Ortiz\_Javier\_GoogleApps

Javier Ortiz Montenegro

4 de enero de 2019

Parte del código está extraida de: “<https://www.kaggle.com/q2000qfq/rate-predictor>”, ejemplo drug001 y ejemplo XGBoost.

Los datos son distintas caracteristicas de un conjunto de apps de Google Play Store entre las que podemos encontrar la cantidad de descargas, el rating, la cantidad de reviews, a que categoría pertenecen, precio, edad objetivo etc. He considerado que era una base de datos interesante ya que la cantidad de apps que se lanzan al año es enorme y creo que podría ser interesante realizar una predicción de las descargas que va a tener una app antes de ser lanzada en funcion de a que categoría pertenece, la edad del publico objetivo, su precio, etc. Esto ayudaría a las empresas desarrolladoras de apps a una asignación eficiente de sus activos a los proyectos con mayor previsión.

Una vez realizada la introducción del dataset se procederá al tratamiento.

Se empieza cargando las librerías necesarias.

library(dplyr)  
library(tidyr)  
library(xgboost)  
require("caret")  
require("randomForest")  
require("gbm")  
library(plotROC)  
library(ggplot2)

Posteriormente se realiza la carga y tratamiento de los datos. Al realizar un “summary” de los datos cargados nos encontramos frente a varios problemas, algunos errores en valores imposibles (Rating de 19 cuando unicamente es de 1 a 5), una observación se ha introducido incorrectamente, NAs, practicamente todas las variables como factor, etc…

#Carga de datos y revisión de los mismos.  
apps.df = read.csv("googleplaystore.csv")  
summary(apps.df)

## App   
## ROBLOX : 9   
## CBS Sports App - Scores, News, Stats & Watch Live: 8   
## 8 Ball Pool : 7   
## Candy Crush Saga : 7   
## Duolingo: Learn Languages Free : 7   
## ESPN : 7   
## (Other) :10796   
## Category Rating Reviews   
## FAMILY :1972 Min. : 1.000 0 : 596   
## GAME :1144 1st Qu.: 4.000 1 : 272   
## TOOLS : 843 Median : 4.300 2 : 214   
## MEDICAL : 463 Mean : 4.193 3 : 175   
## BUSINESS : 460 3rd Qu.: 4.500 4 : 137   
## PRODUCTIVITY: 424 Max. :19.000 5 : 108   
## (Other) :5535 NA's :1474 (Other):9339   
## Size Installs Type Price   
## Varies with device:1695 1,000,000+ :1579 0 : 1 0 :10040   
## 11M : 198 10,000,000+:1252 Free:10039 $0.99 : 148   
## 12M : 196 100,000+ :1169 NaN : 1 $2.99 : 129   
## 14M : 194 10,000+ :1054 Paid: 800 $1.99 : 73   
## 13M : 191 1,000+ : 907 $4.99 : 72   
## 15M : 184 5,000,000+ : 752 $3.99 : 63   
## (Other) :8183 (Other) :4128 (Other): 316   
## Content.Rating Genres Last.Updated   
## : 1 Tools : 842 August 3, 2018: 326   
## Adults only 18+: 3 Entertainment: 623 August 2, 2018: 304   
## Everyone :8714 Education : 549 July 31, 2018 : 294   
## Everyone 10+ : 414 Medical : 463 August 1, 2018: 285   
## Mature 17+ : 499 Business : 460 July 30, 2018 : 211   
## Teen :1208 Productivity : 424 July 25, 2018 : 164   
## Unrated : 2 (Other) :7480 (Other) :9257   
## Current.Ver Android.Ver   
## Varies with device:1459 4.1 and up :2451   
## 1.0 : 809 4.0.3 and up :1501   
## 1.1 : 264 4.0 and up :1375   
## 1.2 : 178 Varies with device:1362   
## 2.0 : 151 4.4 and up : 980   
## 1.3 : 145 2.3 and up : 652   
## (Other) :7835 (Other) :2520

str(apps.df)

## 'data.frame': 10841 obs. of 13 variables:  
## $ App : Factor w/ 9660 levels "- Free Comics - Comic Apps",..: 7229 2563 8998 8113 7294 7125 8171 5589 4948 5826 ...  
## $ Category : Factor w/ 34 levels "1.9","ART\_AND\_DESIGN",..: 2 2 2 2 2 2 2 2 2 2 ...  
## $ Rating : num 4.1 3.9 4.7 4.5 4.3 4.4 3.8 4.1 4.4 4.7 ...  
## $ Reviews : Factor w/ 6002 levels "0","1","10","100",..: 1183 5924 5681 1947 5924 1310 1464 3385 816 485 ...  
## $ Size : Factor w/ 462 levels "1,000+","1.0M",..: 55 30 368 102 64 222 55 118 146 120 ...  
## $ Installs : Factor w/ 22 levels "0","0+","1,000,000,000+",..: 8 20 13 16 11 17 17 4 4 8 ...  
## $ Type : Factor w/ 4 levels "0","Free","NaN",..: 2 2 2 2 2 2 2 2 2 2 ...  
## $ Price : Factor w/ 93 levels "$0.99","$1.00",..: 92 92 92 92 92 92 92 92 92 92 ...  
## $ Content.Rating: Factor w/ 7 levels "","Adults only 18+",..: 3 3 3 6 3 3 3 3 3 3 ...  
## $ Genres : Factor w/ 120 levels "Action","Action;Action & Adventure",..: 10 13 10 10 12 10 10 10 10 12 ...  
## $ Last.Updated : Factor w/ 1378 levels "1.0.19","April 1, 2016",..: 562 482 117 825 757 901 76 726 1317 670 ...  
## $ Current.Ver : Factor w/ 2834 levels "","0.0.0.2","0.0.1",..: 121 1020 466 2827 279 115 279 2393 1457 1431 ...  
## $ Android.Ver : Factor w/ 35 levels "","1.0 and up",..: 17 17 17 20 22 10 17 20 12 17 ...

#Tratamiento de valores atipicos, NA y duplicados.  
apps.df$Installs[apps.df$Installs == "Free" ] = NA  
apps.df$Rating[apps.df$Rating == 19] = NA  
apps.df$Size[apps.df$Size == "Varies with device"] = NA  
apps.df$Content.Rating[apps.df$Content.Rating == "Unrated" | apps.df$Content.Rating == "Adults only 18+"] = NA  
apps.df = mutate(apps.df[,-c(10:13)]) %>% na.omit %>% droplevels()  
apps.df = apps.df[!duplicated(apps.df),]  
  
#Tratamiento de Reviews pasando de factor a numérico.  
apps.df$Reviews = as.numeric(as.character(apps.df$Reviews))  
  
#Tratamiento del tamaño eliminando las letras del factor para poder transformar a numérico.  
apps.df$Size = as.character(apps.df$Size)  
size\_express\_mb <- "[0-9]+[[:punct:]]?[0-9]\*M"  
size\_express\_kb <- "[0-9]+[[:punct:]]?[0-9]\*k"  
  
index\_mb <- grep(apps.df$Size,pattern = size\_express\_mb)  
mb.match = regexpr(apps.df$Size, pattern = size\_express\_mb)  
mb = regmatches(apps.df$Size,mb.match)  
  
mb1 <- substr(mb, 1,nchar(mb) - 1)  
mb1 <- as.numeric(mb1)\*1000  
apps.df$Size[index\_mb] <- mb1  
  
index\_kb <- grep(apps.df$Size,pattern = size\_express\_kb)  
kb.match = regexpr(apps.df$Size, pattern = size\_express\_kb)  
kb = regmatches(apps.df$Size,kb.match)  
  
kb1 <- substr(kb, 1,nchar(kb) - 1)  
kb1 <- as.numeric(kb1)  
apps.df$Size[index\_kb] <- kb1  
apps.df$Size = as.numeric(apps.df$Size)  
  
# Tratamiento de la variable Install eliminando el símbolo "+" y ",".  
  
apps.df$Installs = as.character(apps.df$Installs)  
apps.df$Installs = substr(apps.df$Installs, 1, nchar(apps.df$Installs)-1)  
apps.df$Installs = gsub(",", "", c(apps.df$Installs))  
apps.df$Installs = as.numeric(apps.df$Installs)  
  
#Tratamiento de la variable Price eliminando el símbolo "$".  
  
apps.df$Price = as.character(apps.df$Price)  
apps.df$Price = substr(apps.df$Price, 2, nchar(apps.df$Price))  
apps.df$Price[apps.df$Price == ""] = 0  
apps.df$Price = as.numeric(apps.df$Price)  
  
#Comprobación de los datos.  
summary(apps.df)

## App Category Rating   
## ROBLOX : 9 FAMILY :1591 Min. :1.000   
## 8 Ball Pool : 7 GAME : 959 1st Qu.:4.000   
## Bubble Shooter : 6 TOOLS : 632 Median :4.300   
## Helix Jump : 6 PERSONALIZATION: 279 Mean :4.171   
## Zombie Catchers : 6 MEDICAL : 277 3rd Qu.:4.500   
## Angry Birds Classic: 5 LIFESTYLE : 273 Max. :5.000   
## (Other) :7380 (Other) :3408   
## Reviews Size Installs Type   
## Min. : 1 Min. : 8.5 Min. :1.000e+00 Free:6872   
## 1st Qu.: 99 1st Qu.: 5100.0 1st Qu.:1.000e+04 Paid: 547   
## Median : 2066 Median : 14000.0 Median :1.000e+05   
## Mean : 278924 Mean : 22752.1 Mean :7.826e+06   
## 3rd Qu.: 36868 3rd Qu.: 33000.0 3rd Qu.:1.000e+06   
## Max. :44893888 Max. :100000.0 Max. :1.000e+09   
##   
## Price Content.Rating  
## Min. : 0.000 Everyone :5956   
## 1st Qu.: 0.000 Everyone 10+: 299   
## Median : 0.000 Mature 17+ : 332   
## Mean : 1.117 Teen : 832   
## 3rd Qu.: 0.000   
## Max. :400.000   
##

str(apps.df)

## 'data.frame': 7419 obs. of 9 variables:  
## $ App : Factor w/ 7026 levels "- Free Comics - Comic Apps",..: 5233 1845 6564 5899 5278 5162 5938 4005 3538 4183 ...  
## $ Category : Factor w/ 33 levels "ART\_AND\_DESIGN",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ Rating : num 4.1 3.9 4.7 4.5 4.3 4.4 3.8 4.1 4.4 4.7 ...  
## $ Reviews : num 159 967 87510 215644 967 ...  
## $ Size : num 19000 14000 8700 25000 2800 5600 19000 29000 33000 3100 ...  
## $ Installs : num 1e+04 5e+05 5e+06 5e+07 1e+05 5e+04 5e+04 1e+06 1e+06 1e+04 ...  
## $ Type : Factor w/ 2 levels "Free","Paid": 1 1 1 1 1 1 1 1 1 1 ...  
## $ Price : num 0 0 0 0 0 0 0 0 0 0 ...  
## $ Content.Rating: Factor w/ 4 levels "Everyone","Everyone 10+",..: 1 1 1 4 1 1 1 1 1 1 ...  
## - attr(\*, "na.action")= 'omit' Named int 24 38 43 53 68 69 74 86 89 90 ...  
## ..- attr(\*, "names")= chr "24" "38" "43" "53" ...

Se separan los datos en train/validation/test una vez tratados.

#Definimos una semilla para que el ejercicio sea reproducible  
set.seed(2019)  
# Se separa de la siguiente manera: 60% a train, 20% a validate y 20% a test  
inTraining <- createDataPartition(apps.df$Installs, p=0.6, list=FALSE)  
training.set <- apps.df[inTraining,]  
Totalvalidation.set <- apps.df[-inTraining,]  
# Ahora creamos una nueva partición del 40% de los datos, 20% a testing y 20% a validation  
inValidation <- createDataPartition(Totalvalidation.set$Installs, p=0.5, list=FALSE)  
testing.set <- Totalvalidation.set[inValidation,]  
validation.set <- Totalvalidation.set[-inValidation,]  
  
dataset <- training.set  
validation <- validation.set  
test <- testing.set  
  
# El modelo final usara todos los datos menos test.  
total <- rbind(dataset, validation)

Se pasa a definir los parametros de control y el grid para realizar el tuning de los distintos modelos a comparar.

set.seed(2019)  
fitControl <- trainControl(method = 'cv', number = 5, summaryFunction=defaultSummary)  
  
  
#El primer Grid es el de boosting.  
gbmGrid <- expand.grid(interaction.depth = c(1,4,7,10),  
 n.trees = c(500, 1000, 2000),  
 shrinkage = c(.005, .02,.05),  
 n.minobsinnode = 10)  
  
  
  
#En segundo lugar XGBoost.  
tuneGridXGB <- expand.grid(  
 nrounds=c(350),  
 max\_depth = c(4, 6),  
 eta = c(0.05, 0.1),  
 gamma = c(0.01),  
 colsample\_bytree = c(0.75),  
 subsample = c(0.50),  
 min\_child\_weight = c(0))  
  
#Por ultimo definimos el de Random Forest.  
#mtry max:  
ncol(apps.df)-1

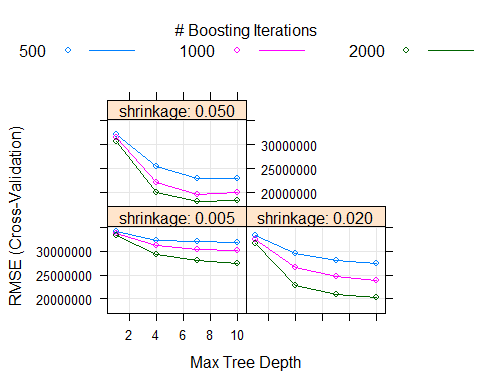
## [1] 8

rfGrid <- expand.grid(mtry = c(1,2,3,4,6,8))

Empezamos la comparación con el modelo boosting.

set.seed(2019)  
### Gradient boosting machine algorithm. ###  
fit.gbm <- train(Installs~. -App, data=dataset, method = 'gbm', trControl=fitControl, tuneGrid=gbmGrid, metric='RMSE')

plot(fit.gbm)



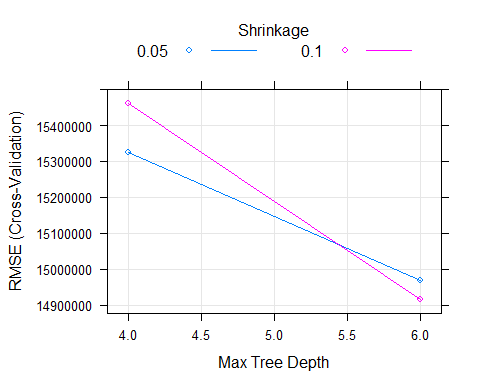
#Como se puede ver en el gráfico el mejor "tune" corresponde a un shrinkage de 0.05, 2000 arboles e "interaction depth"" de 7  
fit.gbm$bestTune

## n.trees interaction.depth shrinkage n.minobsinnode  
## 33 2000 7 0.05 10

res\_gbm <- fit.gbm$results  
RMSE\_gbm <- subset(res\_gbm[5])  
# CV con mejor "tune"  
BCV.RMSE = max(RMSE\_gbm)  
  
boost.caret.pred <- predict(fit.gbm,validation)  
B.RMSE = (mean((boost.caret.pred - validation$Installs)^2))^0.5

En segundo lugar XGBoosting.

set.seed(2019)  
### EXtreme Gradient boosting algorithm. ###  
fit.gbmx <- train(Installs~. -App, data=dataset, method = 'xgbTree', trControl=fitControl, tuneGrid=tuneGridXGB, metric='RMSE')  
  
plot(fit.gbmx)



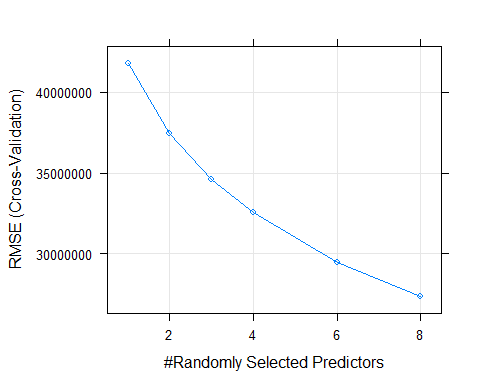
#Se muestran abajo los parámetros del mejor "tune" que se puede apreciar en la gráfica.  
fit.gbmx$bestTune

## nrounds max\_depth eta gamma colsample\_bytree min\_child\_weight subsample  
## 4 350 6 0.1 0.01 0.75 0 0.5

res\_gbmx <- fit.gbmx$results  
RMSE\_gbmx <- subset(res\_gbmx[8])  
# CV con mejor "tune"  
XBCV.RMSE = max(RMSE\_gbmx)  
  
xboost.caret.pred <- predict(fit.gbmx,validation)  
XB.RMSE = (mean((xboost.caret.pred - validation$Installs)^2))^0.5

Por último Random Forest.

set.seed(2019)  
### Random Forest algorithm. ###  
fit.rf <- train(Installs~. -App, data=dataset, method = 'rf', trControl=fitControl, tuneGrid=rfGrid, metric='RMSE')  
  
plot(fit.rf)



#Como se puede ver en la gráfica el número óptimo de predictores seleccionados es 8.  
  
res\_rf <- fit.rf$results  
RMSE\_rf <- subset(res\_rf[2])   
  
# CV con mejor "tune"   
RFCV.RMSE = max(RMSE\_rf)  
  
  
rf.caret.pred <- predict(fit.rf,validation)  
RF.RMSE = (mean((rf.caret.pred - validation$Installs)^2))^0.5

Resultados RMSE

|  |  |  |  |
| --- | --- | --- | --- |
|  | Boosting | XGB | Random Forest |
| CV | 34127383 | 15462078 | 41809107 |
| Validation set | 21712259 | 12386801 | 27337803 |

En todos los casos el validation set da mejores resultados que Cross Validation al comparar por RMSE. No obstante, el mejor resultado corresponde a eXtreme Gradient Boosting, por lo tanto será el modelo que se usará para la predicción evaluando en test usando el 100% de los datos.

set.seed(2019)  
fit.xgb\_total <- train(Installs~. -App, data=total, method = 'xgbTree', trControl=fitControl, tuneGrid=tuneGridXGB, metric='RMSE')  
fit.xgb\_total

## eXtreme Gradient Boosting   
##   
## 5935 samples  
## 8 predictor  
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 4747, 4749, 4748, 4748, 4748   
## Resampling results across tuning parameters:  
##   
## eta max\_depth RMSE Rsquared MAE   
## 0.05 4 15629936 0.8320442 3490602  
## 0.05 6 16334632 0.8194400 3276945  
## 0.10 4 15142301 0.8289312 3343638  
## 0.10 6 16668455 0.8184251 3387337  
##   
## Tuning parameter 'nrounds' was held constant at a value of 350  
## 0.75  
## Tuning parameter 'min\_child\_weight' was held constant at a value  
## of 0  
## Tuning parameter 'subsample' was held constant at a value of 0.5  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were nrounds = 350, max\_depth = 4,  
## eta = 0.1, gamma = 0.01, colsample\_bytree = 0.75, min\_child\_weight =  
## 0 and subsample = 0.5.

fit.xgb\_total$bestTune

## nrounds max\_depth eta gamma colsample\_bytree min\_child\_weight subsample  
## 3 350 4 0.1 0.01 0.75 0 0.5

res\_xgb\_total <- fit.xgb\_total$results  
RMSE\_xgb\_total <- subset(res\_xgb\_total[8])  
# CV con mejor "tune"  
XBCV\_T.RMSE = max(RMSE\_xgb\_total)  
  
#Evaluamos en test  
xboost.caret.pred\_total <- predict(fit.xgb\_total,test)  
XB\_T.RMSE = (mean((xboost.caret.pred\_total - test$Installs)^2))^0.5

Resultados RMSE

|  |  |
| --- | --- |
|  | eXtreme Gradient Boosting |
| CV | 16668455 |
| Validation set | 16263415 |

Se puede apreciar que aun con una diferencia menor validation set sigue teniendo un mejor RMSE que CV. Por lo tanto este sería el mejor modelo para predecir las descargas de una app con una error medio de 1.626341510^{7}. Teniendo en cuenta el gran tamaño de la variable a predecir es un resultado aceptable.