Using XGBoost to predict Successful Drug Repurposing Candidates

James Young

3/14/2020

## The Problem

A recent paper tried to make a model that could predict succesful candidates for drug repurposing <https://www.nature.com/articles/s41598-019-42806-6#data-availability>. They incorporated new predictor variables by scraping the literature for semantic relations between drugs, diseases, etc. The data is available here <https://github.com/Wytz/Drug_repurposing>. I wanted to see if I could achieve better results than their .922 AUC (a metric of model performance in unbalanced classification data). They used a random forest and I will use the newer xgboost framework.

## The Value

If a company can more accurately predict which drugs have a better chance of being successfully repurposed, they can save money and time. For individuals needing new drugs, this would hopefully mean they could get drugs sooner and more affordably as well.

## Read The Data In

First I will pull the data into R and change all of the categorical data to one-hot encoded dummy variables. We can see we have ~8,000 drugs and ~1,835 columns of data. The one hot encoded categorical variables account for many of these columns, which can be reffered to as sparse data. Modern machine learning algorithms such as xgboost can handle this sparse data better than classic approaches such as linear regression. The xgboost model will limit the effect of less important variables by using them in less trees of the underlying ensemble. The xgboost effectively does feature selection while building the model.

setwd("C://data/")  
x <- read.csv("DrugRepurposing.csv", stringsAsFactors = T)  
x2 <- predict(dummyVars(~ disease\_id, data = x), newdata = x)  
x <- as.data.frame(cbind(x2, x))  
cat("Rows of Data")

## Rows of Data

nrow(x)

## [1] 8065

cat("Columns of Data")

## Columns of Data

ncol(x)

## [1] 1835

## Split The Data 90/10

I have split the data into train/test groups to see the generalization of the model using a 90/10 split.With approximately 8,000 samples to split, this means we will validate on about 800 samples. This is the same split as the cross validation used in the paper. Later on I Will proved a cross validation as well.

set.seed(1234)  
#We will get rid of the two character based features below before splitting the data. Character based features do not work in xgboost. There are ways to try to retain the information of the features for xgboost but that is for another day.  
x$disease\_id <- NULL  
x$drug\_id <- NULL  
sample <- sample.int(n = nrow(x), size = floor(.90\*nrow(x)), replace = F)  
train <- x[sample, ]  
test <- x[-sample, ]

## Preparing The Data

Here I will prepare the data into a matrix of predictor variables and a vector of labels. The labels are binary (1 and 0) with representing a successfully repurposed drug and 0 representing failure.

labels <- as.numeric(train$Status)  
ts\_label <- as.numeric(test$Status)  
  
#Below, by limiting our predictor variable matrix to 1:1832, we make sure we don't leave the response variable in.  
  
new\_tr <- as.matrix(train[,1:1832])   
new\_ts <- as.matrix(test[,1:1832])  
labels <- as.numeric(labels-1)  
ts\_label <- as.numeric(ts\_label-1)  
dtrain <- xgb.DMatrix(data = new\_tr,label = labels)   
dtest <- xgb.DMatrix(data = new\_ts,label=ts\_label)

#default parameters  
params <- list(booster = "gbtree", objective = "binary:logistic", eta=0.1, gamma=1, max\_depth=7, min\_child\_weight=1, subsample=1, colsample\_bytree=1)

set.seed(123)  
xgb1 <- xgb.train (params = params, data = dtrain, nrounds = 150, watchlist = list(val=dtest,train=dtrain), print.every.n = 10, early.stop.round = 100, maximize = T , eval\_metric = "auc")

## Warning: 'print.every.n' is deprecated.  
## Use 'print\_every\_n' instead.  
## See help("Deprecated") and help("xgboost-deprecated").

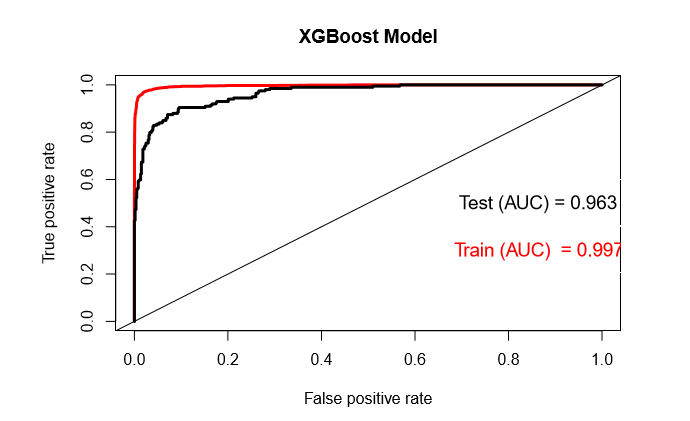
## Warning: 'early.stop.round' is deprecated.  
## Use 'early\_stopping\_rounds' instead.  
## See help("Deprecated") and help("xgboost-deprecated").

## [1] val-auc:0.882765 train-auc:0.889281   
## Multiple eval metrics are present. Will use train\_auc for early stopping.  
## Will train until train\_auc hasn't improved in 100 rounds.  
##   
## [11] val-auc:0.932045 train-auc:0.953255   
## [21] val-auc:0.941682 train-auc:0.969443   
## [31] val-auc:0.947575 train-auc:0.978741   
## [41] val-auc:0.953104 train-auc:0.984693   
## [51] val-auc:0.954633 train-auc:0.987426   
## [61] val-auc:0.956906 train-auc:0.988976   
## [71] val-auc:0.957700 train-auc:0.990373   
## [81] val-auc:0.958609 train-auc:0.991877   
## [91] val-auc:0.959493 train-auc:0.992662   
## [101] val-auc:0.960865 train-auc:0.993759   
## [111] val-auc:0.961964 train-auc:0.994504   
## [121] val-auc:0.962584 train-auc:0.995250   
## [131] val-auc:0.963080 train-auc:0.995751   
## [141] val-auc:0.962981 train-auc:0.996181   
## [150] val-auc:0.963336 train-auc:0.996508

## Visualize Model Generalization and Skill

Below we can see the ROC plot of the train and test data. An AUC near 1 is better, an AUC near 0 represents a model with no skill. We want to see both the train and test have relatively similar AUC’s to show the model generalizes well, and we also want as high of a test AUC as possible.

library(ROCR)  
library(pROC)  
set.seed(1234)  
sample <- sample.int(n = nrow(x), size = floor(.90\*nrow(x)), replace = F)  
train <- x[sample, ]  
test <- x[-sample, ]  
p2 <- predict(xgb1, dtrain, type = 'prob')  
p2 <- as.numeric(p2)  
  
p3 <- predict(xgb1, dtest, type = 'response')  
p3 <- as.numeric(p3)  
  
  
pred1 <- prediction(p2, train$Status)  
roc.perf = ROCR::performance(pred1, measure = "tpr", x.measure = "fpr")  
  
pred2 <- prediction(p3, test$Status)  
roc.perf2 = ROCR::performance(pred2, measure = "tpr", x.measure = "fpr")  
  
plot(roc.perf,col='red', lty=1, lwd=3, main = "XGBoost Model")  
abline(a=0, b= 1)  
  
plot(roc.perf2, add=TRUE, lty=1, lwd=3)  
#roc.perf  
  
  
auctest <- ROCR::performance(pred2,"auc")  
auctrain <- ROCR::performance(pred1,"auc")  
# now converting S4 class to vector  
auctest <- unlist(slot(auctest, "y.values"))  
auctrain <- unlist(slot(auctrain, "y.values"))  
# adding min and max ROC AUC to the center of the plot  
auctest<-mean(round(auctest, digits = 3))  
auctrain<-mean(round(auctrain, digits = 3))  
minauct <- paste(c("Train (AUC) = "), auctrain,sep="")  
maxauct <- paste(c("Test (AUC) = "),auctest,sep="")  
legend(0.62,0.6,c(maxauct),cex=1.2,box.col = "white", text.col = "black")  
legend(0.61,0.4,c(minauct),cex=1.2,box.col = "white", text.col = "red")



## Visualizaing Important Variables

Here you can see how frequently different variables are used in the trees of the xgboost model, suggesting they are important variables. While it is common place to regard some machine learning models as hard to interpret, xgboost actually has a nice waterfall plot that can show how variables played into any given prediction. I will add that at a later date but the intstructional can be found here <https://stats.stackexchange.com/questions/342090/xgboostexplainer-intercept-and-xgboost-parameters>.

xgb.importance(colnames(train, do.NULL = TRUE, prefix = "col"), model = xgb1)

## Feature Gain Cover Frequency  
## 1: Cell 1.476278e-01 0.0480110948 0.0152477764  
## 2: disease\_id\_EKP 1.183667e-01 0.1016144902 0.1141465481  
## 3: Disease.or.Syndrome 1.048511e-01 0.0257493111 0.0247776366  
## 4: Neoplastic.Process 8.936063e-02 0.0361269097 0.0247776366  
## 5: drug\_id\_EKP 5.003431e-02 0.0519386583 0.0616264295  
## ---   
## 157: disease\_id.C0085136 3.785472e-05 0.0007433534 0.0002117747  
## 158: disease\_id.C2973725 2.713300e-05 0.0006823479 0.0002117747  
## 159: disease\_id.C0264716 2.317555e-05 0.0006591431 0.0002117747  
## 160: disease\_id.C0007131 2.198190e-05 0.0006497556 0.0002117747  
## 161: disease\_id.C0085074 2.086935e-05 0.0006450763 0.0002117747

## 10 Fold Cross Validation

For the 10 fold cross validation, the validation will test 10 different chunks of 10% of the data with the other 90% being the train data. This will happen for each round of training. This is more computationally intense but can give a better idea of how well the model generalizes.I have limited the training rounds here due to the time it takes.

labels <- as.numeric(x$Status)  
#Below, by limiting our predictor variable matrix to 1:1832, I make sure I don't leave the response variable in.  
  
new\_tr <- as.matrix(x[,1:1832])   
  
labels <- as.numeric(labels-1)  
  
dall <- xgb.DMatrix(data = new\_tr,label = labels)

set.seed(123)  
xgbcv <- xgb.cv( params = params, data = dall, nrounds = 100, nfold = 10, showsd = T, stratified = T, print.every.n = 10, early.stop.round = 20, maximize = T, eval\_metric = "auc")

## Warning: 'print.every.n' is deprecated.  
## Use 'print\_every\_n' instead.  
## See help("Deprecated") and help("xgboost-deprecated").

## Warning: 'early.stop.round' is deprecated.  
## Use 'early\_stopping\_rounds' instead.  
## See help("Deprecated") and help("xgboost-deprecated").

## [1] train-auc:0.893854+0.003187 test-auc:0.868259+0.012705   
## Multiple eval metrics are present. Will use test\_auc for early stopping.  
## Will train until test\_auc hasn't improved in 20 rounds.  
##   
## [11] train-auc:0.953742+0.001369 test-auc:0.910410+0.012082   
## [21] train-auc:0.969622+0.000851 test-auc:0.918608+0.009296   
## [31] train-auc:0.980590+0.000655 test-auc:0.926121+0.007231   
## [41] train-auc:0.986081+0.000736 test-auc:0.931431+0.006423   
## [51] train-auc:0.988446+0.000663 test-auc:0.933730+0.006231   
## [61] train-auc:0.990148+0.000826 test-auc:0.935663+0.006070   
## [71] train-auc:0.991521+0.000873 test-auc:0.937191+0.006365   
## [81] train-auc:0.992634+0.000855 test-auc:0.938344+0.006172   
## [91] train-auc:0.993580+0.000888 test-auc:0.939418+0.006064   
## [100] train-auc:0.994428+0.000839 test-auc:0.940410+0.005652

## Conclusions (Thus Far)

Building this xgboost model has shown that it can beat the previous AUC of 0.922. Moving forward I will also supply a gains table and lift chart as I have in other projects to quantify how much an improvement of random chance the highest scoring buckets provide. I have done this in other projects available such as <https://github.com/jamesyoung93/StatisticalProgrammingClasses/tree/Predictive-Analytics-I>.