DATA SCIENCE FINAL PROJECT

Jobert Jay R. Pandan

2022-12-16

MODEL 1:RANDOM FOREST MODEL

Libraries Needed for Pre-processing

```
library(readr)
library(dplyr)
library(ggplot2) # for plotting
library(caret) # pre-processing and modeling
library(corrplot)
library(fastDummies) # for creating dummy variables
pacman::p_load(tidyverse)
pacman::p_load(bestNormalize)

##Modelling package

library(ranger)   # a c++ implementation of random forest
library(h2o)      # a java-based implementation of random forest
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 5 minutes
##     H2O cluster timezone:       +08:00
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    2 months and 28 days
##     H2O cluster name:           H2O_started_from_R_Jamila_vhm229
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   0.87 GB
##     H2O cluster total cores:    2
##     H2O cluster allowed cores:  2
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.3 (2022-03-10)
```

DATA ACQUISITION AND PRE-PROCESSING

Dataset

radiomics_completedata.csv data was used for this project. The data has a 431 variables and 197 observations

including *Failure.binary* as our target/response/outcome variable with its 430 predictors/features/independent variables. *Failure.binary* is a binary variable with 1 and 0 as its values.

```
radiomics = read.csv("radiomics_completedata.csv")
newdf = dummy_cols(radiomics, select_columns = "Institution" )
newdf = newdf[,-1]
newdf$Institution_A = as.factor(newdf$Institution_A)
newdf$Institution_B = as.factor(newdf$Institution_B)
newdf$Institution_C = as.factor(newdf$Institution_C)
newdf$Institution_D = as.factor(newdf$Institution_D)
newdf$Failure.binary = as.factor(newdf$Failure.binary)
str(newdf[1])
```

```
## 'data.frame':    197 obs. of  1 variable:
##  $ Failure.binary: Factor w/ 2 levels "0","1": 1 2 1 2 1 2 1 1 2 2 ...
```

```
newdf1 = newdf %>% select_if(is.numeric)
tempDF=apply(newdf1,2,orderNorm)
tempDF=lapply(tempDF, function(x) x$x.t)
tempDF=tempDF%>%as.data.frame()
norm_data = cbind(newdf[c('Failure.binary','Institution_A','Institution_B','Institution_C','Institution
```

```
set.seed(3333)
trainIndex <- createDataPartition(norm_data$Failure.binary, p = .80,
                                  list = FALSE,
                                  times = 1)
finaldata_train<- norm_data[ trainIndex,]
finaldata_test<- norm_data[-trainIndex,]
```

```
# train a default random forest model
n_features <- length(setdiff(names(finaldata_train), "Failure.binary"))
rf_mod1 <- ranger(
  Failure.binary ~ .,
  data = finaldata_train,
  mtry = floor(n_features / 3),
  respect.unordered.factors = "order",
  seed = 123
)
(default_rmse <- sqrt(rf_mod1$prediction.error))
```

```
## [1] 0.3375264
```

This model uses the basic functions of modeling using ranger() in training the model. This model has an RMSE of 0.3375264 which will be our baseline model

```
hyper_grid <- expand.grid(
  mtry = floor(n_features * c(.05, .15, .25, .333, .4)),
  min.node.size = c(1, 3, 5, 10),
  replace = c(TRUE, FALSE),
  sample.fraction = c(.5, .63, .8),
  rmse = NA
)

# execute full cartesian grid search
for(i in seq_len(nrow(hyper_grid))) {
  # fit model for ith hyperparameter combination
  rf_fit <- ranger(
```

```
    formula         = Failure.binary ~ .,
    data            = finaldata_train,
    num.trees       = n_features * 10,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$min.node.size[i],
    replace         = hyper_grid$replace[i],
    sample.fraction = hyper_grid$sample.fraction[i],
    verbose         = FALSE,
    seed            = 123,
    respect.unordered.factors = 'order',
  )
  # export OOB error
  hyper_grid$rmse[i] <- sqrt(rf_fit$prediction.error)
}
# assess top 10 models
hyper_grid %>%
  arrange(rmse) %>%
  mutate(perc_gain = (default_rmse - rmse) / default_rmse * 100) %>%
  head(10)
```

```
##    mtry min.node.size replace sample.fraction      rmse perc_gain
## 1   173             1   FALSE             0.5 0.3182229  5.719096
## 2   173             5   FALSE             0.5 0.3182229  5.719096
## 3   173            10   FALSE             0.5 0.3182229  5.719096
## 4   173             1    TRUE             0.5 0.3280167  2.817468
## 5   173             3    TRUE             0.5 0.3280167  2.817468
## 6   173             5    TRUE             0.5 0.3280167  2.817468
## 7   173            10    TRUE             0.5 0.3280167  2.817468
## 8   173             3   FALSE             0.5 0.3280167  2.817468
## 9   108             5   FALSE             0.5 0.3280167  2.817468
## 10  108            10   FALSE             0.5 0.3280167  2.817468
```

This model rf_fit uses ranger() with a hyperparameter grid shows the top 10 good-performing models with RMSE below 0.32900. Five of those models performed better than the baseline model with an RMSE of 0.3182229 and a model percentage gain of 5.7%.

```
h2o_datatraining <- as.h2o(finaldata_train)
```

```
##   |                                                                      |
```

```
# set the response column to Failure.binary
response <- "Failure.binary"

# set the predictor names
predictors <- setdiff(colnames(finaldata_train), response)

h2o_rf1 <- h2o.randomForest(
  x = predictors,
  y = response,
  training_frame = h2o_datatraining,
  ntrees = n_features * 10,
  seed = 123
)
```

```
##   |                                                                      |
```

```
h2o_rf1
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: drf
## Model ID:  DRF_model_R_1671282356539_94
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1            4330                     4330             1187345         4
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        15    6.97136          7         28    17.03811
##
##
## H2OBinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## MSE:  0.1386716
## RMSE:  0.3723863
## LogLoss:  0.4353124
## Mean Per-Class Error:  0.2069088
## AUC:  0.8630698
## AUCPR:  0.804894
## Gini:  0.7261396
## R^2:  0.3835832
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error     Rate
## 0      86 18 0.173077  =18/104
## 1      13 41 0.240741   =13/54
## Totals 99 59 0.196203  =31/158
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold      value idx
## 1                       max f1  0.361580   0.725664  58
## 2                       max f2  0.177437   0.810398 110
## 3                 max f0point5  0.589059   0.758427  30
## 4                 max accuracy  0.511017   0.810127  39
## 5                max precision  0.920654   1.000000   0
## 6                   max recall  0.084427   1.000000 141
## 7              max specificity  0.920654   1.000000   0
## 8             max absolute_mcc  0.361580   0.574781  58
## 9   max min_per_class_accuracy  0.342398   0.777778  61
## 10 max mean_per_class_accuracy  0.361580   0.793091  58
## 11                     max tns  0.920654 104.000000   0
## 12                     max fns  0.920654  53.000000   0
## 13                     max fps  0.014681 104.000000 157
## 14                     max tps  0.084427  54.000000 141
## 15                     max tnr  0.920654   1.000000   0
## 16                     max fnr  0.920654   0.981481   0
## 17                     max fpr  0.014681   1.000000 157
## 18                     max tpr  0.084427   1.000000 141
##
```

```
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

h2o_rf1 model uses h2o() in training the model with an RMSE of 0.3723863 which means this model doen't provide gain percentage from the baseline model Thus, the model rf_fit performed better than this model.

Therefore, rf_fit model is our final model. Testing the model performance we have,

```
predictions = predict(rf_fit, data = finaldata_test)
confusionMatrix(data = finaldata_test$Failure.binary, predictions$predictions )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 25  1
##          1  3 10
##
##                Accuracy : 0.8974
##                  95% CI : (0.7578, 0.9713)
##     No Information Rate : 0.7179
##     P-Value [Acc > NIR] : 0.006455
##
##                   Kappa : 0.76
##
##  Mcnemar's Test P-Value : 0.617075
##
##             Sensitivity : 0.8929
##             Specificity : 0.9091
##          Pos Pred Value : 0.9615
##          Neg Pred Value : 0.7692
##              Prevalence : 0.7179
##          Detection Rate : 0.6410
##    Detection Prevalence : 0.6667
##       Balanced Accuracy : 0.9010
##
##        'Positive' Class : 0
##
```

The rf_fit model has an accuracy of 89.74%, looking at the confusion matrix, the model predicted `Failure.binary` 0 corrrectly with only 1 observation that is misclassified and 10 observation are correctly classified in `failure.binary` 1 with only 3 misclassified observation.

It is important to determine the variables that are most influential in predicting accuracy of the model. Based on the figure below, we can say that:

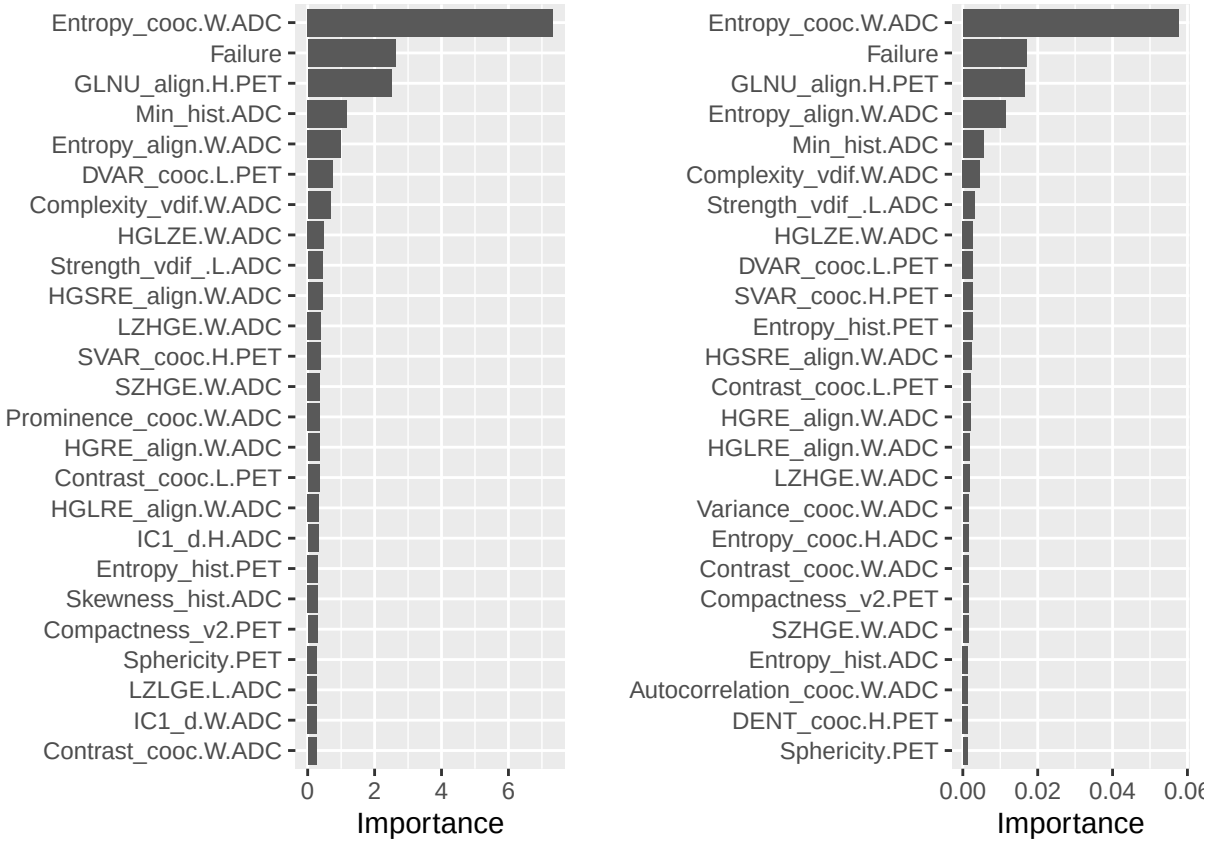
```
rf_impurity <- ranger(
  formula = Failure.binary ~ .,
  data = finaldata_train,
  num.trees = 2000,
  mtry = 32,
  min.node.size = 1,
  sample.fraction = .80,
  replace = FALSE,
  importance = "impurity",
  respect.unordered.factors = "order",
  verbose = FALSE,
  seed  = 123
```

```
)
```

```
rf_permutation <- ranger(
  formula = Failure.binary ~ .,
  data = finaldata_train,
  num.trees = 2000,
  mtry = 32,
  min.node.size = 1,
  sample.fraction = .80,
  replace = FALSE,
  importance = "permutation",
  respect.unordered.factors = "order",
  verbose = FALSE,
  seed  = 123
)
```

```
p1 <- vip::vip(rf_impurity, num_features = 25, bar = FALSE)
p2 <- vip::vip(rf_permutation, num_features = 25, bar = FALSE)

gridExtra::grid.arrange(p1, p2, nrow = 1)
```



Entropy_cooc.W.ADC, Failure and GLNU_align.H.PET are the variables that helps the model in predicting the classification correctly.