**Primera Corrida:**

**# Utilizaré XGBoost para el modelo de clasificación**

**# Activar librerías necesarias para el algoritmo**

library(mice)

library(VIM)

library(caret)

library(ggplot2)

library(GGally)

**# Importar el dataset y analizar las variables**

training\_set = read.csv('train.csv', stringsAsFactors = TRUE)

**# Ploteo Multivariante**

ggpairs(training\_set[,5:14],

aes(colour = Approve.Loan,

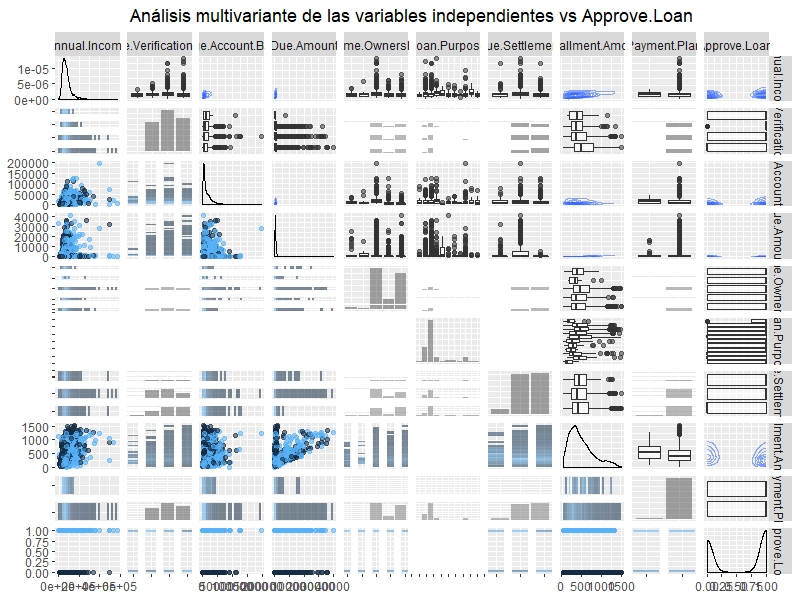
alpha = 0.4),

title = "Análisis multivariante de las variables independientes vs Approve.Loan",

upper = list(continuous = "density"),

lower = list(combo = "denstrip"))+

theme(plot.title = element\_text(hjust = 0.5))



# Del gráfico, vemos como la diagonal principal contiene la distribución de las variables,

# cada fila y columna representa el cruce de cada variable, siendo por debajo de la diagonal

# Scatter plots que permiten ver la relación de las variables y ver si son correlacionadas o n

#por encima de la diagonal vemos box plots para cuando una de las variables es categórica

# y la otra numérica (hacer lo mismo para las variables restantes si se desea ver estas

# relaciones.

data\_nueva = read.csv('test.csv', stringsAsFactors = TRUE)

data\_nueva$id=NULL

str(training\_set) # Hay variables de tipo numérica y de tipo categórica

head(training\_set) # Ver primeros valores de las variables

summary(training\_set) # Ver proncipales estadísticos

md.pattern(training\_set) # Existen 329 valores perdidos (NAs), hay que rellenarlos con alguna técnica.

**# Llenar valores perdidos con imputaciones**

# a) Si la variable es numéricá con predictive mean matching

# b) Si la variable es categórica con multinomial logistic regression

p <- function(x) {sum(is.na(x))/length(x)\*100}

apply(training\_set, 2, p)

impute <- mice(training\_set[,2:14], m=5, seed = 1234)

print(impute)

**# Completar los datos perdidos**

# Haré 5 training sets para ver con cual se clasifica mejor

training\_new1 <- complete(impute, 1)

training\_new2 <- complete(impute, 2)

training\_new3 <- complete(impute, 3)

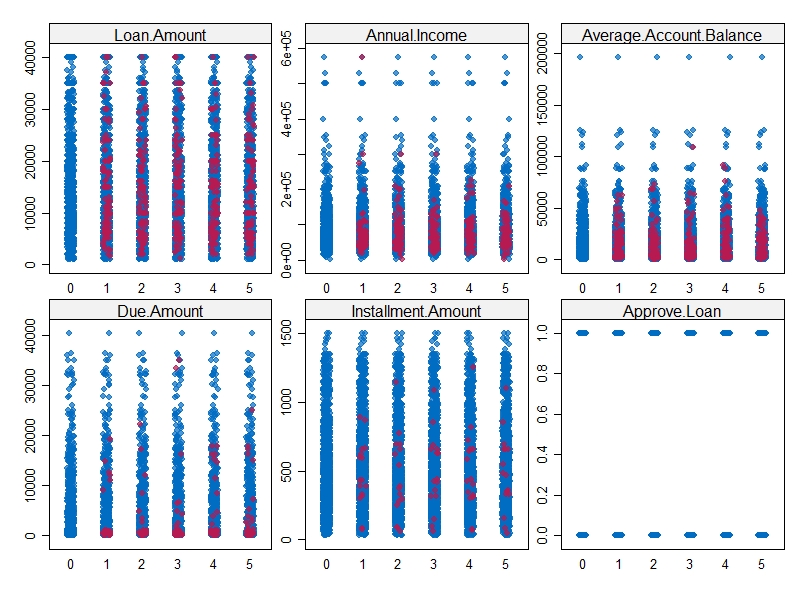
training\_new4 <- complete(impute, 4)

training\_new5 <- complete(impute, 5)

**# Distribución de valores observados e imputados**

# Los rojos son los imputados y los azules los observados

stripplot(impute, pch = 20, cex = 1.2)



**#**Del gráfico, apreciamos que se están imputando correctamente los valores perdidos (rojo), no se

**#** ve que existan importaciones lejanas a los datos (azul). Esta metodología de reemplazo de NAs

**#** es muy potente.

**# Codificar los factores para tener imput correcto del modelo**

str(training\_new1)

training\_new1$Term = as.numeric(factor(training\_new1$Term, labels = c(1:3)))

training\_new1$State = as.numeric(factor(training\_new1$State, labels = c(1:49)))

training\_new1$Income.Verification.Status = as.numeric(factor(training\_new1$Income.Verification.Status, labels = c(1:4)))

training\_new1$Home.Ownership = as.numeric(factor(training\_new1$Home.Ownership, labels = c(1:5)))

training\_new1$Loan.Purpose = as.numeric(factor(training\_new1$Loan.Purpose, labels = c(1:11)))

training\_new1$Due.Settlement = as.numeric(factor(training\_new1$Due.Settlement, labels = c(1:3)))

training\_new1$Payment.Plan = as.numeric(factor(training\_new1$Payment.Plan, labels = c(1:2)))

data\_nueva$Term = as.numeric(factor(data\_nueva$Term, labels = c(1:2)))

data\_nueva$State = as.numeric(factor(data\_nueva$State, labels = c(1:49)))

data\_nueva$Income.Verification.Status = as.numeric(factor(data\_nueva$Income.Verification.Status, labels = c(1:3)))

data\_nueva$Home.Ownership = as.numeric(factor(data\_nueva$Home.Ownership, labels = c(1:4)))

data\_nueva$Loan.Purpose = as.numeric(factor(data\_nueva$Loan.Purpose, labels = c(1:11)))

data\_nueva$Due.Settlement = as.numeric(factor(data\_nueva$Due.Settlement, labels = c(1:2)))

data\_nueva$Payment.Plan = as.numeric(factor(data\_nueva$Payment.Plan, labels = c(1)))

str(training\_new1)

str(data\_nueva)

md.pattern(training\_new1)

**# Dividir los datos en conjunto de entrenamiento y conjunto de test**

# install.packages("caTools")

library(caTools)

set.seed(123)

split = sample.split(training\_new1$Approve.Loan, SplitRatio = 0.8)

entrenamiento = subset(training\_new1, split == TRUE)

testeo = subset(training\_new1, split == FALSE)

**# Aplicar el algoritmoo XGBoost**

classifier = xgboost(data = as.matrix(entrenamiento[, -13]),

label = entrenamiento$Approve.Loan,

nrounds = 100,

objective = "binary:logistic",

max.depth = 5,

nthread = 2,

verbose = 1)

**# Predecir con el clasificador**

y\_pred = predict(classifier, newdata = as.matrix(testeo[,-13]))

y\_pred = (y\_pred > 0.5)

**# Hallar matríz de confusión y porcentaje de aciertos**

cm = table(testeo[, 13], y\_pred)

accuracy=sum(diag(cm))/sum(cm)

> cm

y\_pred

FALSE TRUE

0 153 12

1 8 206

**#**De la tabla, apreciamos que 153 casos fueron predichos como 0 (no prestar) siendo 0 en la

**#**realidad, además, 206 casos fueron predichos como 1(si prestar) siendo 1 en la realidad. Tan solo

**#**han sido clasificados incorrectamente 20 casos.

> accuracy

[1] 0.9472296

**Conclusión:**

El 94.72% de los casos han sido predichos correctamente con el modelo XGBoost, es un modelo excelente (más del 90% de los casos son predichos correctamente), ahora en el siguiente paso lo aplicaremos a la data nueva para generar una columna extra al final que contenga los valores predichos para saber si le debemos prestar o no a la gente.

**Segunda Corrida (Con el conjunto a determinar si se presta o no a los clientes):**

**# Utilizaré XGBoost para el modelo de clasificación**

**# Activar librerías necesarias para el algoritmo**

library(mice)

library(VIM)

library(caret)

**# Importar el dataset y analizar las variables**

training\_set = read.csv('train.csv', stringsAsFactors = TRUE)

testing\_set = read.csv('test.csv', stringsAsFactors = TRUE)

testing\_set$id=NULL

str(training\_set) # Hay variables de tipo numérica y de tipo categórica

head(training\_set) # Ver primeros valores de las variables

summary(training\_set) # Ver proncipales estadísticos

md.pattern(training\_set) # Existen 329 valores perdidos (NAs), hay que rellenarlos con alguna técnica.

**# Llenar valores perdidos con imputaciones**

# a) Si la variable es numéricá con predictive mean matching

# b) Si la variable es categórica con multinomial logistic regression

p <- function(x) {sum(is.na(x))/length(x)\*100}

apply(training\_set, 2, p)

impute <- mice(training\_set[,2:14], m=5, seed = 1234)

print(impute)

**# Completar los datos perdidos**

# Haré 5 training sets para ver con cual se clasifica mejor

training\_new1 <- complete(impute, 1)

training\_new2 <- complete(impute, 2)

training\_new3 <- complete(impute, 3)

training\_new4 <- complete(impute, 4)

training\_new5 <- complete(impute, 5)

**# Distribución de valores observados e imputados**

# Los rojos son los imputados y los azules los observados

stripplot(impute, pch = 20, cex = 1.2)

**# Codificar los factores para tener imput correcto del modelo**

str(training\_new1)

training\_new1$Term = as.numeric(factor(training\_new1$Term, labels = c(1:3)))

training\_new1$State = as.numeric(factor(training\_new1$State, labels = c(1:49)))

training\_new1$Income.Verification.Status = as.numeric(factor(training\_new1$Income.Verification.Status, labels = c(1:4)))

training\_new1$Home.Ownership = as.numeric(factor(training\_new1$Home.Ownership, labels = c(1:5)))

training\_new1$Loan.Purpose = as.numeric(factor(training\_new1$Loan.Purpose, labels = c(1:11)))

training\_new1$Due.Settlement = as.numeric(factor(training\_new1$Due.Settlement, labels = c(1:3)))

training\_new1$Payment.Plan = as.numeric(factor(training\_new1$Payment.Plan, labels = c(1:2)))

testing\_set$Term = as.numeric(factor(testing\_set$Term, labels = c(1:2)))

testing\_set$State = as.numeric(factor(testing\_set$State, labels = c(1:49)))

testing\_set$Income.Verification.Status = as.numeric(factor(testing\_set$Income.Verification.Status, labels = c(1:3)))

testing\_set$Home.Ownership = as.numeric(factor(testing\_set$Home.Ownership, labels = c(1:4)))

testing\_set$Loan.Purpose = as.numeric(factor(testing\_set$Loan.Purpose, labels = c(1:11)))

testing\_set$Due.Settlement = as.numeric(factor(testing\_set$Due.Settlement, labels = c(1:2)))

testing\_set$Payment.Plan = as.numeric(factor(testing\_set$Payment.Plan, labels = c(1)))

str(training\_new1)

str(testing\_set)

md.pattern(training\_new1)

# Aplicando el algoritmoo XGBoost

classifier = xgboost(data = as.matrix(training\_new1[, -13]),

label = training\_new1$Approve.Loan,

nrounds = 100,

objective = "binary:logistic",

max.depth = 5,

nthread = 2,

verbose = 1)

y\_pred = predict(classifier, newdata = as.matrix(testing\_set[,]))

testing\_set$Approve.Loan = ifelse((y\_pred > 0.5),1,0)

head(testing\_set)

> head(testing\_set)

Loan.Amount Term State Annual.Income Income.Verification.Status Average.Account.Balance Due.Amount

1 21300 1 18 47360 3 13879 791.48

2 11000 1 33 80000 1 2591 357.48

3 16000 1 14 89000 1 3019 10593.00

4 12000 2 38 41000 2 4142 298.17

5 5000 1 5 20000 3 3448 178.24

6 5000 1 43 33000 2 7448 164.74

Home.Ownership Loan.Purpose Due.Settlement Installment.Amount Payment.Plan Approve.Loan

1 2 3 2 791.48 1 0

2 4 2 2 357.48 1 1

3 3 2 1 527.16 1 1

4 4 3 2 298.17 1 1

5 4 3 2 178.24 1 0

6 2 3 1 164.74 1 1

**Conclusión:**

En la columna final de testing\_set apreciamos con 0 a los clientes que no se les debería prestar y con 1 a los que sí se les debería prestar. Se recomienda al área de producto correspondiente que se priorice prestar a aquellos clientes que tienen “y\_pred” más cercano a 1 (ya que es más probable que estos cierren el préstamo y paguen con mayor seguridad.

En caso se tenga una oferta especial con baja tasa de interés, se podría ajustar el modelo para que la última columna “Approve.Loan” sea 1 cuando la probabilidad en “y\_pred” sea 0.9 a más (excelentes pagadores con baja riesgo).