#check data type
str(train)
dim(train)
str(test)
dim(test)

# merge all data together for missing value imputation. We don't really need deal with missing value in the test data if this data is only for generate prediction.
# however, I decided to handle the missing as well. (not hurt to do one extra step)
all<- merge(train, test, all=TRUE)


```

First of all, I would like to see which variables contain missing values.

```{r}

NAcol <- which(colSums(is.na(all)) > 0)
sort(colSums(sapply(all[NAcol], is.na)), decreasing = TRUE)
~~~~

~~~~
cat('There are', length(NAcol), 'columns with missing values')

```
~~~~

Missing Value Imputation is a statistical method that replaces missing data points with substituted values.

this step takes some time to impute.

~~~~
```{r, message=FALSE, results = FALSE,echo=TRUE,warning=FALSE}
# Install and load the R package mice
install.packages("mice")
library("mice")
# Impute missing data
imp <- mice(all, m = 1)
# Store imputed data as new data frame
final <- complete(imp)

#write out the final dataset as csv file for future use.
write.csv(final, "final.csv")

library(tidyverse)
sapply(final, function(x) sum(is.na(x)))

```
~~~~

last step to check duplicated in the final data set and reassign train and test data for modeling, as well as check the corrections.

~~~
```{r, message=FALSE, results = FALSE,echo=TRUE,warning=FALSE}

final<- read.csv("C:\\Users\\SSE6\\Downloads\\final.csv")

#Check duplicate in the final data set.
dup<-final[duplicated(final)==TRUE,]



#clearn data

final <-
  final %>%
  dplyr::select(-X) %>%
  mutate(x3= case_when(
 str_detect(x3,pattern = "Sun") ~ "Sunday",
 str_detect(x3,pattern = "Mon") ~ "Monday",
 str_detect(x3,pattern = "Tue") ~ "Tuesday",
 str_detect(x3,pattern = "Wed") ~ "Wednesday",
 str_detect(x3,pattern = "Thu") ~ "Thursday",
 str_detect(x3,pattern = "Fri") ~ "Friday",
 str_detect(x3,pattern = "Sat") ~ "Saturday"))
~~~

```
knitr::opts_chunk$set(echo = TRUE)

library(tidyverse)

library(knitr)

library(ggplot2)

library(plyr)

library(dplyr)

library(corrplot)

library(caret)

library(gridExtra)

library(scales)

library(randomForest)

library(psych)

library(xgboost)

#library(gtsummary)


# get the training data(missing value imputed)

data <- final %>% dplyr::filter(!y %in% c("ZERO"))


#split data 70% and 30% for training and testing data

train<- data %>%

  slice_sample(prop=0.7)


test  <- dplyr::anti_join(data, train)

test1<- test

# test_tbl data is for generate prediction purpose(missing value imputed)

test_tbl<- final %>% dplyr::filter(y %in% c("ZERO")) %>% dplyr::select(-y)
```

~~~
#check correlations of all numeric variables
all_numVar<- final %>% keep(is.numeric)

cor_numVar <- cor(all_numVar, use="pairwise.complete.obs")

```
~~~

Visualization to check the distribution of our target variable "y".

We can see the data is unbalanced,but data imbalanced problem are more likely to arise in the case of insurance data since the number of occurring claims is usually significantly lower than the number of non-occurring claims. So we don't need to use any balancing method.

~~~
```{r, message=FALSE, results = FALSE,echo=TRUE,warning=FALSE}
a<- train_tbl %>% ggplot(., aes(x = y)) + geom_bar(fill = 'lightblue')
ggsave("y.png", a)

```
~~~

To identify which features should be included in the models, I will use the random forest algorithm to identify the most important predictors.

~~~
```{r, message=FALSE, results = TRUE,echo=TRUE,warning=FALSE}

library(randomForest)
library(gmodels)
library(caret)
library(pROC)
~~~

```
# random forest "Area under curve (AUC) : 0.532"

train$y <- as.numeric(train$y)

rf<- randomForest(y ~ x7+x4+x47+x28+x40+x36+x18+x44+x16+x11+x8+x57+x89+x64+x90,
data=train,ntree=300,na.action = na.omit, importance=TRUE)


which.min(rf$mse)#find number of trees that produce lowest test MSE


test$predicted <- round(predict(rf,test),0)

print(sprintf("Area under curve (AUC) : %.3f",auc(test$y, test$predicted)))








# rf1<- randomForest(y ~ . , data=train,ntree=100,na.action = na.omit, importance=TRUE)

# test$predicted <- round(predict(rf1,test),0)

# print(sprintf("Area under curve (AUC) : %.3f",auc(test$y, test$predicted)))




test_tbl$predicted <- round(predict(rf,test_tbl),0)

write.csv( test_tbl$predicted,"nonglmresult.csv")
```

```
# Gradient Boosting Machine (GBM) model "Area under curve (AUC) : 0.506"

library(xgboost)
library(gbm)


train$x7<- as.factor( train$x7)


n<-names(train)
gbm.form <- as.formula(paste("y ~", paste(n[!n %in% "y"], collapse = " + ")))

gbmCV = gbm(formula = y~ x7+x4+x47+x28+x40+x36+x18+x44+x16+x11+x8+x57+x89+x64+x90,
          distribution = "bernoulli",
          data = train,
          n.trees = 500,
          shrinkage = .1,
          n.minobsinnode = 15,
          cv.folds = 5,
          n.cores = 1)


optimalTreeNumberPredictionCV = gbm.perf(gbmCV)
gbmTest = predict(object = gbmCV,
                newdata = test,
                n.trees = optimalTreeNumberPredictionCV,
                type = "response")
```

```
test$predicted <- round(gbmTest,0)
print(sprintf("Area under curve (AUC) : %.3f",auc(test$y, test$predicted)))
```

```
set.seed(123)

imp <- importance(rf, type=1)
featureImportance <- data.frame(Feature=row.names(imp), Importance=imp[,1])
featureImportance <- featureImportance %>%
  arrange(desc(Importance))%>%
  top_n(15)

p <- ggplot(featureImportance, aes(x=reorder(Feature, Importance), y=Importance)) +
    geom_bar(stat="identity", fill="#53cfff") +
    coord_flip() +
    theme_light(base_size=12) +
    xlab("") +
    ylab("Importance plot") +
    ggtitle("Random Forest Top15 Importance Features \n")+
    theme(plot.title=element_text(size=12))
```

~~~
ggsave("feature_importance.png", p)
print(p)
library(finalfit) #OR PLOT
library(dplyr)
library(ggplot2)

name<-featureImportance$Feature


var<- paste(shQuote(name,type = "cmd2"),    # Converting vector
            collapse = "+")


```


```{r}

#XGBoost model "Area under curve (AUC) : 0.502"


dMtrain <- xgb.DMatrix(as.matrix(train %>% keep(is.numeric)), label = train$y)
dMtest <-  xgb.DMatrix(as.matrix(test  %>% keep(is.numeric)), label = test$y)


params <- list(
  "objective"          = "binary:logistic",
~~~

```
  "eval_metric"        = "auc",
  "eta"                = 0.012,
  "subsample"          = 0.8,
  "max_depth"          = 8,
  "colsample_bytree"   =0.9,
  "min_child_weight"   = 5
)

nRounds <- 5000
earlyStoppingRound <- 100
printEveryN = 100
model_xgb.cv <- xgb.cv(params=params,
                  data = dMtrain,
                  maximize = TRUE,
                  nfold = 5,
                  nrounds = nRounds,
                  nthread = 3,
                  early_stopping_round=earlyStoppingRound,
                  print_every_n=printEveryN)

model_xgb <- xgboost(params=params,
                  data =  dMtrain ,
                  maximize = TRUE,
                  nrounds = nRounds,
                  nthread = 3,
                  early_stopping_round=earlyStoppingRound,
                  print_every_n=printEveryN)

colnames(dMtest) <- NULL
```

~~~~
test$predicted <- round(predict(model_xgb,dMtest),0)

testset_xgboost <- test$predicted

print(sprintf("Area under curve (AUC) : %.3f",auc(test$y, test$predicted)))


```


# Logistic regression

```{r}

set.seed(123)

library(pROC)




glm.model<- glm(as.factor(y) ~
x7+x47+x4+x36+x16+x28+x18+x11+x40+x89+x57+x64+x8+x44+x90,data=train,family = 'binomial')

prob <- predict(glm.model,newdata=test1,type = "response")

predict = rep(0, length(prob))

predict[prob > 0.5] <- 1

table(predict,test1$y)

mean(predict==test1$y)

print(sprintf("Area under curve (AUC) : %.3f",auc(test1$y, predict)))

# Multicollinearity


# One of the assumptions of logistic regression is that the predictors are not too highly correlated with
each other. The Variance Inflation Factor (VIF) measures the amount of multicollinearity between the
features in the model. A general rule of thumb is a VIF score of no higher than between 5 and 10. Since
the majority of the predictors have a VIF of higher than 5, we dont need to drop variables among the
model.


suppressMessages(library(regclass))

VIF(glm.model)
~~~~

```
glmresult <- predict(glm.model,newdata=test,type = "response")

write.csv(predict,"glmresult.csv")
```

```{r}

library(tidymodels)
library(discrim)
library(baguette)
library(bonsai)
train$y<- as.numeric(train$y)
train_n<- train %>% keep(is.numeric)
train_n$y <- as.factor(train_n$y)

model_recipe <-
  recipe(y~ . , data = train_n) %>%
  step_corr(all_numeric_predictors(), threshold = .5) %>%
  step_nzv(all_predictors()) %>%
  step_YeoJohnson(all_predictors()) %>%
  step_zv(all_predictors())
```

```
cv <- vfold_cv(train_n, v = 10)

 # Support Vector Machines
svm_model <-
  svm_rbf(mode = "classification",
          cost = tune(),
          rbf_sigma = tune(),
          engine = "kernlab"
  )

svm_wf <-
  workflow() %>%
  add_model(svm_model) %>%
  add_recipe(model_recipe)
svm_wf


# Hyperparameter Tuning
svm_results <-
  svm_wf %>%
  tune_grid(resamples = cv,
            metrics = metric_set(accuracy)
  )

svm_results %>%
  collect_metrics()


# Final Hyperparameter
param_final <- svm_results %>%
  select_best(metric = "accuracy")

svm_wf <- svm_wf %>%
  finalize_workflow(param_final)
svm_wf

# Last fit
svm_fit <- svm_wf %>%
  last_fit(diabetes_split)
# Test Data Prediction
test_performance <- svm_fit %>% collect_predictions()
test_performance

# Performance Metrics
diabetes_metrics <- metric_set(accuracy, f_meas, precision, recall)
diabetes_metrics(data = test_performance, truth = y, estimate = .pred_class)
```