Pluralsight Model Building Take-Home

William Rinauto

2022-03-23

library(tidyverse)  
library(h2o)  
library(vtable)  
library(scales)  
library(ggcorrplot)  
library(kableExtra)  
library(rpart)  
library(rpart.plot)

training\_dat <- read.csv('recruiting\_zeta-disease\_training-data\_take-home-challenge - 2021\_zeta-disease\_training-data\_take-home-challenge.csv')  
predict\_these <- read.csv('recruiting\_zeta-disease\_prediction-data\_take-home-challenge - 2021-01-21\_zeta-disease\_prediction-data\_take-home-challenge.csv')  
  
#Check for duplicate rows.  
#Even though there is no person identifier in this data, I am going to assume that if   
#data values are the same for each data field in two or more rows, then that is duplicate data  
#(meaning the same person has been included twice in the data).   
#I think this is a fair assumption because it is highly improbable that two individuals would match   
#across all of these features.   
  
#How many rows will be dropped:   
nrow(training\_dat) - nrow(distinct(training\_dat))

[1] 5

#Check to make sure that a person hasn't been recorded twice with two different values for zeta\_disease  
#If this matches return value of above line of code, then I'll know they haven't  
  
nrow(training\_dat %>% select(-zeta\_disease)) - nrow(distinct(training\_dat %>% select(-zeta\_disease)))

[1] 5

#Drop the 5 duplicate rows:  
training\_dat <- distinct(training\_dat)  
  
#What proportion of training data is zeta positive?   
mean(training\_dat$zeta\_disease)

[1] 0.3496855

#Define function to count NA values in each column of some dataframe  
na\_count <- function(df) sapply(df, function(c) sum(is.na(c)))  
  
#A return value of true here indicates that there is no missing data in training data  
all(na\_count(training\_dat) == 0)

[1] TRUE

#A return value of true here indicates that there is no missing data in testing data  
all(na\_count(predict\_these) == 0)

[1] FALSE

#Returns False so I will investigate further:  
# na\_count(predict\_these)  
  
#In this case, I can see that the only "missing" data is zeta\_disease, which makes sense because it hasn't been predicted yet  
  
  
#Summary statistics table to get a quick sense of data distributions and also sanity check   
#for outliers in mins and maximums (negative weight or age for example would be an indicator of bad data)  
# sumtable(training\_dat)  
  
#Also want to see summary table of the testing data  
# sumtable(predict\_these)  
  
#A few things stick out in the summary table for the training data.   
#There are 5 fields (ignoring zeta\_disease) that have minimum values of 0.   
#Two of these stick out to me, because I would think that there is something problematic about having a 0 value here:  
#bmi and blood\_pressure  
  
#Let's investigate further  
  
  
#How often does 0 occur by data field?   
  
#Define function to count 0's.   
zero\_count <- function(df) sapply(df, function(c) percent(mean(c == 0)))  
# zero\_count(training\_dat)  
  
#I can see that 1% of the data has bmi = 0, and 4% of the data has blood\_pressure = 0  
#(Not as important, but we should probably understand why only 14% of people in this dataset don't smoke. Is this a non-representative sample, or is something going on on mars  
#that makes people more inclined to take up smoking?)  
  
#At this point, I am going to make two assumptions moving forward with this project:  
  
#ASSUMPTION 1: A person's BMI cannot be 0. According to the data dictionary, bmi = weight/height. Since weight definitely can't be 0  
#(and a weight of 0 would have showed up in the data, assuming that it is the same instance of weight measured here as was used to calculate  
#bmi), bmi can't be 0.  
  
#ASSUMPTION 2: A person's blood\_pressure CANNOT be 0. I believe this means that the heart has stopped. I'm assuming no dead people had data collected here  
  
#Assumptions 1 and 2 require me to take action to clean the data:  
  
#Some possible options for each field:   
#1. Mean imputation  
#2. Group mean imputation  
#3. KNN imputation  
#4. Delete bad data  
  
#To keep things simple, I will use mean imputation and replace all 0 values for blood\_pressure and bmi with the mean of all non-zero values for each  
non\_zer\_bp <- training\_dat[training\_dat$blood\_pressure > 0,]  
avg\_bp <- mean(non\_zer\_bp$blood\_pressure)  
  
non\_zer\_bmi <- training\_dat[training\_dat$bmi > 0,]  
avg\_bmi <- mean(non\_zer\_bmi$bmi)  
  
replacements\_vals <- tibble(blood\_pressure = avg\_bp,  
 bmi = avg\_bmi)  
  
#This will be used in "production" environment to clean data before  
#creating predictions  
write.csv(replacements\_vals, 'replacement\_vals.csv',row.names =F)  
  
training\_dat <- training\_dat %>%  
 #Replace values of 0 with non-zero means  
 mutate(blood\_pressure = ifelse(blood\_pressure == 0, avg\_bp, blood\_pressure),  
 bmi = ifelse(bmi == 0, avg\_bmi, bmi))  
  
  
  
#see that min blood\_pressure and bmi are no longer 0  
# min(training\_dat$blood\_pressure)  
min(training\_dat$bmi)

[1] 18.2

#One more sanity check: is years smoking ever greater than age? If it is, that's another indicator of bad data  
#In this case, I will use greater than/equal to in my logical statement rather than strict inequality,  
#therefore implying that it is not irrational for a person to have begun smoking  
#on mars at 0 years-old. I will not presume to understand mars culture.  
  
#A return value of false here indicates a logical contradiction in the data  
all(training\_dat$age >= training\_dat$years\_smoking)

[1] FALSE

#Returns false so I will investigate further:  
smoker\_anomaly\_training <- filter(training\_dat, years\_smoking > age)  
# head(smoker\_anomaly\_training)  
  
#In this case, we can see that two individuals labeled as 19 years old have supposedly been smoking for longer than they've been alive  
#At this point, I have to clean up the data. I can do so by selecting from the four options listed above.   
  
#This situation is a little different than the blood pressure and bmi situations, though, in that I am finding a contradiction in the data by comparing two  
#data fields to each other, rather than making a logical assumption about a single data field in isolation. The implication is that I don't know   
#which of these two data fields is the incorrect one for these two observations, age or years\_smoking. Therefore, a decision to replace bad data values by imputation would  
#be arbitrary: I have no way of saying that age must be replaced and years\_smoking kept, or vice versa. Therefore, I will delete these rows of data entirely.   
#Since this is only two rows of data that account for 0.25% of total observations in this dataset, it should be fairly inconsequential to remove them.  
  
#Delete contradictory data:  
training\_dat <- filter(training\_dat, age >= years\_smoking)  
  
  
#Summary table after cleaning up data:  
# sumtable(training\_dat)  
  
  
#Now that I am done cleaning the training data, I am going to   
#search for relationships in the data  
  
cor\_dat <- training\_dat  
names(cor\_dat) <- knitr:::escape\_latex(names(cor\_dat))  
  
cors <- cor(cor\_dat)  
cors <- cors[,order(cors['zeta\\\_disease',], decreasing = F)]  
cor\_plot <- ggcorrplot(cors,  
 # colors = c(min(cors), 0, max(cors[cors!=1])),  
 type = 'lower',  
 lab = T) +  
 scale\_fill\_gradient2(limit = c(min(cors),max(cors[cors!=1])), low = "blue", high = "forestgreen", mid = "white", midpoint = 0) +  
 ggtitle('Training Data Correlations\n(Sorted desc on zeta\_disease correlation)')  
  
# print(cor\_plot)  
  
#Based on correlations alone, weight immediately sticks out as potentially predictive. Cardio stress test seems like it will be  
#the least predictive  
  
  
#Another angle I'd like to see on the data is average of each candidate predictor based on whether or not the person is  
#zeta positive  
  
#This shows an average profile of a zeta positive individual compared to a non zeta positive individual  
  
zeta\_means <- training\_dat %>%  
 group\_by(zeta\_disease) %>%  
 summarise\_all(mean) %>%   
 left\_join(training\_dat %>% count(zeta\_disease,name = 'Count')) %>%  
 arrange(desc(zeta\_disease))  
  
  
#Chop off some decimal places  
rnd <- function(x) round(x,2)  
  
#mutate\_all is a quick way to apply a function to every column in a dataframe  
zeta\_means <- mutate\_all(zeta\_means, rnd)  
  
kable(zeta\_means) %>%  
 kable\_styling(bootstrap\_options = "bordered",  
 full\_width = FALSE)

zeta\_disease

age

weight

bmi

blood\_pressure

insulin\_test

liver\_stress\_test

cardio\_stress\_test

years\_smoking

Count

1

34.40

192.92

35.62

74.98

107.56

0.63

44.57

5.14

278

0

28.65

161.35

31.14

71.59

73.85

0.50

42.26

3.37

515

#Want to also see how various candidate predictors are distributed  
  
#Create function that takes dataframe and column name as input, and outputs density plot  
plot\_dense <- function(dat, col) {  
   
 ggplot(dat, aes\_string(x = col)) +   
 geom\_density(lwd = .8, fill = 'blue',alpha = .2) +  
 ylab('Density') +  
 ggtitle(c) +  
 xlab(paste0(c, "\n\n\n"))  
   
   
}  
  
cols <- names(training\_dat)[!names(training\_dat) == 'zeta\_disease']  
  
for(c in cols) {  
   
 p <- plot\_dense(training\_dat, c)  
 # print(p)  
}  
  
#Also interested in seeing each variable plotted against zeta\_disease in a scatterplot  
#I will add a smoothing line to give some indicacator of the relationship between each  
#variable and zeta\_disease. It's important to note that outliers can have a major impact  
#on the visual interpretation of the smoothing line, so a wider "shadow" around the line essentially   
#indicates less trustworthiness in the shape of the line within that region of data.  
  
#Create scatterplot function  
plot\_scatter <- function(dat1, dat2, col) {  
 # browser()  
 dat1 <- dat1 %>%  
 mutate(zeta\_disease1 = as.numeric(zeta\_disease))   
   
 ggplot(dat1, aes\_string(x = col, y = 'zeta\_disease', color = 'zeta\_disease')) +  
 stat\_smooth(method="glm", color="black", se=T,  
 method.args = list(family=binomial)) + geom\_point() +  
 # geom\_vline(data = dat2, aes\_string(xintercept = col, color = 'zeta\_disease'), lwd = 1) +  
 xlab(paste0(c, "\n\n\n")) +  
 ggtitle(c) +  
 scale\_y\_continuous(breaks = c(0,1)) +  
 theme(legend.position = 'none') +  
 ylab('Zeta Status')  
   
   
   
}  
  
  
  
for(c in cols) {  
   
 p <- plot\_scatter(training\_dat, zeta\_means, c)  
 # print(p)  
}  
  
#Tree------  
  
  
#Another thing I want to see to get a very simple idea of how these features influence the outcome and interact with  
#one another is a decision tree  
  
#Indicate in training data that zeta\_disease is categorical  
training\_dat <- mutate(training\_dat, zeta\_disease = as.factor(zeta\_disease))  
  
#I am going to shorten the variable names just so the plot prints tidier  
tree\_dat <- training\_dat  
  
shorten <- function(x) substr(x, 1,5)  
names(tree\_dat) <- sapply(names(tree\_dat), shorten)  
  
simple\_tree <- rpart(zeta\_ ~ ., data = tree\_dat,  
 control = rpart.control(maxdepth = 4))  
  
#Forcing the tree to make only 4 splits at most indicates that weight,   
#BMI, and age will likely be important in a final model predicting zeta\_disease  
  
# rpart.plot(simple\_tree, cex = 1, extra = 2)  
  
  
#What I'm seeing in the tree corroborates some of what I was seeing in the correlation table and plot:  
#weight as the root node and the field with the highest importance measure is consistent with it having  
#the highest correlation with the target variable. cardio\_stress\_test seems to be inconsequential based on both  
#correlation with the target as well as importance in the tree

#Model----  
  
#Now that I have cleaned the data and explored some of the relationships, I will   
#build a few different classification models and select the one that is most successful  
  
  
#The first thing I'm going to do is drop cardio\_stress\_test from the training data because it seems to be unimportant based   
#on the preliminary analysis  
  
training\_dat <- select(training\_dat, -cardio\_stress\_test)  
  
#I also want to add a few features to the data. I could have  
#added these in the data exploration step, but I wanted to avoid cluttering  
#the rmarkdown document too much  
  
# square <- function(x) x^2  
#   
# feature\_funs <- list(sqrt = sqrt, square = square)  
  
  
#mutate\_at to target every variable that is not zeta\_disease  
#training\_dat <- mutate\_at(training\_dat, vars(!matches('zeta\_disease')), feature\_funs)  
  
  
#Actually, after creating features this way and running the models, I am seeing that they add nothing  
#To the accuracy of any of the models, so I will comment out the prior couple lines of code  
  
#Initialize h2o  
h2o.init(nthreads = -1)

H2O is not running yet, starting it now…

Note: In case of errors look at the following log files: C:4IXrPl3f60375a2e83/h2o\_willi\_started\_from\_r.out C:4IXrPl3f6053bb25ce/h2o\_willi\_started\_from\_r.err

Starting H2O JVM and connecting: Connection successful!

R is connected to the H2O cluster: H2O cluster uptime: 1 seconds 853 milliseconds H2O cluster timezone: America/New\_York H2O data parsing timezone: UTC H2O cluster version: 3.36.0.3 H2O cluster version age: 1 month and 6 days  
H2O cluster name: H2O\_started\_from\_R\_willi\_uud350 H2O cluster total nodes: 1 H2O cluster total memory: 15.93 GB H2O cluster total cores: 16 H2O cluster allowed cores: 16 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)

#Hide progrses bar because it looks bad in rendered document  
h2o.no\_progress()  
  
#Convert target variable to categorical to avoid accidentally predicting continuous outcome:  
training\_dat <- mutate(training\_dat, zeta\_disease = as.factor(zeta\_disease))  
  
#Convert training data to h2o object:  
training\_dat\_h2o <- as.h2o(training\_dat)  
  
  
#this line looks pointless but I had deleted a few lines of code   
#where previously it made sense to have this here. Now I'm keeping it to avoid  
#changing code below  
train\_h2o <- training\_dat\_h2o  
  
  
#Want to run predictions on final test data as I go along  
testing\_dat\_h2o <- predict\_these %>%  
 select(-zeta\_disease) %>%  
 as.h2o()  
  
#character vector of candidate regressors:  
candidate\_regressors <- names(training\_dat)[names(training\_dat) != 'zeta\_disease']  
  
#target\_variable  
target <- 'zeta\_disease'  
  
#See all data types to make sure nothing accidentally ended up as categorical somehow (zeta\_disease should show as factor here)  
# sapply(training\_dat, class)  
  
  
#For each model, I will use a grid search method to tune hyperparameters  
  
#First test GLM  
  
#It's probably overkill to use elastic net regression but   
#if alpha = lamda = 0 is optimal, the grid search will indicate this  
hyper\_params\_glm <- list(  
 #Controls distribution between ridge and lasso componenets of penalty  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/alpha.html  
 alpha = seq(from = 0, to = 1, by = 0.001),  
   
 #Amount of regularization (large value here means coefficients shrink closer to 0  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/lambda.html  
 lambda = c(.00001, .0001, .001, .01, .1, .5, 1)  
)  
  
#Number of models to be tested for in absence of RandomDiscrete strategy:  
sapply(hyper\_params\_glm, length) %>% prod()

[1] 7007

#Controls how grid search is run (will use same search\_criteria for every model)  
search\_criteria <- list(  
 #RandomDiscrete grid search samples from parameter space  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/grid-search.html  
 strategy = "RandomDiscrete",  
 max\_runtime\_secs = 30,  
 max\_models = 200,  
 stopping\_metric = "AUC",   
 stopping\_tolerance = 0.00001,   
 stopping\_rounds = 5,   
 seed = 123  
)  
  
glm\_models <- h2o.grid(algorithm = "glm",  
 grid\_id = "regression",  
 x = candidate\_regressors,   
 y = target,   
 training\_frame = train\_h2o,  
 nfolds = 10,   
 family = "binomial",   
 hyper\_params = hyper\_params\_glm,   
 search\_criteria = search\_criteria,  
 seed = 123)  
  
#Since h2o.predict will use f1 under the hood to determine classification   
#threshold, I will select the model with the highest cv f1 score  
glm\_sorted <- h2o.getGrid(grid\_id = "regression", sort\_by = "f1", decreasing = TRUE)  
  
#Top model when sorted descending on f1  
glm\_best <- h2o.getModel(glm\_sorted@model\_ids[[1]])  
  
  
#Useful stackoverflow thread discussing h2o.performance object:  
# https://stackoverflow.com/questions/43699454/how-to-understand-the-metrics-of-h2omodelmetrics-object-through-h2o-performance  
  
#xval = T below means I am pulling performance data based on cross validation  
#testing datasets, not training data. Will do this throughout  
glm\_performance <- h2o.performance(glm\_best, xval = T)  
glm\_f1 <- h2o.F1(glm\_performance) %>%  
 as.data.frame() %>%  
 arrange(desc(f1))   
  
#Store f1 value of best performing glm model  
glm\_f1 <- glm\_f1[1,2]  
  
#Appy predictions to test data. Will take a look at this later  
final\_predictions\_glm <- h2o.predict(glm\_best, testing\_dat\_h2o) %>%  
 as.data.frame()  
h2o.varimp(glm\_best)

Variable Importances: variable relative\_importance scaled\_importance percentage 1 weight 1.095977 1.000000 0.403601 2 bmi 0.565687 0.516149 0.208318 3 years\_smoking 0.355938 0.324768 0.131077 4 liver\_stress\_test 0.310933 0.283704 0.114503 5 age 0.182344 0.166376 0.067149 6 insulin\_test 0.139783 0.127542 0.051476 7 blood\_pressure 0.064837 0.059159 0.023876

#Next try a random forest. This tree based model will be better if there are   
#interactions among variables  
  
hyper\_params\_forest <- list(  
 #number of trees  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/ntrees.html  
 ntrees = 10000,   
   
 #how deep tree can go:  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/max\_depth.html  
 max\_depth = 12:25,  
   
 #How much data must be in each bucket to make a split:  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/min\_rows.html  
 min\_rows = seq(1,101, 5),  
   
 #Row sampling rate:  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/sample\_rate.html  
 sample\_rate = seq(.1, 1, by = .1),  
   
 #Number of columns to sample at each node  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/mtries.html  
 mtries = c(-1,1:7)  
)  
  
  
forest\_models <- h2o.grid(algorithm = "randomForest",   
 grid\_id = "forest",   
 x = candidate\_regressors,   
 y = target,   
 training\_frame = train\_h2o,   
 nfolds = 10,   
 hyper\_params = hyper\_params\_forest,   
 search\_criteria = search\_criteria,   
 seed = 123)  
  
forest\_sorted <- h2o.getGrid(grid\_id = "forest", sort\_by = "f1", decreasing = TRUE)  
  
#Grab top performing model based on f1  
forest\_best <- h2o.getModel(forest\_sorted@model\_ids[[1]])  
  
forest\_perf <- h2o.performance(forest\_best, xval = T)  
forest\_f1 <- h2o.F1(forest\_perf) %>%  
 as.data.frame() %>%  
 arrange(desc(f1))   
  
#grab f1 statistic  
forest\_f1 <- forest\_f1[1,2]  
  
#Apply predictions to test data  
final\_predictions\_forest <- h2o.predict(forest\_best, testing\_dat\_h2o) %>%  
 as.data.frame()  
  
  
#Last I will try gradient boosting:  
hyper\_params\_gbm <- list(ntrees = 10000,   
 max\_depth = 5:15,   
 min\_rows = c(15, 20,30, 50,100),  
   
 #GBM learn rate (how much to adjust predicted residuals based on new tree)  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/learn\_rate.html  
 learn\_rate = c(0.001,0.01,0.1, .3, .5),   
   
 #Change in learn rate each round  
 #https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/algo-params/learn\_rate\_annealing.html  
 learn\_rate\_annealing = c(0.99,0.999,1),  
 sample\_rate = seq(.2, 1, by = 1),  
 col\_sample\_rate = seq(.1, 1, by = 1)  
   
)  
  
  
models\_gbm <- h2o.grid(algorithm = "gbm", grid\_id = "gbm",  
 x = candidate\_regressors,   
 y = target,  
 training\_frame = train\_h2o,   
 nfolds = 10,   
 hyper\_params = hyper\_params\_gbm,   
 search\_criteria = search\_criteria,   
 seed = 123)  
  
gbm\_sorted <- h2o.getGrid(grid\_id = "gbm", sort\_by = "f1", decreasing = TRUE)  
  
#Best gbm model based on f1 stat  
gbm\_best <- h2o.getModel(gbm\_sorted@model\_ids[[1]])  
  
gbm\_perf <- h2o.performance(gbm\_best, xval = T)  
gbm\_f1 <- h2o.F1(gbm\_perf) %>%  
 as.data.frame() %>%  
 arrange(desc(f1))   
  
#grab best gbm f1 statistic  
gbm\_f1 <- gbm\_f1[1,2]  
  
#Final predictions.. will take a look later  
final\_predictions\_gbm <- h2o.predict(gbm\_best, testing\_dat\_h2o) %>%  
 as.data.frame()  
  
models\_perf\_metric <- list(glm = glm\_f1, rf = forest\_f1, gbm = gbm\_f1)  
#f1 score of each model:  
models\_perf\_metric

$glm [1] 0.6872111

$rf [1] 0.695935

$gbm [1] 0.6992366

#See how each model predicts on test data  
dat <- bind\_cols(list(final\_predictions\_glm, final\_predictions\_forest, final\_predictions\_gbm)) %>%  
 select(contains('predict'))  
names(dat) <- c('glm','forest','gbm')  
# head(dat, 20)  
  
#proportion of zeta positive predictions by model on test data:  
convert\_fct\_numeric <- function(x) mean(as.numeric(as.character(x)))  
sapply(dat, convert\_fct\_numeric)

glm forest gbm 0.8 0.7 0.7

#Based on f1, random forest just barely squeaks out as the winner,  
#so I will go with that  
  
#Save model to be ingested by python   
h2o.saveModel(forest\_best, getwd(), filename = 'model', force = T)

[1] “C:\Users\willi\Desktop\Pluralsight Test\model”

#Take a look at confusion matrix. By default,  
#this will be based on training data  
  
#Note that h2o switches 0 and 1 from their conventional positions   
#(0/1 instead of 1/0)  
h2o.confusionMatrix(forest\_best)

Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.3814088300181: 0 1 Error Rate 0 392 123 0.238835 =123/515 1 62 216 0.223022 =62/278 Totals 454 339 0.233291 =185/793

#ROC Curve for GBM model based on cross validation   
plot(h2o.performance(forest\_best, xval = T) ,type='roc')

