Curso : MACHINE LEARNING CON R

Tema : Preprocesamiento de Datos.

Docente : Victor Henostroza

**GUIA DE LABORATORIO 7**

**Objetivos del laboratorio:**

1. **Desarrollar un Modelo de Series Temporales (Descomposición, Predicción).**
2. **Desarrollar una Modelo de Regla de Asociación APRIORI.**
3. **Desarrollar un modelo de mejora del rendimiento con K-Fold-Cross-Validation que realice 10 iteraciones (K = 10).**
4. **Desarrollar un modelo de mejora del rendimiento con Grid Search para la selección de los mejores hiperparámetros.**
5. **Desarrollar modelos de mejora del rendimiento cuando existe desequilibrio de clases (librería Rose).**
6. **Desarrollar un modelo de clasificación con XGBoost (modelo que permite trabajar con grandes cantidades de datos y muchos parámetros).**

**----------------------------------------------------------------------------------------------**

**Objetivo 1:**

En el siguiente problema, utilizaremos un dataset que muestra cómo ha ido evolucionando la cantidad de viajeros por mes de una aerolínea, se apreciará esto con un objeto serie temporal que nos servirá para predecir cómo serán las ventas de vuelos de la empresa para los próximos 4 años teniendo en cuenta la tendencia, estacionalidad y parte aleatoria de la serie temporal.

**# Series Temporales**

**# Importar el dataset**

data("AirPassengers")

AP <- AirPassengers

> AP

Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec

1949 112 118 132 129 121 135 148 148 136 119 104 118

1950 115 126 141 135 125 149 170 170 158 133 114 140

1951 145 150 178 163 172 178 199 199 184 162 146 166

1952 171 180 193 181 183 218 230 242 209 191 172 194

str(AP)

> str(AP)

Time-Series [1:144] from 1949 to 1961: 112 118 132 129 121 135 148 148 136 119 ...

head(AP)

> head(AP)

[1] 112 118 132 129 121 135

**# Visualizar serie de tiempo**

# ts(objeto serie temporal, frecuencia por año, año y mes de inicio)

ts(AP, frequency = 12, start=c(1949,1))

attributes(AP)

> attributes(AP)

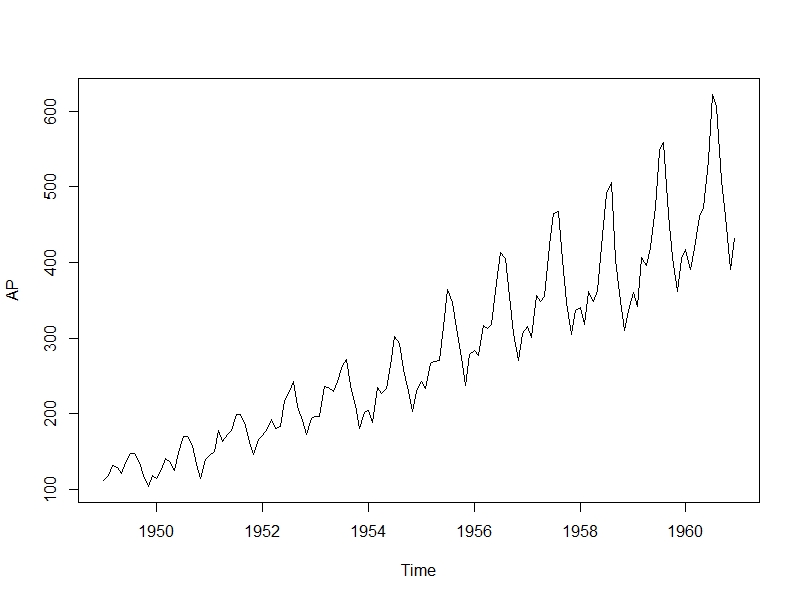
$tsp

[1] 1949.000 1960.917 12.000

$class

[1] "ts"

plot(AP)

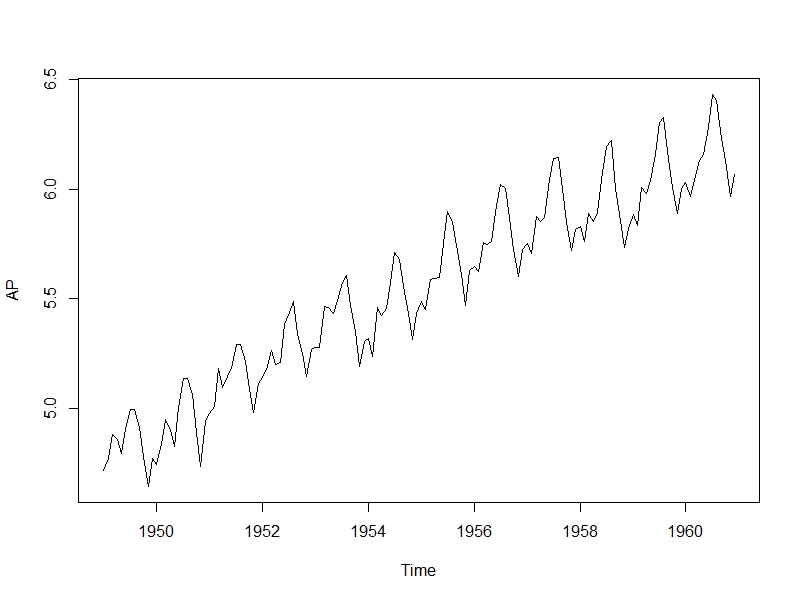
****

Del gráfico, observamos como la cantidad de pasajeros (eje y) va aumentando con el paso de los años (eje x). Además de la tendencia creciente del gráfico, apreciamos que existe una cierta estacionalidad año a año, repitiéndose picos y descensos abruptos, si nos fijamos bien, dichos picos y descensos abruptos se están incrementando con el paso de los años, esto es algo que debe evitarse para generar un análisis satisfactorio de la serie temporal.

**# Transformar con logaritmo**

AP <- log(AP)

plot(AP)



Del gráfico, apreciamos como ahora la estacionalidad no tiene picos ni caídas crecientes con los años.

**# Descomponer la serie temporal en componentes estacionaria, tendencia e irregular**

# La técnica usada es la de promedios móviles

decomp <- decompose(AP)

decomp$figure

> decomp$figure

[1] -0.085815019 -0.114412848 0.018113355 -0.013045611 -0.008966106 0.115392997 0.210816435

[8] 0.204512399 0.064836351 -0.075271265 -0.215845612 -0.100315075

plot(decomp$figure,

type = 'b',

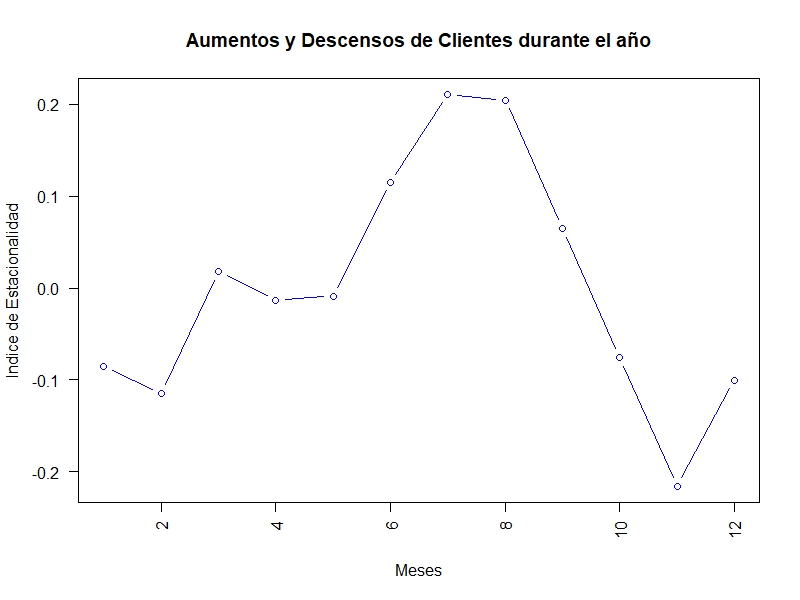
xlab = 'Meses',

ylab = 'Indice de Estacionalidad',

main = 'Aumentos y Descensos de Clientes durante el año',

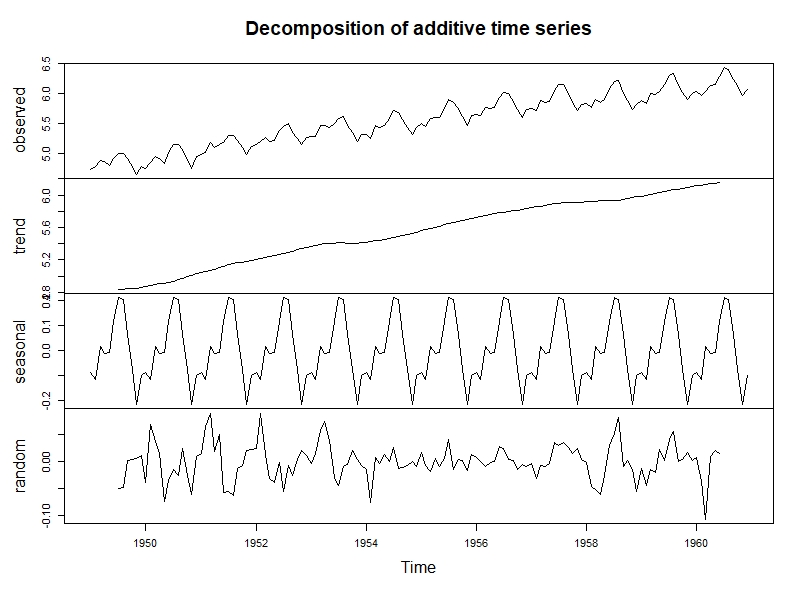
col = 'blue',

las = 2) #pone números en vertical



Del gráfico, se muestra como fluctúa la cantidad de cliente durante el año, en el eje “x” tenemos a los 12 meses y en el eje “y” tenemos el porcentaje de cambio en la cantidad de clientes. Vemos que al centro está el promedio con 0.0, los meses de julio y agosto tienen los picos más altos de ventas (aproximadamente un 20% más de lo habitual) y el mes de noviembre tiene el descenso más bajo de ventas (aproximadamente un 20% menos de lo habitual).

plot(decomp)



Del gráfico, vemos en la parte superior la serie temporal original (luego de aplicar logaritmo), esta se descompone en 3 partes:

Trend = Componente de tendencia

Seasonal = Componente estacionaria

Random = Componente aleatorio (ruido)

La suma de los puntos de estas tres partes descompuesta debe dar la serie original.

**# Generar modelo ARIMA (Media móvil integrada autorregresiva)**

library(forecast)

model <- auto.arima(AP)

> model

Series: AP

ARIMA(0,1,1)(0,1,1)[12] # (p,d,q) del modelo Arima

Coefficients:

ma1 sma1

-0.4018 -0.5569 # coeficientes

s.e. 0.0896 0.0731 # error estándar

sigma^2 estimated as 0.001371: log likelihood=244.7

**AIC=-483.4** AICc=-483.21 **BIC=-474.77**

De la instrucción anterior, vemos el criterio de información de Akaike (AIC) y el criterio de información Bayesiano (BIC), estos criterios ayudarán a encontrar el mejor modelo de serie temporal.

attributes(model)

> attributes(model)

$names

[1] "coef" "sigma2" "var.coef" "mask" "loglik" "aic" "arma"

[8] "residuals" "call" "series" "code" "n.cond" "nobs" "model"

[15] "bic" "aicc" "x" "fitted"

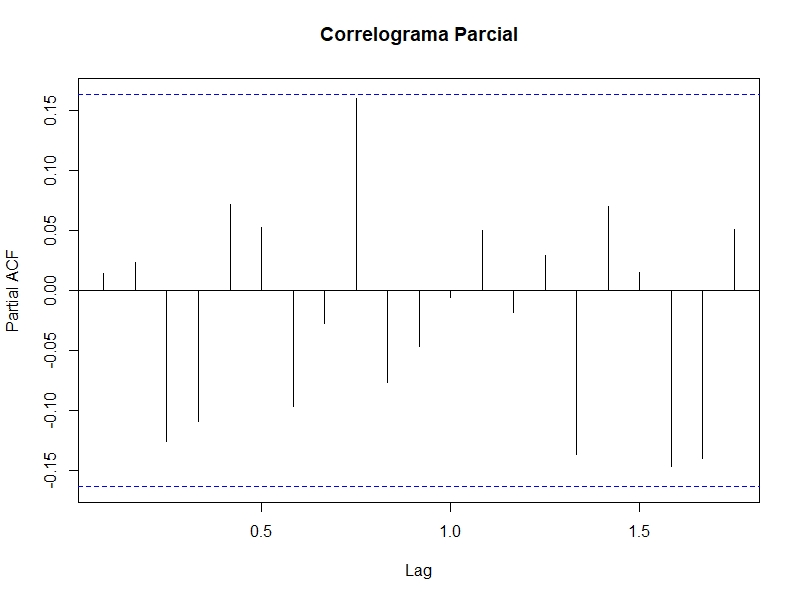
$class

[1] "forecast\_ARIMA" "ARIMA" "Arima"

**# Plotear PACF**

**#PACF** es una función de autocorrelación parcial. Básicamente, encuentra la correlación de los **#**residuos con el siguiente valor de retraso, por eso se llama'parcial' y no 'completo', ya que **#**eliminamos las variaciones ya encontradas antes de encontrar la siguiente correlación.

pacf(model$residuals, main = 'Correlograma Parcial' )

****

Del gráfico, apreciamos como se generan los errores (diferencia entre valor real y pronosticado) de la serie temporal, estos están dentro del intervalo de las líneas punteadas, un mejor modelo haría que estas líneas verticales sean más pequeñas en promedio.

**# Test de Ljung-Box**

Box.test(model$residuals, lag=20, type = 'Ljung-Box')

**# La prueba de Ljung-Box se puede definir de la siguiente manera:**

**#Ho**: Los datos se distribuyen de forma independiente (es decir, las correlaciones en la **#**población de la que se toma la muestra son 0, de modo que cualquier correlación observada **#**en los datos es el resultado de la aleatoriedad del proceso de muestreo).

**#Ha**: Los datos no se distribuyen de forma independiente.

> Box.test(model$residuals, lag=20, type = 'Ljung-Box')

Box-Ljung test

data: model$residuals

X-squared = 17.688, df = 20, p-value = 0.6079

**# Plotear Residuos**

hist(model$residuals,

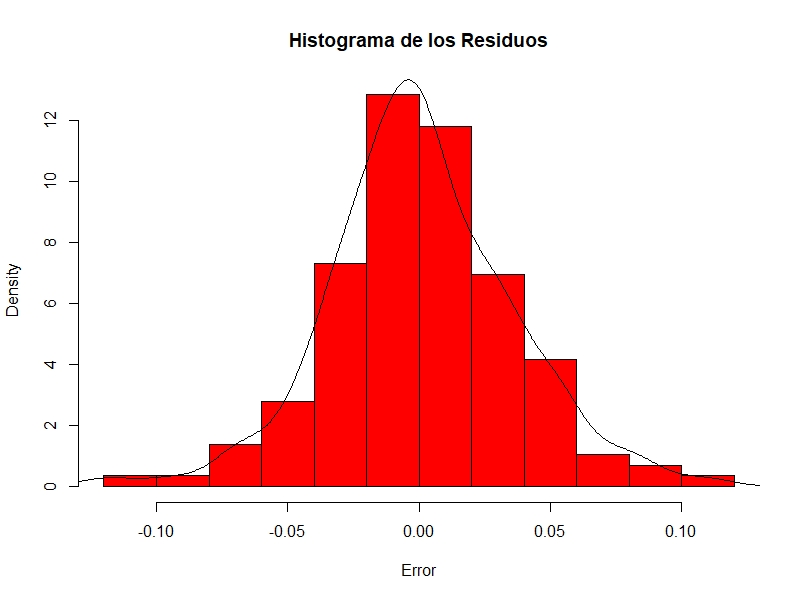
col = 'red',

xlab = 'Error',

main = 'Histograma de los Residuos',

freq = FALSE)

lines(density(model$residuals))

****

Del gráfico, apreciamos que los residuos se distribuyen normalmente con media 0, la curva ideal normal se ajusta bastante bien a los datos.

**# Predecir con la serie temporal**

f <- forecast(model, 48)

> f

Point Forecast Lo 80 Hi 80 Lo 95 Hi 95

Jan 1961 6.110186 6.062729 6.157642 6.037607 6.182764

Feb 1961 6.053775 5.998476 6.109074 5.969203 6.138347

Mar 1961 6.171715 6.109555 6.233874 6.076650 6.266779

Apr 1961 6.199300 6.130966 6.267635 6.094792 6.303809

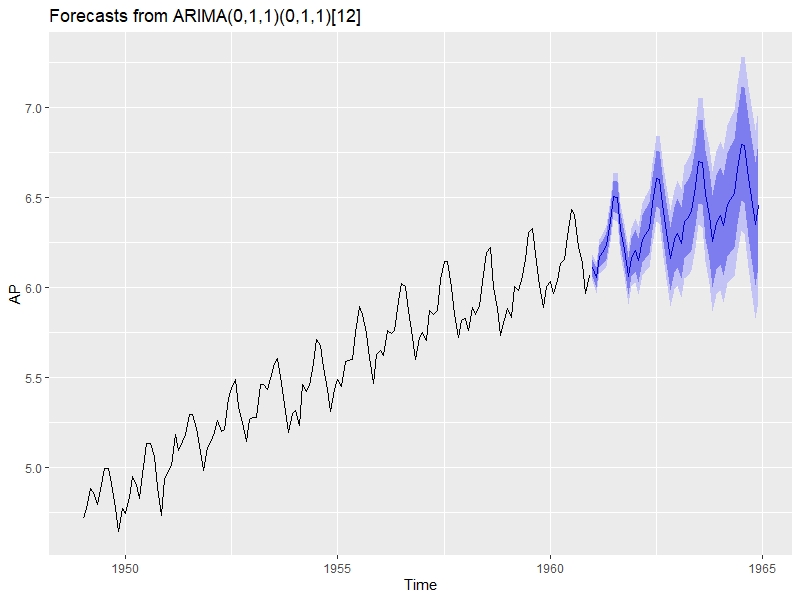
May 1961 6.232556 6.158560 6.306552 6.119388 6.345724

Jun 1961 6.368779 6.289524 6.448033 6.247569 6.489988

De la tabla, en la columna Point Forecast vemos el valor de predicción de cantidad de clientes, siendo para enero de 1961 de 6.11, en la segunda y tercera columna se muestra el intervalo de confianza de dicho punto al 80% de la misma y las columnas cuarta y quinta los intervalos de confianza con el 95%

library(ggplot2)

autoplot(f)



Del gráfico, vemos como se grafica con color azul los valores futuros de la serie temporal de los próximos 48 meses (4 años),

accuracy(f)

> accuracy(f)

ME RMSE MAE MPE MAPE MASE ACF1

Training set 0.0005730622 0.03504883 0.02626034 0.01098898 0.4752815 0.2169522 0.01443892

**Objetivo 2:**

Desarrollaremos en esta parte un modelo de asociación A PRIORI con el dataset “Market\_Basquet\_Optimisation”, este contiene las compras por ticket que se realizan durante la semana de un minimarket, donde cada fila corresponde a la compra que realizó un determinado cliente. En base a estos datos, queremos determinar las mejores combinaciones posibles de compra para generar una mejor distribución de los productos en los pasillos del minimarket y con ello repercutir en el aumento las ventas.

**# Apriori**

**# Preprocesado de Datos**

#install.packages("arules")

library(arules)

dataset = read.csv("Market\_Basket\_Optimisation.csv", header = FALSE)

head(dataset)

|  |
| --- |
| > head(dataset)  V1 V2 V3 V4 V5 V6 V7  1 shrimp almonds avocado vegetables mix green grapes whole weat flour yams  2 burgers meatballs eggs  3 chutney  4 turkey avocado  5 mineral water milk energy bar whole wheat rice green tea  6 low fat yogurt |

De la tabla, apreciamos como se encuentra las compras de cada cliente en cada fila y las columnas no tienen nombre, pero contienen algún producto que llegó a comprar el cliente en cuestión, si la columna está vacía es porque no hay más productos comprados. Este tipo de dataset debe ser modificado a una matriz esparcida, en la cual se mantengan las compras por fila, pero las columnas sean cada uno de los alimentos o cosas que se compraron, para que así, si se llegó a comprar ese alimento este se encuentre con un nivel positivo y si no lo compró este vacío.

**# Generar Matríz Esparcida**

dataset = read.transactions("Market\_Basket\_Optimisation.csv",

sep = ",", rm.duplicates = TRUE)

summary(dataset)

> summary(dataset)

transactions as itemMatrix in sparse format with

7501 rows (elements/itemsets/transactions) and

119 columns (items) and a density of 0.03288973

most frequent items:

mineral water eggs spaghetti french fries chocolate (Other)

1788 1348 1306 1282 1229 22405

element (itemset/transaction) length distribution:

sizes

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 20

1754 1358 1044 816 667 493 391 324 259 139 102 67 40 22 17 4 1 2 1

Min. 1st Qu. Median Mean 3rd Qu. Max.

1.000 2.000 3.000 3.914 5.000 20.000

includes extended item information - examples:

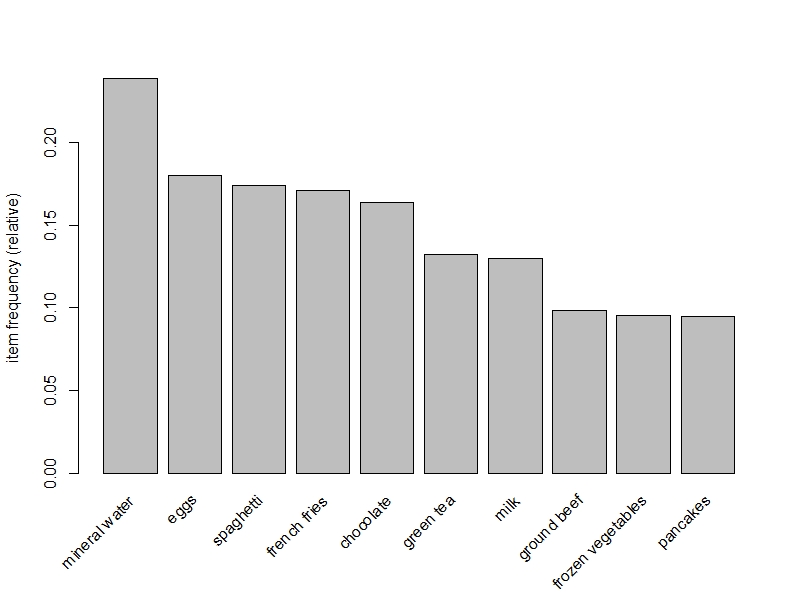
labels

1 almonds

2 antioxydant juice

3 asparagus

itemFrequencyPlot(dataset, topN = 10)



Del gráfico, apreciamos la cantidad (en porcentaje) de compra de los 10 principales productos, estos productos son muy probables de encontrarse en las próximas reglas de asociación que hallaremos.

**# Entrenar algoritmo Apriori con el dataset**

rules = apriori(data = dataset,

parameter = list(support = 0.004, confidence = 0.2))

> rules = apriori(data = dataset,

+ parameter = list(support = 0.004, confidence = 0.2))

Apriori

Parameter specification:

confidence minval smax arem aval originalSupport maxtime support minlen maxlen target ext

0.2 0.1 1 none FALSE TRUE 5 0.004 1 10 rules FALSE

Algorithmic control:

filter tree heap memopt load sort verbose

0.1 TRUE TRUE FALSE TRUE 2 TRUE

Absolute minimum support count: 30

set item appearances ...[0 item(s)] done [0.00s].

set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s].

sorting and recoding items ... [114 item(s)] done [0.00s].

creating transaction tree ... done [0.00s].

checking subsets of size 1 2 3 4 done [0.00s].

writing ... [811 rule(s)] done [0.00s].

creating S4 object ... done [0.00s].

**# Visualización de los resultados**

inspect(sort(rules, by = 'lift')[1:10])

> inspect(sort(rules, by = 'lift')[1:10])

lhs rhs support confidence lift count

[1] {light cream} => {chicken} 0.004532729 0.2905983 4.843951 34

[2] {pasta} => {escalope} 0.005865885 0.3728814 4.700812 44

[3] {pasta} => {shrimp} 0.005065991 0.3220339 4.506672 38

[4] {eggs,ground beef} => {herb & pepper} 0.004132782 0.2066667 4.178455 31

[5] {whole wheat pasta} => {olive oil} 0.007998933 0.2714932 4.122410 60

[6] {herb & pepper,spaghetti} => {ground beef} 0.006399147 0.3934426 4.004360 48

[7] {herb & pepper,mineral water} => {ground beef} 0.006665778 0.3906250 3.975683 50

[8] {tomato sauce} => {ground beef} 0.005332622 0.3773585 3.840659 40

[9] {mushroom cream sauce} => {escalope} 0.005732569 0.3006993 3.790833 43

[10] {frozen vegetables,mineral water,spaghetti} => {ground beef} 0.004399413 0.3666667 3.731841 33

De la tabla, apreciamos las 10 mejores asociaciones de items en cuanto al lifting más alto obtenido, los niveles de soporte varían bastante (support) y la confianza (confiance) indica que tanto porcentaje de las canastas de compra se generan cuando existe compras de los ítems de la izquierda (lhs) y luego se compra los de la derecha (rhs), finalmente, la cantidad de elementos comprados (count) puede también ayudar a tomar decisiones para la futura distribución de productos.

**Objetivo 3:**

La técnica de K-Fold-Cross-Validation nos permite mejorar el rendimiento de los modelos predictivos a través de la generación de múltiples divisiones del conjunto de entrenamiento en dos partes que se generan aleatoriamente K veces, la primera parte (la más grande, con alrededor del 80% o 90% de los datos) sirve para entrenar el modelo y la segunda parte (la más pequeña, con alrededor del 20% o 10% de los datos) sirve para validar el modelo, para que al finalizar las K iteraciones, se promedien los resultados y así se genere una predicción más adecuada con los datos.

**# k-fold cross validation**

**# Importar el dataset**

dataset = read.csv('Social\_Network\_Ads.csv')

dataset = dataset[, 3:5]

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

# install.packages("caTools")

library(caTools)

set.seed(123)

split = sample.split(dataset$Purchased, SplitRatio = 0.75)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

training\_set[,1:2] = scale(training\_set[,1:2])

testing\_set[,1:2] = scale(testing\_set[,1:2])

**# Ajustar el clasificador con el conjunto de entrenamiento.**

#install.packages("e1071")

library(e1071)

classifier = svm(formula = Purchased ~ .,

data = training\_set,

type = "C-classification",

kernel = "radial")

**# Predicción de los resultados con el conjunto de testing**

y\_pred = predict(classifier, newdata = testing\_set[,-3])

**# Crear la matriz de confusión**

cm = table(testing\_set[, 3], y\_pred)

> cm

y\_pred

0 1

0 58 6

1 4 32

**# Ratio de clasificación**

ratio= sum(diag(cm))/sum(cm)

> ratio

[1] 0.9

**# Aplicar algoritmo de k-fold cross validation (veremos la mejora en la clasificación)**

# install.packages("caret")

library(caret)

folds = createFolds(training\_set$Purchased, k = 10)

cv = lapply(folds, function(x) {

training\_fold = training\_set[-x, ]

test\_fold = training\_set[x, ]

classifier = svm(formula = Purchased ~ .,

data = training\_fold,

type = "C-classification",

kernel = "radial")

y\_pred = predict(classifier, newdata = test\_fold[,-3])

cm = table(test\_fold[, 3], y\_pred)

accuracy = (cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])

return(accuracy)

})

> cv

$Fold01

[1] 0.9333333

$Fold02

[1] 0.9333333

$Fold03

[1] 1

$Fold04

[1] 0.8666667

$Fold05

[1] 0.9

$Fold06

[1] 0.9666667

$Fold07

[1] 0.9

$Fold08

[1] 0.9333333

$Fold09

[1] 0.9333333

$Fold10

[1] 0.7666667

accuracy = mean(as.numeric(cv))

> accuracy

[1] 0.9133333

Vemos como ha mejorado nuestro modelo siendo ahora de 91.33% la clasificación correcta y antes de aplicar el K-Fold-Cross-Validation fue de 90.00%.

accuracy\_sd = sd(as.numeric(cv))

> accuracy\_sd

[1] 0.06324555

**# Visualización del conjunto de testing**

set = testing\_set

X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)

X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid\_set = expand.grid(X1, X2)

colnames(grid\_set) = c('Age', 'EstimatedSalary')

y\_grid = predict(classifier, newdata = grid\_set)

plot(set[, -3],

main = 'SVM Kernel (Conjunto de Testing)',

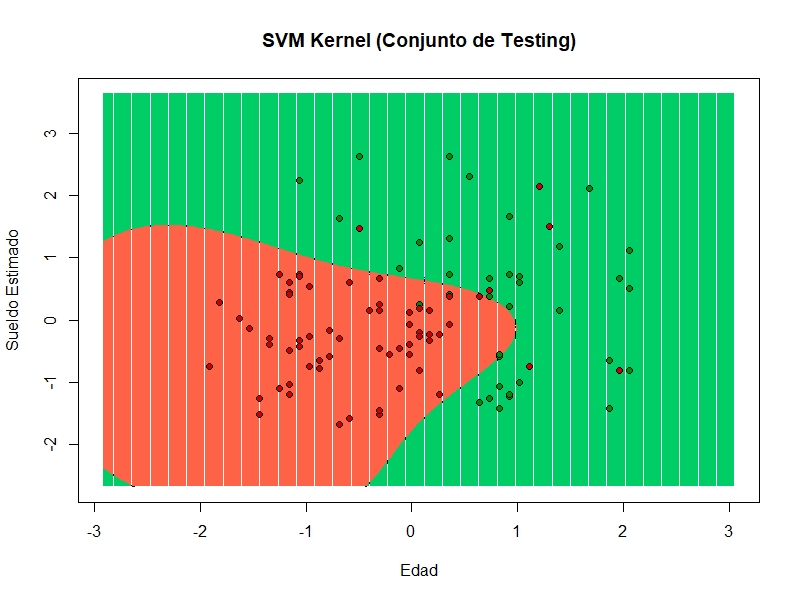
xlab = 'Edad', ylab = 'Sueldo Estimado',

xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y\_grid), length(X1), length(X2)), add = TRUE)

points(grid\_set, pch = '.', col = ifelse(y\_grid == 1, 'springgreen3', 'tomato'))

points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))



Del gráfico, apreciamos las dos regiones de clasificación, de rojo la zona de compra, y de verde la zona de no compra. Los puntos tienen colores rojo y verde también, siendo los comprados los rojos y no comprados los verdes. Existen puntos verdes en la zona roja y puntos rojos en la zona verde, estos son los datos mal clasificados según la matriz de confusión final.

**Objetivo 4:**

La técnica de grid search optimiza los modelos de predicción tras ubicar los hiperparámetros ideales a través de varias iteraciones, estos son aquellos argumentos en las funciones de clasificación que nosotros indicamos como diseñadores de los algoritmos, esto permitirá que en alguna iteración “X” el algoritmo converja al lograr el mayor porcentaje correcto en la predicción.

**# Grid Search**

**# Importar el dataset**

dataset = read.csv('Social\_Network\_Ads.csv')

dataset = dataset[, 3:5]

dataset$Purchased = factor(dataset$Purchased)

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

# install.packages("caTools")

library(caTools)

set.seed(123)

split = sample.split(dataset$Purchased, SplitRatio = 0.75)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Escalado de valores**

training\_set[,1:2] = scale(training\_set[,1:2])

testing\_set[,1:2] = scale(testing\_set[,1:2])

**# Ajustar el clasificador con el conjunto de entrenamiento.**

#install.packages("e1071")

library(e1071)

classifier = svm(formula = Purchased ~ .,

data = training\_set,

type = "C-classification",

kernel = "radial")

**# Predicción de los resultados con el conjunto de testing**

y\_pred = predict(classifier, newdata = testing\_set[,-3])

**# Crear la matriz de confusión**

cm = table(testing\_set[, 3], y\_pred)

**# Ratio de clasificación**

ratio= sum(diag(cm))/sum(cm)

> ratio

[1] 0.9

**# Aplicar Grid Search para encontrar los parámetros óptimos**

#install.packages("caret")

library(caret)

classifier = train(form = Purchased ~ .,

data = training\_set, method = 'svmRadial')

classifier

> classifier

Support Vector Machines with Radial Basis Function Kernel

300 samples

2 predictor

2 classes: '0', '1'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 300, 300, 300, 300, 300, 300, ...

Resampling results across tuning parameters:

C Accuracy Kappa

0.25 0.9139188 0.8168631

0.50 0.9142551 0.8173176

**1.00 0.9147648 0.8182548**

Tuning parameter 'sigma' was held constant at a value of 1.433809

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were **sigma = 1.433809 and C = 1**.

Vemos como ha mejorado nuestro modelo siendo ahora de 91.47% la clasificación correcta y antes de aplicar el Grid-Search fue de 90.00%.

classifier$bestTune

> classifier$bestTune

sigma C

3 1.433809 1

**# Visualización del conjunto de testing**

set = testing\_set

X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)

X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid\_set = expand.grid(X1, X2)

colnames(grid\_set) = c('Age', 'EstimatedSalary')

y\_grid = predict(classifier, newdata = grid\_set)

plot(set[, -3],

main = 'SVM Kernel (Conjunto de Testing)',

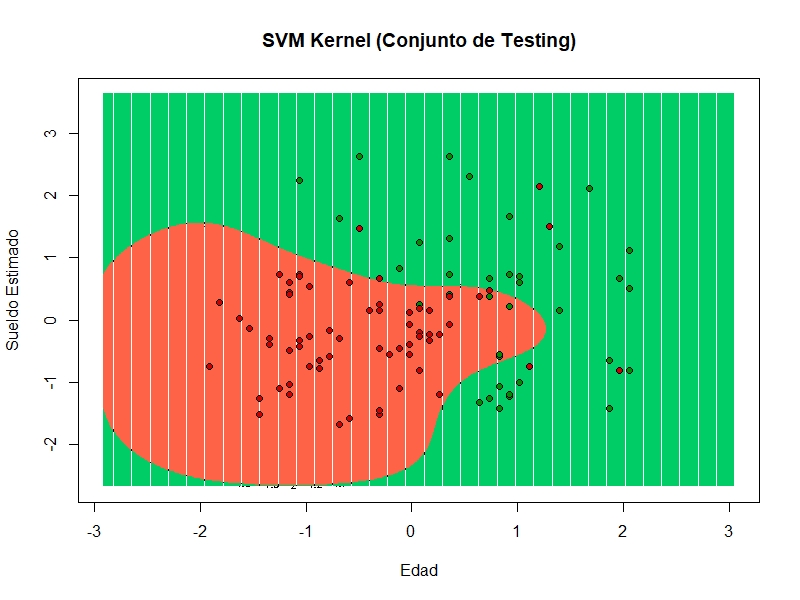
xlab = 'Edad', ylab = 'Sueldo Estimado',

xlim = range(X1), ylim = range(X2))

contour(X1, X2, matrix(as.numeric(y\_grid), length(X1), length(X2)), add = TRUE)

points(grid\_set, pch = '.', col = ifelse(y\_grid == 1, 'springgreen3', 'tomato'))

points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))

****

Del gráfico, apreciamos las dos regiones de clasificación, de rojo la zona de compra, y de verde la zona de no compra. Los puntos tienen colores rojo y verde también, siendo los comprados los rojos y no comprados los verdes. Existen puntos verdes en la zona roja y puntos rojos en la zona verde, estos son los datos mal clasificados según la matriz de confusión final.

**Objetivo 5:**

Cuando tienes una predicción que deseas mejorar para alguna de las variables respuesta en cuanto a aumentar su porcentaje de clasificación correcta puedes usar la técnica que emplearemos en esta parte del laboratorio. Si queremos saber que personas adquirirán un producto y que personas no, obviamente nos interesa clasificar correctamente más a la clase “compra” que la clase “no compra”, pudiéndose sacrificar la segunda para que mejore la primera.

**# Mejora del rendimiento de modelos predictivos cuando existe desequilibrio de clases**

**# Importar el dataset**

data <- read.csv("binary.csv", header = TRUE)

str(data)

> str(data)

'data.frame': 400 obs. of 4 variables:

$ admit: int 0 1 1 1 0 1 1 0 1 0 ...

$ gre : int 380 660 800 640 520 760 560 400 540 700 ...

$ gpa : num 3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...

$ rank : int 3 3 1 4 4 2 1 2 3 2 ...

De la tabla, debemos cambiar a factor la variable respuesta “admit” para poder realizar nuestra clasificación inicial.

data$admit <- as.factor(data$admit)

summary(data)

> summary(data)

admit gre gpa rank

0:273 Min. :220.0 Min. :2.260 Min. :1.000

1:127 1st Qu.:520.0 1st Qu.:3.130 1st Qu.:2.000

Median :580.0 Median :3.395 Median :2.000

Mean :587.7 Mean :3.390 Mean :2.485

3rd Qu.:660.0 3rd Qu.:3.670 3rd Qu.:3.000

Max. :800.0 Max. :4.000 Max. :4.000

De la tabla, vemos los principals estadísticos de las variables independientes (min, max, mean, median, etc) y apreciamos que existen más estudiantes que no obtuvieron el ingreso (0) que los que sí lo lograron (1).

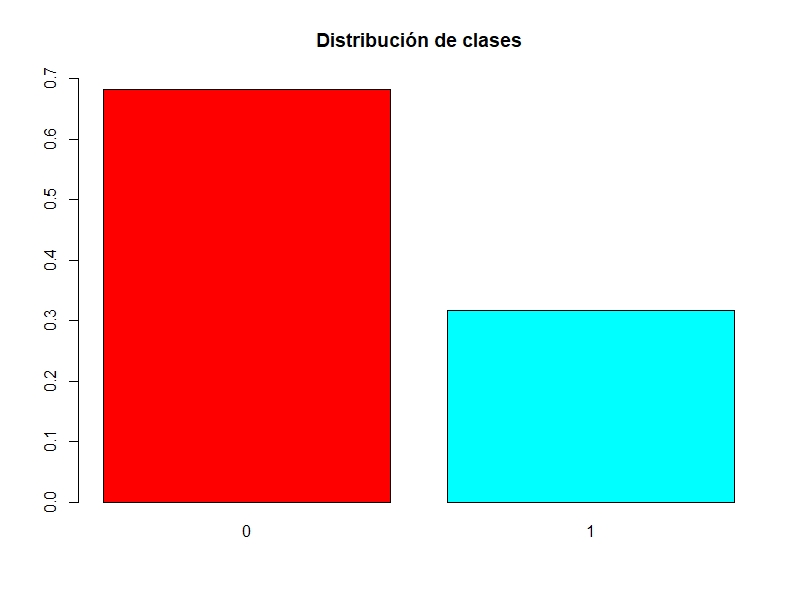
**# Gráfico de barras de la proporción de adminitidos y no admitidos**

barplot(prop.table(table(data$admit)),

col = rainbow(2),

ylim = c(0, 0.7),

main = "Distribución de clases")



Del gráfico, apreciamos que casi el 70% de alumnos no han logrado el ingreso a la universidad, el restante porcentaje es de los que sí lo lograron.

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

set.seed(123)

ind <- sample(2, nrow(data), replace = TRUE, prob = c(0.7, 0.3))

train <- data[ind==1,]

test <- data[ind==2,]

**# Desarrollar el modelo predictivo con los datos de entrenamiento**

table(train$admit)

prop.table(table(train$admit))

> prop.table(table(train$admit))

0 1

0.68125 0.31875

Se mantienen aproximadamente los porcentajes de admitidos (1) y no admitidos (0).

summary(train)

> summary(train)

admit gre gpa rank

0:218 Min. :300.0 Min. :2.420 Min. :1.000

1:102 1st Qu.:520.0 1st Qu.:3.130 1st Qu.:2.000

Median :580.0 Median :3.395 Median :2.000

Mean :588.6 Mean :3.391 Mean :2.456

3rd Qu.:660.0 3rd Qu.:3.663 3rd Qu.:3.000

Max. :800.0 Max. :4.000 Max. :4.000

Los estadísticos principales de las variables independientes también están aproximados a los del conjunto total.

library(ROSE)

**# Desarrollar el modelo con over-sampling (sobremuestreo)**

# Igualar la cantidad de elementos de clase (1) a los de la clase (0)

over <- ovun.sample(admit~., data = train, method = "over", N = 376)$data

table(over$admit)

> table(over$admit)

0 1

188 188

summary(over)

> summary(over)

admit gre gpa rank

0:188 Min. :220.0 Min. :2.260 Min. :1.000

1:188 1st Qu.:520.0 1st Qu.:3.138 1st Qu.:2.000

Median :590.0 Median :3.455 Median :2.000

Mean :590.6 Mean :3.416 Mean :2.415

3rd Qu.:660.0 3rd Qu.:3.700 3rd Qu.:3.000

Max. :800.0 Max. :4.000 Max. :4.000

Los estadísticos principales (media y mediana) de las variables independientes están aproximados a los del conjunto total, no se han movido mucho respecto al conjunto normal de entrenamiento.

**# Desarrollar el modelo con under-sampling (submuestreo)**

# Igualar la cantidad de elementos de clase (0) a los de la clase (1)

under <- ovun.sample(admit~., data=train, method = "under", N = 194)$data

table(under$admit)

> table(under$admit)

0 1

97 97

summary(under)

> summary(under)

admit gre gpa rank

0:97 Min. :300.0 Min. :2.260 Min. :1.000

1:97 1st Qu.:520.0 1st Qu.:3.170 1st Qu.:2.000

Median :580.0 Median :3.460 Median :2.000

Mean :591.3 Mean :3.429 Mean :2.402

3rd Qu.:660.0 3rd Qu.:3.708 3rd Qu.:3.000

Max. :800.0 Max. :4.000 Max. :4.000

Los estadísticos principales (media y mediana) de las variables independientes están aproximados a los del conjunto total, no se han movido mucho respecto al conjunto normal de entrenamiento.

**# Desarrollar el modelo con booth-sampling (sobremuestro y submuestreo a la vez)**

# Intenta igualar para arriba y para abajo las dos clases

both <- ovun.sample(admit~., data=train, method = "both", p = 0.5, seed = 222, N = 285)$data

table(both$admit)

> table(both$admit)

0 1

134 151

summary(both)

> summary(both)

admit gre gpa rank

0:134 Min. :220.0 Min. :2.420 Min. :1.000

1:151 1st Qu.:520.0 1st Qu.:3.130 1st Qu.:2.000

Median :600.0 Median :3.460 Median :2.000

Mean :591.7 Mean :3.402 Mean :2.463

3rd Qu.:660.0 3rd Qu.:3.670 3rd Qu.:3.000

Max. :800.0 Max. :4.000 Max. :4.000

Los estadísticos principales (media y mediana) de las variables independientes están aproximados a los del conjunto total, no se han movido mucho respecto al conjunto normal de entrenamiento.

**# Desarrollar el modelo con rose-sampling y data sintética aleatoria**

# Intenta igualar para arriba y para abajo las dos clases con datos adicionales en las clases

rose <- ROSE(admit~., data = train, N = 500, seed=111)$data

table(rose$admit)

> table(rose$admit)

0 1

234 266

summary(rose)

> summary(rose)

admit gre gpa rank

0:234 Min. :130.0 Min. :2.186 Min. :-0.6079

1:266 1st Qu.:502.9 1st Qu.:3.127 1st Qu.: 1.5553

Median :587.0 Median :3.401 Median : 2.3204

Mean :589.7 Mean :3.389 Mean : 2.3655

3rd Qu.:684.2 3rd Qu.:3.673 3rd Qu.: 3.1457

Max. :887.2 Max. :4.595 Max. : 4.9871

Los estadísticos principales (media y mediana) de las variables independientes están aproximados a los del conjunto total, pero los valores (ver mín y máx) no tienen sentido, no puede haber -0.6079 de ranking o 4.9871 de ranking por ejemplo. Sin embargo, en otro tipo de datasets esta técnica puede ser valiosa, así que dejamos para el análisis posterior esta técnica.

**# Generar modelo de clasificación (Random Forest)**

library(randomForest)

rftrain <- randomForest(admit~., data = train)

rfover <- randomForest(admit~., data = over)

rfunder <- randomForest(admit~., data=under)

rfboth <-randomForest(admit~., data=both)

rfrose <- randomForest(admit~., data=rose)

**# Generar matrices de confusion para los diferentes modelos realizados**

library(caret)

library(e1071)

confusionMatrix(predict(rftrain, test), test$admit, positive = '1')

confusionMatrix(predict(rfover, test), test$admit, positive = '1')

confusionMatrix(predict(rfunder, test), test$admit, positive = '1')

confusionMatrix(predict(rfboth, test), test$admit, positive = '1')

confusionMatrix(predict(rfrose, test), test$admit, positive = '1')

> confusionMatrix(predict(rftrain, test), test$admit, positive = '1')

Confusion Matrix and Statistics

Reference

Prediction 0 1

0 69 22

1 16 8

**Accuracy : 0.6696**

95% CI : (0.5757, 0.7544)

No Information Rate : 0.7391

P-Value [Acc > NIR] : 0.9619

Kappa : 0.0839

Mcnemar's Test P-Value : 0.4173

**Sensitivity : 0.26667**

**Specificity : 0.81176**

Pos Pred Value : 0.33333

Neg Pred Value : 0.75824

Prevalence : 0.26087

Detection Rate : 0.06957

Detection Prevalence : 0.20870

Balanced Accuracy : 0.53922

'Positive' Class : 1

> confusionMatrix(predict(rfover, test), test$admit, positive = '1')

Confusion Matrix and Statistics

Reference

Prediction 0 1

0 56 14

1 29 16

**Accuracy : 0.6261**

95% CI : (0.531, 0.7145)

No Information Rate : 0.7391

P-Value [Acc > NIR] : 0.99720

Kappa : 0.1654

Mcnemar's Test P-Value : 0.03276

**Sensitivity : 0.5333**

**Specificity : 0.6588**

Pos Pred Value : 0.3556

Neg Pred Value : 0.8000

Prevalence : 0.2609

Detection Rate : 0.1391

Detection Prevalence : 0.3913

Balanced Accuracy : 0.5961

'Positive' Class : 1

> confusionMatrix(predict(rfunder, test), test$admit, positive = '1')

Confusion Matrix and Statistics

Reference

Prediction 0 1

0 45 11

1 40 19

**Accuracy : 0.5565**

95% CI : (0.4609, 0.6491)

No Information Rate : 0.7391

P-Value [Acc > NIR] : 1

Kappa : 0.124

Mcnemar's Test P-Value : 8.826e-05

**Sensitivity : 0.6333**

**Specificity : 0.5294**

Pos Pred Value : 0.3220

Neg Pred Value : 0.8036

Prevalence : 0.2609

Detection Rate : 0.1652

Detection Prevalence : 0.5130

Balanced Accuracy : 0.5814

'Positive' Class : 1

> confusionMatrix(predict(rfboth, test), test$admit, positive = '1')

Confusion Matrix and Statistics

Reference

Prediction 0 1

0 38 9

1 47 21

**Accuracy : 0.513**

95% CI : (0.4181, 0.6073)

No Information Rate : 0.7391

P-Value [Acc > NIR] : 1

Kappa : 0.1043

Mcnemar's Test P-Value : 7.641e-07

**Sensitivity : 0.7000**

**Specificity : 0.4471**

Pos Pred Value : 0.3088

Neg Pred Value : 0.8085

Prevalence : 0.2609

Detection Rate : 0.1826

Detection Prevalence : 0.5913

Balanced Accuracy : 0.5735

'Positive' Class : 1

> confusionMatrix(predict(rfrose, test), test$admit, positive = '1')

Confusion Matrix and Statistics

Reference

Prediction 0 1

0 38 14

1 47 16

**Accuracy : 0.4696**

95% CI : (0.3759, 0.5649)

No Information Rate : 0.7391

P-Value [Acc > NIR] : 1

Kappa : -0.0145

Mcnemar's Test P-Value : 4.182e-05

**Sensitivity : 0.5333**

**Specificity : 0.4471**

Pos Pred Value : 0.2540

Neg Pred Value : 0.7308

Prevalence : 0.2609

Detection Rate : 0.1391

Detection Prevalence : 0.5478

Balanced Accuracy : 0.4902

'Positive' Class : 1

De las tablas anteriores, concluimos que el mejor modelo de Random Forest se da con both-sampling, pues nos interesa tener la mayor cantidad de aceptados (1) a la universidad.

**Objetivo 6:**

Para desarrollar nuestro modelo de XGBoost utilizaremos un dataset que muestra los datos de un banco donde la variable dependiente es la última columna “Exited” la cual tiene 0 para cuando el cliente del banco no se ha ido y un 1 si este se fue del mismo. Las variables independientes que nos interesan son de la columna 4 en adelante, ya que las demás no aportarán al desarrollo del modelo.

**# XGBoost**

**# Importar el dataset**

dataset = read.csv('Churn\_Modelling.csv')

dataset = dataset[, 4:14]

str(dataset)

**# Codificar los factores para la RNA**

dataset$Geography = as.numeric(factor(dataset$Geography,

levels = c("France", "Spain", "Germany"),

labels = c(1, 2, 3)))

dataset$Gender = as.numeric(factor(dataset$Gender,

levels = c("Female", "Male"),

labels = c(1,2)))

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

# install.packages("caTools")

library(caTools)

set.seed(123)

split = sample.split(dataset$Exited, SplitRatio = 0.8)

training\_set = subset(dataset, split == TRUE)

testing\_set = subset(dataset, split == FALSE)

**# Aplicar algoritmo de k-fold cross validation**

# install.packages("caret")

library(caret)

folds = createFolds(training\_set$Exited, k = 10)

**# Parámetros comunes de XGBoost:**

# objective = "binary:logistic": tipo de modelo a entrenar

# max.depth = 2: Profundidad de los árboles de clasificacion (cuantar ramas tendrà el arbol)

# nthread = 2: número de cpus empleadas para el algoritmo (depende de cuantos cores exista)

# nrounds = 2: Número máximo de iteraciones de refuerzo.(mientras màs rondas mejorará)

# verbose =1 : La opción 1 hace que devuelva información del rendimiento del modelo

cv = lapply(folds, function(x) {

training\_fold = training\_set[-x, ]

test\_fold = training\_set[x, ]

classifier = xgboost(data = as.matrix(training\_set[, -11]),

label = training\_set$Exited,

nrounds = 100,

objective = "binary:logistic",

max.depth = 5,

nthread = 2,

verbose = 1)

y\_pred = predict(classifier, newdata = as.matrix(test\_fold[,-11]))

y\_pred = (y\_pred >= 0.5)

cm = table(test\_fold[, 11], y\_pred)

accuracy = (cm[1,1]+cm[2,2])/(cm[1,1]+cm[1,2]+cm[2,1]+cm[2,2])

return(accuracy)

})

> accuracy

[1] 0.92875

> accuracy\_sd

[1] 0.01005195

Concluimos que, con el modelo de XGBoost obtenemos un total de aciertos del 92.88% en la clasificación de los clientes con una desviación estándar de la misma de 1.00%.