Machine Learning con R y caret

Manuel Rain

07-11-2020

Contents

Introducción

Durante los últimos años, el interés y la aplicación de machine learning ha experimentado tal expansión, que se ha convertido en una disciplina aplicada en prácticamente todos los ámbitos de investigación académica e industrial.

El término machine learning engloba al conjunto de algoritmos que permiten identificar patrones presentes en los datos y crear con ellos estructuras (modelos) que los representan.

A diferencia de otros documentos, este pretende ser un ejemplo práctico con menos desarrollo teórico. El lector podrá darse cuenta de lo sencillo que es aplicar un gran abanico de métodos predictivos con R y sus librerías.

Etapas de un problema de machine learning

El siguiente es un listado de las etapas que suelen formar parte de la mayoría de problemas de machine learning.

1. Definir el problema: ¿Qué se pretende predecir? ¿De qué datos se dispone? ¿Qué datos es necesario conseguir?
2. Explorar y entender los datos que se van a emplear para crear el modelo.
3. Métrica de existo: definir una forma apropiada de cuantificar cómo de buenos son los resultados obtenidos.
4. Preparar la estrategia para evaluar el modelo: separar las observaciones en un conjunto de entrenamiento, un conjunto de validación (este último suele ser un subconjunto del de entrenamiento) y un conjunto de test. Ninguna información del conjunto de test debe participar en el proceso de entrenamiento del modelo.
5. Preprocesar los datos: aplicar las transformaciones necesarias para que los datos puedan ser interpretados por el algoritmo de machine learning seleccionado.
6. Ajustar un primer modelo capaz de superar unos resultados mínimos. Por ejemplo, en problemas de clasificación, el mínimo a superar es el porcentaje de la clase mayoritaria (la moda).
7. Gradualmente, mejorar el modelo optimizando sus hiperparámetros.
8. Evaluar la capacidad del modelo final con el conjunto de test para tener una estimación de la capacidad que tiene el modelo cuando predice nuevas observaciones.

Paquete caret

R es uno de los lenguajes de programación que domina dentro del ámbito de la estadística, data mining, machine learning, entre otros. Al tratarse de un software libre, innumerables usuarios han podido implementar sus algoritmos, dando lugar a un número muy elevado de paquetes/librerías donde encontrar prácticamente todas las técnicas de machine learning existentes. Sin embargo, esto tiene un lado negativo, cada paquete tiene una sintaxis, estructura e implementación propia, lo que dificulta su aprendizaje. El paquete caret, desarrollado por Max Kuhn, es una interfaz que unifica bajo un único marco cientos de funciones de distintos paquetes, facilitando en gran medida todo el proceso de preprocesado, entrenamiento, optimización y validación de modelos predictivos. Existen otros proyectos similares y muy prometedores como mlr pero, para este ejemplo, se emplea únicamente caret.

Dataset

Este set de datos contiene información sobre un Banco de Colombia. Entre la información almacenada se encuentran la edad, género, características socio-económicas. Aunque pueda resultar un clásico poco original, estos datos tienen una serie de características que los hacen idóneos para ser utilizados como ejemplo introductorio al machine learning:

Análisis exploratorio de los datos

Antes de entrenar un modelo predictivo, o incluso antes de realizar cualquier cálculo con un nuevo conjunto de datos, es muy importante realizar una exploración descriptiva de los mismos. Este proceso permite entender

mejor que información contiene cada variable, así como detectar posibles errores. Algunos ejemplos frecuentes son:

1. Que una columna se haya almacenado con el tipo incorrecto: una variable numérica está siendo reconocida como texto.
2. Que una variable contenga valores que no tienen sentido: para indicar que no se dispone de la altura de una persona se introduce el valor cero o un espacio en blanco. No existe nadie cuya altura sea cero.
3. Que en una variable de tipo numérico se haya introducido una palabra en lugar de un número.

Además, puede dar pistas sobre qué variables no son adecuadas como predictores en un modelo (más sobre esto en los siguientes apartados).

Acorde a la información facilitada, las variables disponibles son:

1. Act. Económica: Actividad que desmpeña el cliente
2. Auto: tipo de automovil del cliente
3. Campaña: nombre comercial con el cual se oferto la trajeta
4. Canal: fuerza de venta que realizó la venta
5. Ciudad: ciudad del cliente
6. Contrato: tipo de contrato laboral del cliente
7. Cta. Ahorro (Ent): posee o no cuenta de nomina con la entidad
8. Estado civil: estado civil del cliente
9. Estrato estrato que registro el cliente
10. Franquicia: compañía con la que se creo la tarjeta
11. Genero: genero del ciente
12. Motivo Retiro: descripción de la cancelación del producto
13. Nivel Académico: nivel academico del cliente
14. Vivienda: tipo de vivienda del cliente
15. Cupo: cupo actual que tiene el cliente en la tarjeta
16. Edad: edad del cliente (al corte)
17. endeudamiento rotativo Sector: Endeudamiento global (rotativos) que tiene el cliente en el sector financiero
18. Ingresos Cliente: ingresos del cliente en el momento de la entrega de la tarjeta
19. Máxima Mora Últimos 12 meses (Sec): Máximo numero de días en los que el cliente a pagado la tarjeta posterior a la fecha limite de pago
20. Máximo Plazo Diferido: Máximo plazo en el que el cliente diferio alguna compra (Historico)
21. Personas a cargo: personas a cargo del cliente
22. Uso% 12 meses a tras: Karate
23. Score Central: puntaje central de riesgo (al corte)
24. Target: variable objetivo
25. Uso% historico: ratio de uso del cupo de la tarjeta historico

Tipo de variables

Una de las primeras comprobaciones que hay que hacer tras cargar los datos, es verificar que cada variable se ha almacenado con el tipo de valor que le corresponde, es decir, que las variables numéricas sean números y las cualitativas factor, character o booleanas. En el lenguaje de programación R, cuando la variable es cualitativa, conviene almacenarla con el tipo factor.

```
data <- read.delim("F:/sergio/taller 3/data.txt", header=T,encoding = 'UTF-8')


set.seed(345)
data2= sample(1:nrow(data),size=2000,replace=FALSE)

data<- data[data2, ]


data$Target=factor(data$Target, levels = c(0,1),
                   labels = c("No","Si"))

set.seed(345)


library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unable to identify current timezone 'H':
## please set environment variable 'TZ'
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.0.3     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
```

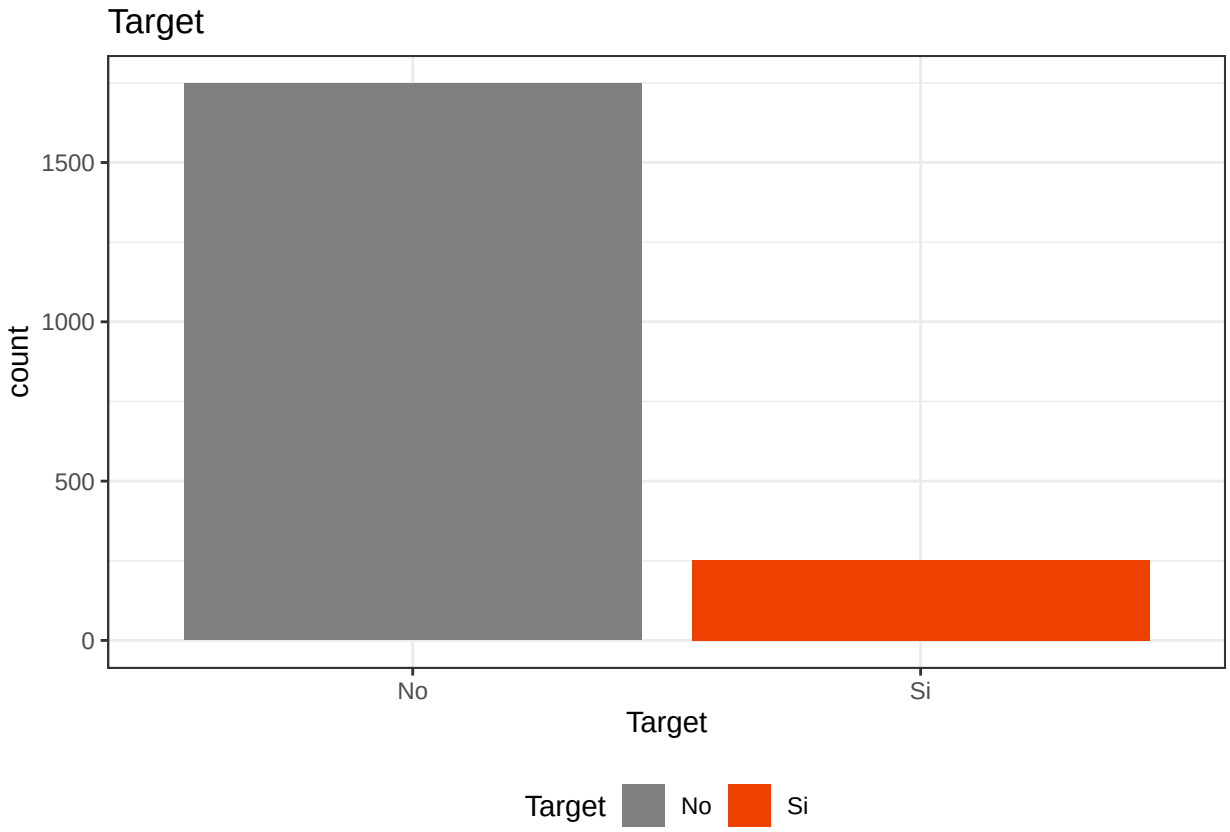
```
data$Ciudad=as.factor(data$Ciudad)
data$Estrato=as.factor(data$Estrato)
data$Canal=as.factor(data$Canal)
data$Campaña=as.factor(data$Campaña)
data$Nivel_Academico=as.factor(data$Nivel_Academico)
data$Motivo_Retiro=as.factor(data$Motivo_Retiro)
data$Franquicia=as.factor(data$Franquicia)
data$Genero=as.factor(data$Genero)
data$Vivienda=as.factor(data$Vivienda)
data$Auto=as.factor(data$Auto)
data$Contrato=as.factor(data$Contrato)
data$Act._Economica=as.factor(data$Act._Economica)
data$Estado_civil=as.factor(data$Estado_civil)
```

```r
data$Cta._Ahorro_.Ent.=as.factor(data$Cta._Ahorro_.Ent.)
```

Distribución de variables respuesta

Cuando se crea un modelo, es muy importante estudiar la distribución de la variable respuesta, ya que, a fin de cuentas, es lo que nos interesa predecir.

```r
ggplot(data = data, aes(x = Target, y = ..count.., fill = Target)) +
  geom_bar() + scale_fill_manual(values = c("gray50", "orangered2")) +
  labs(title = "Target") + theme_bw() +
  theme(legend.position = "bottom")
```




```r
prop.table(table(data$Target)) %>% round(digits = 2)
```

```
##
##   No   Si
## 0.87 0.13
```

Para que un modelo predictivo sea útil, debe de tener un porcentaje de acierto superior a lo esperado por azar o a un determinado nivel basal. En problemas de clasificación, el nivel basal es el que se obtiene si se asignan todas las observaciones a la clase mayoritaria (la moda). En nuestra data el 87% de los clientes no obtuvieron credito, si siempre se predice Target = No, el porcentaje de aciertos será aproximadamente del 87%. Este es el porcentaje mínimo que hay que intentar superar con los modelos predictivos. (Siendo estrictos, este porcentaje tendrá que ser recalculado únicamente con el conjunto de entrenamiento).

Porcentaje de aciertos si se predice para todas las observaciones

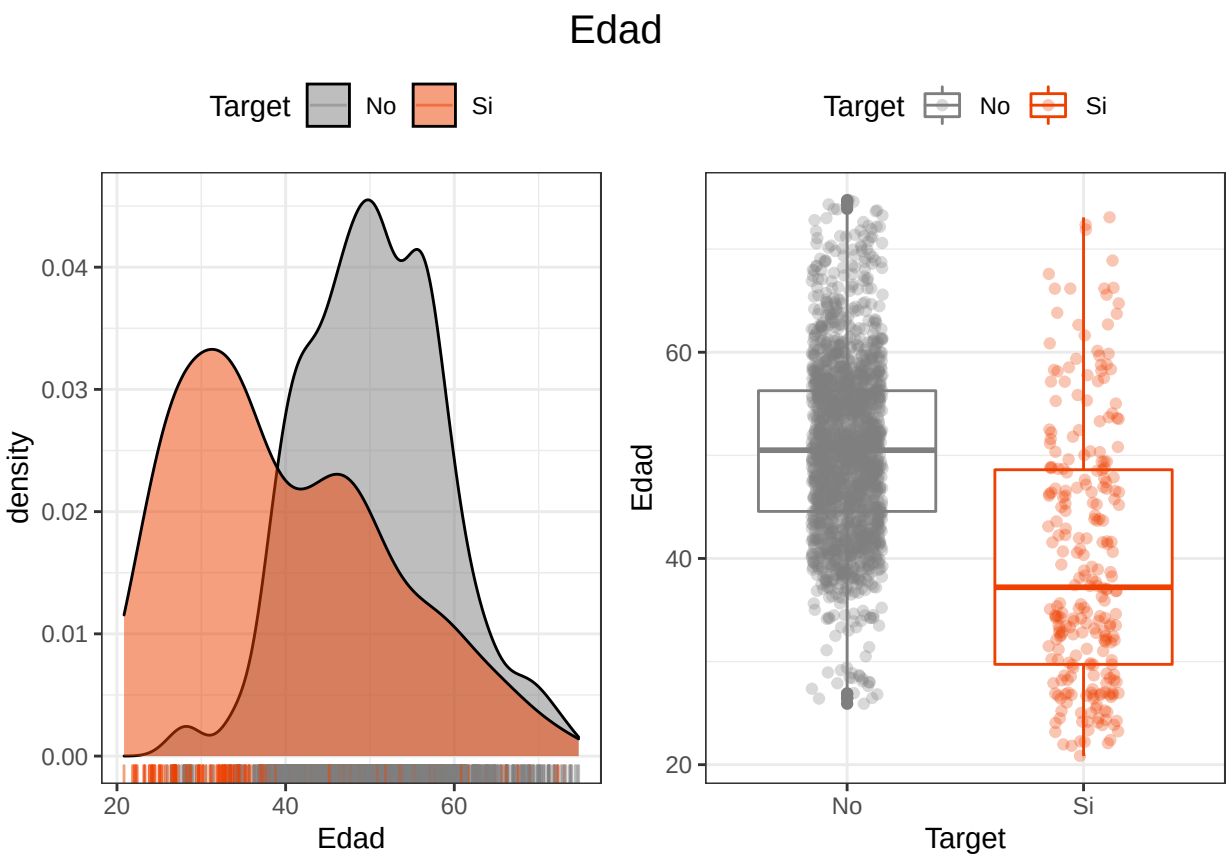
```r
n_observaciones <- nrow(data)
predicciones <- rep(x = "No", n_observaciones)
mean(predicciones == data$Target) * 100
```

```
## [1] 87.4
```

Distribución de variables continuas

Como el objetivo del estudio es predecir qué clientes si y cuáles no obtuvieron el credito, el análisis de cada variable se hace en relación a la variable respuesta Target. Analizando los datos de esta forma, se pueden empezar a extraer ideas sobre qué variables están más relacionadas con la Target.

```r
library(ggpubr)

p1 <- ggplot(data = data, aes(x = Edad, fill = Target)) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("gray50", "orangered2")) +
  geom_rug(aes(color = Target), alpha = 0.5) +
  scale_color_manual(values = c("gray50", "orangered2")) +
  theme_bw()

p2 <- ggplot(data = data,
             aes(x = Target, y = Edad, color = Target)) +
  geom_boxplot() +
  geom_jitter(alpha = 0.3, width = 0.15) +
  scale_color_manual(values = c("gray50", "orangered2")) +
  theme_bw()


final_plot <- ggarrange(p1, p2, legend = "top")
final_plot <- annotate_figure(final_plot,
                              top = text_grob("Edad", size = 15))
final_plot
```



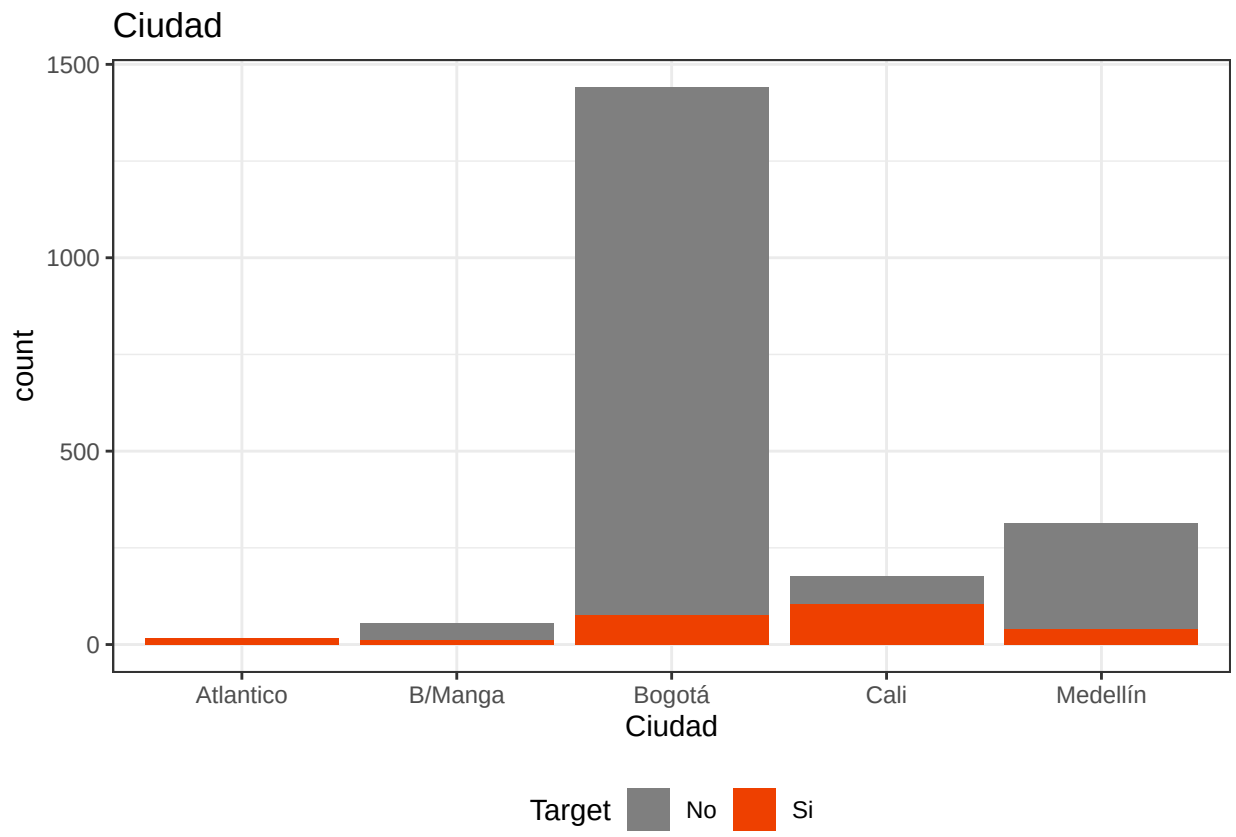

```
data %>% group_by(Target) %>%
  summarise(count=n(),
            media = mean(Edad),
            mediana = median(Edad),
            min = min(Edad),
            max = max(Edad))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 6
##   Target count media mediana   min   max
##   <fct>  <int> <dbl>   <dbl> <dbl> <dbl>
## 1 No      1748  50.6    50.5  25.9  74.8
## 2 Si       252  40.0    37.2  20.8  73.1
```

Distribución de variables cualitativas

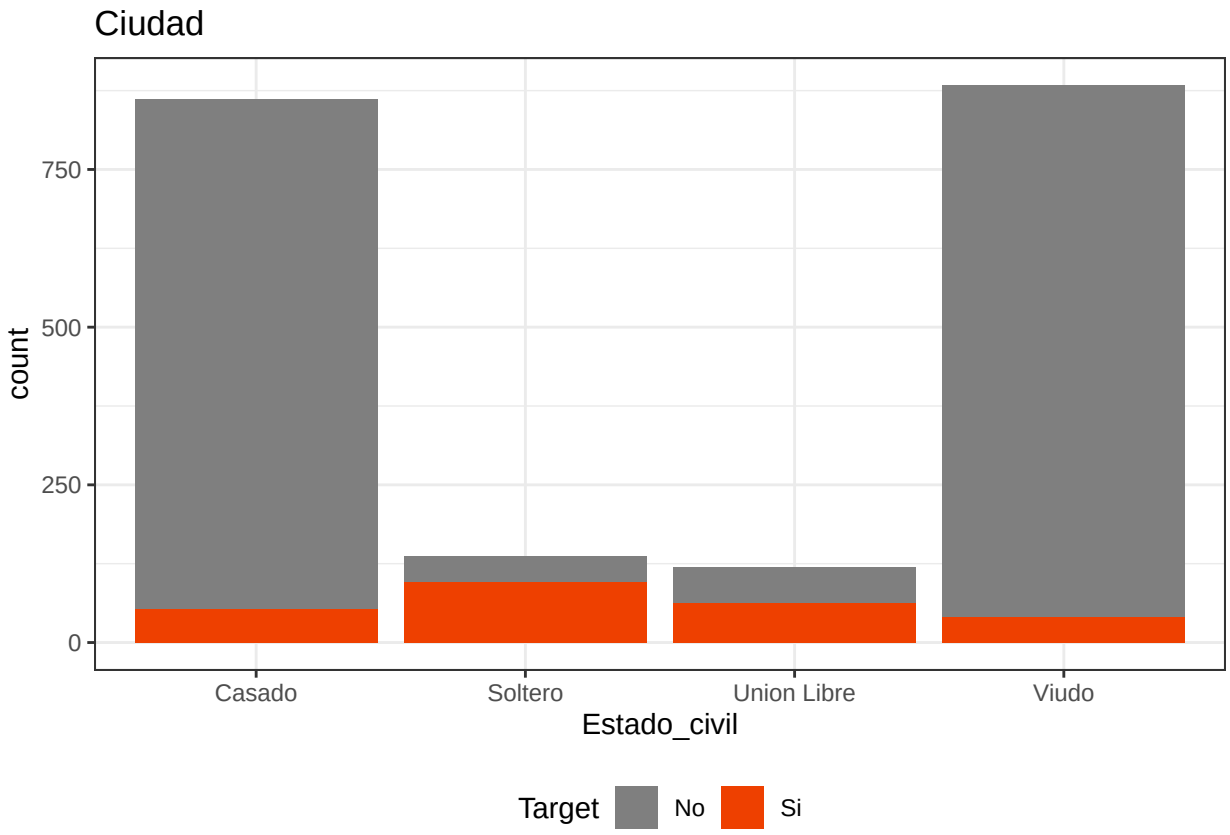
```
ggplot(data = data, aes(x =Ciudad , y = ..count.., fill = Target)) +
  geom_bar() + scale_fill_manual(values = c("gray50", "orangered2")) +
  labs(title = "Ciudad") + theme_bw() +
  theme(legend.position = "bottom")
```




```
prop.table(table(data$Ciudad, data$Target), margin = 1) %>%
  round(digits = 2)
```

```
##
##                No   Si
##   Atlantico 0.06 0.94
##   B/Manga   0.78 0.22
##   Bogotá    0.95 0.05
##   Cali      0.40 0.60
##   Medellín  0.87 0.13
```

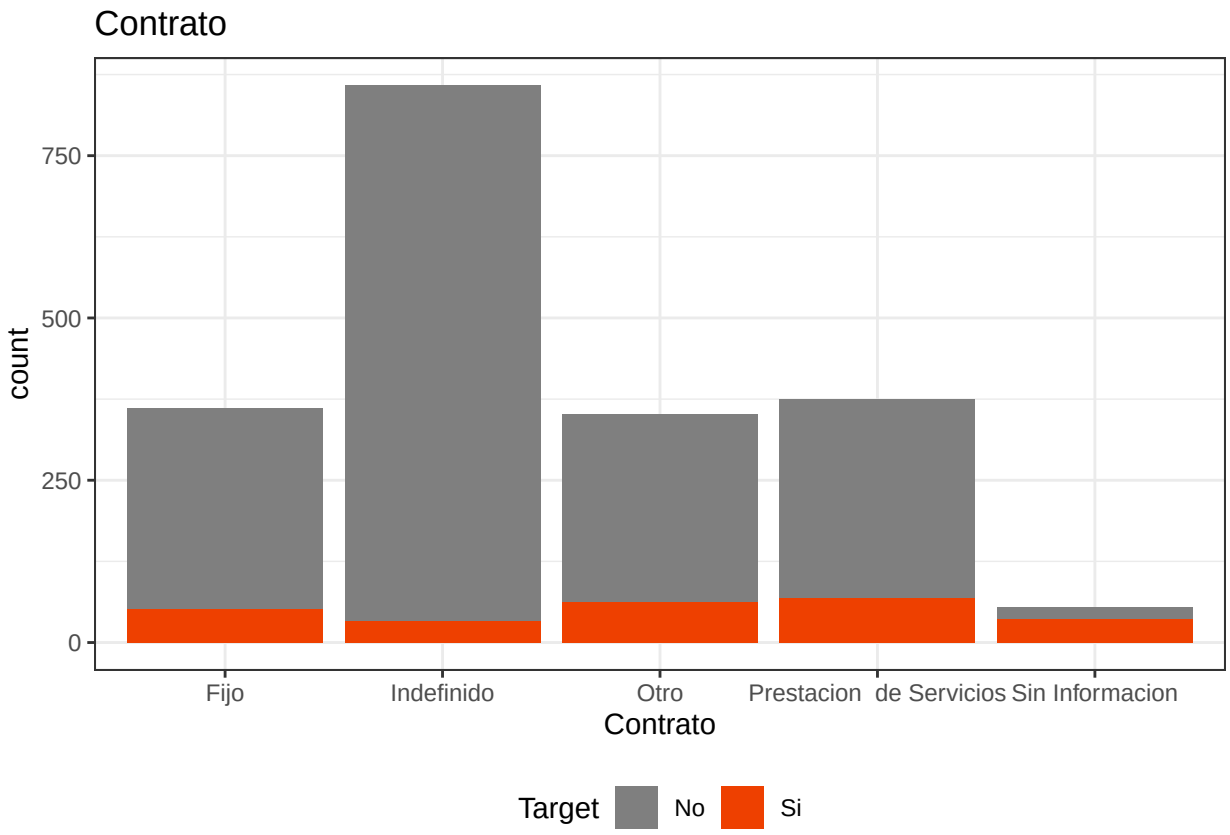
```
ggplot(data = data, aes(x =Estado_civil , y = ..count.., fill = Target)) +
  geom_bar() + scale_fill_manual(values = c("gray50", "orangered2")) +
  labs(title = "Ciudad") + theme_bw() +
  theme(legend.position = "bottom")
```




```
prop.table(table(data$Estado_civil, data$Target), margin = 1) %>%
  round(digits = 2)
```

```
##
##                  No   Si
##   Casado       0.94 0.06
##   Soltero      0.29 0.71
##   Union Libre  0.48 0.52
##   Viudo        0.95 0.05
```

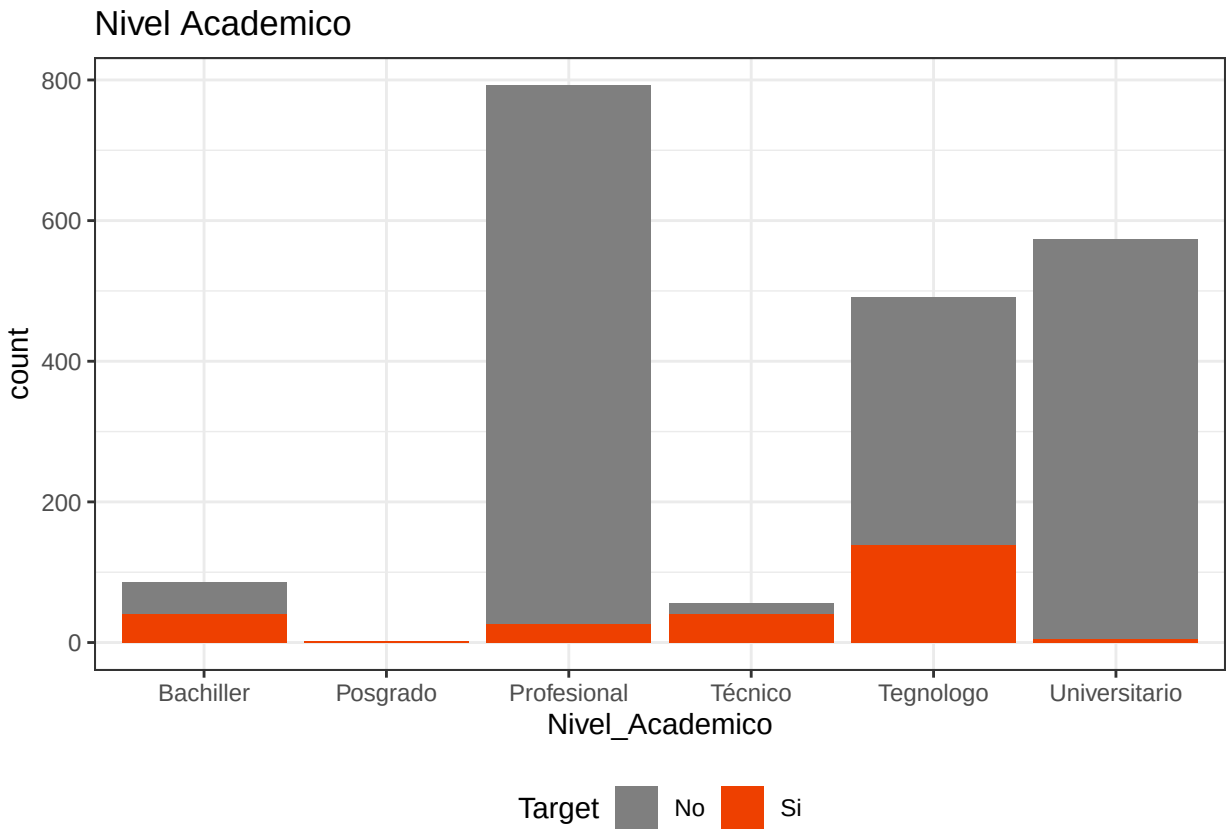
```
ggplot(data = data, aes(x =Contrato , y = ..count.., fill = Target)) +
  geom_bar() + scale_fill_manual(values = c("gray50", "orangered2")) +
  labs(title = "Contrato") + theme_bw() +
  theme(legend.position = "bottom")
```




```
prop.table(table(data$Contrato, data$Target), margin = 1) %>%
  round(digits = 2)
```

```
##
##                              No   Si
##   Fijo                     0.86 0.14
##   Indefinido               0.96 0.04
##   Otro                     0.82 0.18
##   Prestacion  de Servicios 0.82 0.18
##   Sin Informacion          0.33 0.67
```

```
ggplot(data = data, aes(x =Nivel_Academico , y = ..count.., fill = Target)) +
  geom_bar() + scale_fill_manual(values = c("gray50", "orangered2")) +
  labs(title = "Nivel Academico") + theme_bw() +
  theme(legend.position = "bottom")
```




```
prop.table(table(data$Nivel_Academico, data$Target), margin = 1) %>%
  round(digits = 2)
```

```
##
##                    No   Si
##   Bachiller      0.52 0.48
##   Posgrado       0.00 1.00
##   Profesional    0.97 0.03
##   Técnico        0.29 0.71
##   Tegnologo      0.72 0.28
##   Universitario  0.99 0.01
```

Importancia de las variables

La representación gráfica de la distribución de las variables en función de target si o no, ayuda a tener una idea de qué variables pueden ser buenos predictores para el modelo y cuales no aportan información o la que aportan es redundante. Aunque la creación de un buen modelo debe entenderse como un proceso iterativo, en el que se van ajustando y probando distintos modelos, existen ciertas pistas que pueden ayudar a realizar una selección inicial adecuada.

1. Si dos variables numéricas están muy correlacionadas, añaden información redundante al modelo, por lo tanto, no conviene incorporar ambas. Si esto ocurre, se puede: excluir aquella que, acorde al criterio del analista, no está realmente asociada con la variable respuesta; o combinarlas para recoger toda su información en una única nueva variable, por ejemplo, con un PCA.

2. Si una variable tiene varianza igual o próxima a cero (su valor es el mismo o casi el mismo para todas las observaciones) añade al modelo más ruido que información, por lo que suele ser conveniente excluirla.

3. Si alguno de los niveles de una variable cualitativa tiene muy pocas observaciones en comparación a los otros niveles, puede ocurrir que, durante la validación cruzada o bootstrapping, algunas particiones no contengan ninguna observación de dicha clase (varianza cero), lo que puede dar lugar a errores. En estos casos, suele ser conveniente eliminar las observaciones del grupo minoritario (si es una variable multiclase), eliminar la variable (si solo tiene dos niveles) o asegurar que, en la creación de las particiones, se garantice que todos los grupos estén representados en cada una de ellas.

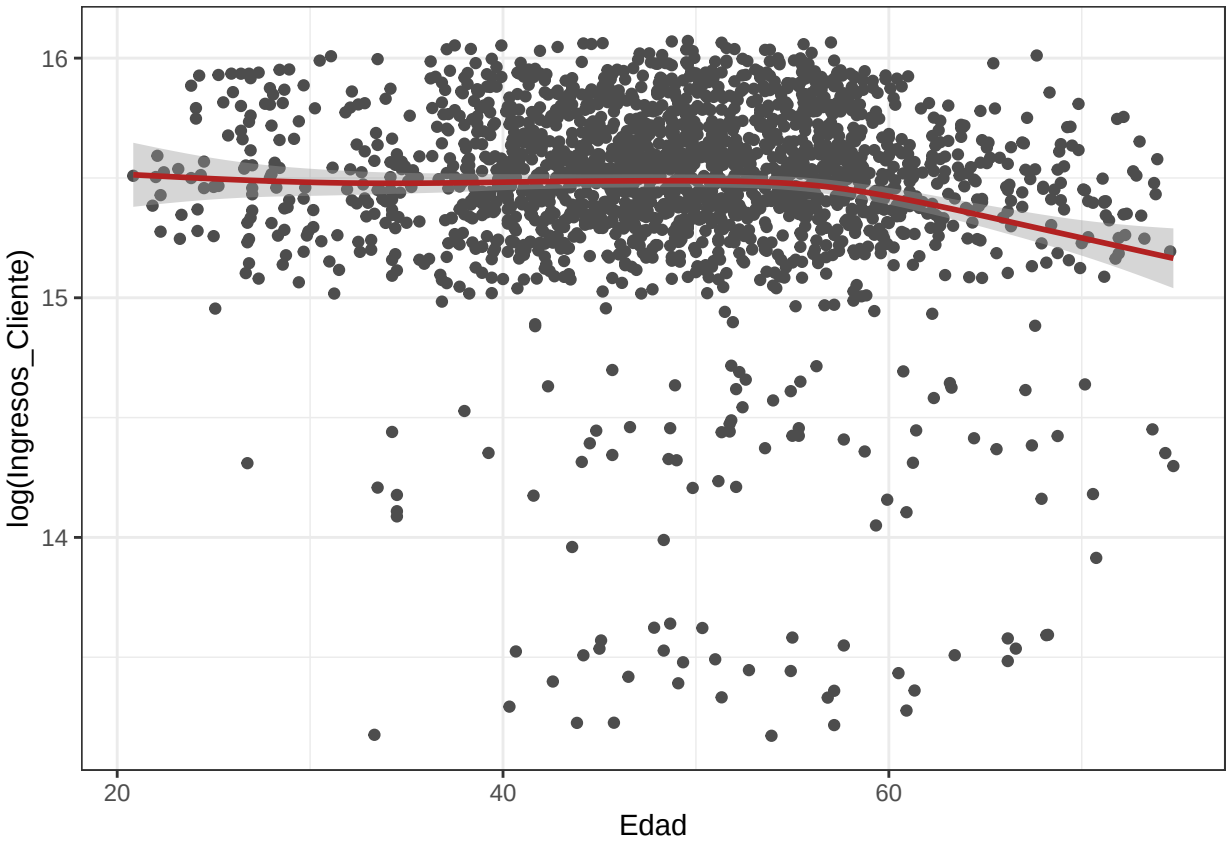
Correlación entre variables continuas

```
cor.test(x = data$Edad, y = data$Ingresos_Cliente, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  data$Edad and data$Ingresos_Cliente
## t = -3.7822, df = 1998, p-value = 0.00016
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.12767329 -0.04063341
## sample estimates:
##         cor
## -0.08431418
```

```
ggplot(data = data, aes(x = Edad, y = log(Ingresos_Cliente))) + geom_point(color = "gray30") + geom_smo
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



La correlación lineal entre la Edad y el Ingresos_Cliente, aun cuando significativa (p-value = 0.00016), es mínima (cor = -0.08). El diagrama de dispersión tampoco apunta a ningún tipo de relación no lineal evidente. Las variables no contienen información redundante.

Contraste de proporciones

Para la identificación de potenciales predictores cualitativos, es interesante encontrar las variables y niveles de las mismas que muestran una proporción de supervivientes alejada de lo esperado por el nivel basal, en este caso el 13%. Este porcentaje se corresponde con la proporción de Target respecto al total de lso clientes, es decir, el valor esperado si no existiese relación entre la variable y la supervivencia. Estas diferencias no siempre son fáciles de apreciar en una gráfica, sobre todo, cuando el número de observaciones es distinto en cada grupo.

Para facilitar este tipo de análisis, resulta útil crear variables dummy con todos los niveles de las variables cualitativas (proceso conocido como binarización o one hot encoding) y aplicar un test de contraste de proporciones.

```
datos_cualitativos <-  data %>%
  select(-Tipo_documento, -ID, -Fecha_de_Corte,
         -Fecha_nacimiento, -Fecha_activación,-Corte_AAAAMM,
         -Motivo_Retiro,-Promedio_Uso._12_meses_atras,-Cupo,
         -Ingresos_smlv,-Ingresos_Cliente,-Promedio_Uso._historico,
         -Edad,-Máximo_Plazo_Diferido,-Score_Central,-endeudamiento_rotativo_Sector,
         -Corte_AAAAMM)


datos_cualitativos_tidy <- datos_cualitativos %>%
  gather(key = "variable", value = "grupo",-Target)
```

```
## Warning: attributes are not identical across measure variables;
## they will be dropped
```

```
datos_cualitativos_tidy <- datos_cualitativos_tidy %>%
  mutate(variable_grupo = paste(variable, grupo, sep="_"))



test_proporcion <- function(df){
  n_si <- sum(df$Target == "Si")
  n_no <- sum(df$Target == "No")
  n_total <- n_si + n_no
  test <- prop.test(x = n_si, n = n_total, p = 0.13)
  prop_si <- n_si / n_total
  return(data.frame(p_value = test$p.value, prop_si))
  }


analisis_prop <- datos_cualitativos_tidy %>%
  group_by(variable_grupo) %>% nest() %>%
  arrange(variable_grupo) %>%
  mutate(prop_test = map(.x = data, .f = test_proporcion)) %>%
  unnest(prop_test) %>% arrange(p_value) %>%
  select(variable_grupo,p_value, prop_si)
```

```
## Warning in prop.test(x = n_si, n = n_total, p = 0.13): Chi-squared approximation
## may be incorrect
```

```
## Warning in prop.test(x = n_si, n = n_total, p = 0.13): Chi-squared approximation
## may be incorrect

## Warning in prop.test(x = n_si, n = n_total, p = 0.13): Chi-squared approximation
## may be incorrect

## Warning in prop.test(x = n_si, n = n_total, p = 0.13): Chi-squared approximation
## may be incorrect

## Warning in prop.test(x = n_si, n = n_total, p = 0.13): Chi-squared approximation
## may be incorrect

## Warning in prop.test(x = n_si, n = n_total, p = 0.13): Chi-squared approximation
## may be incorrect
```

```
analisis_prop
```

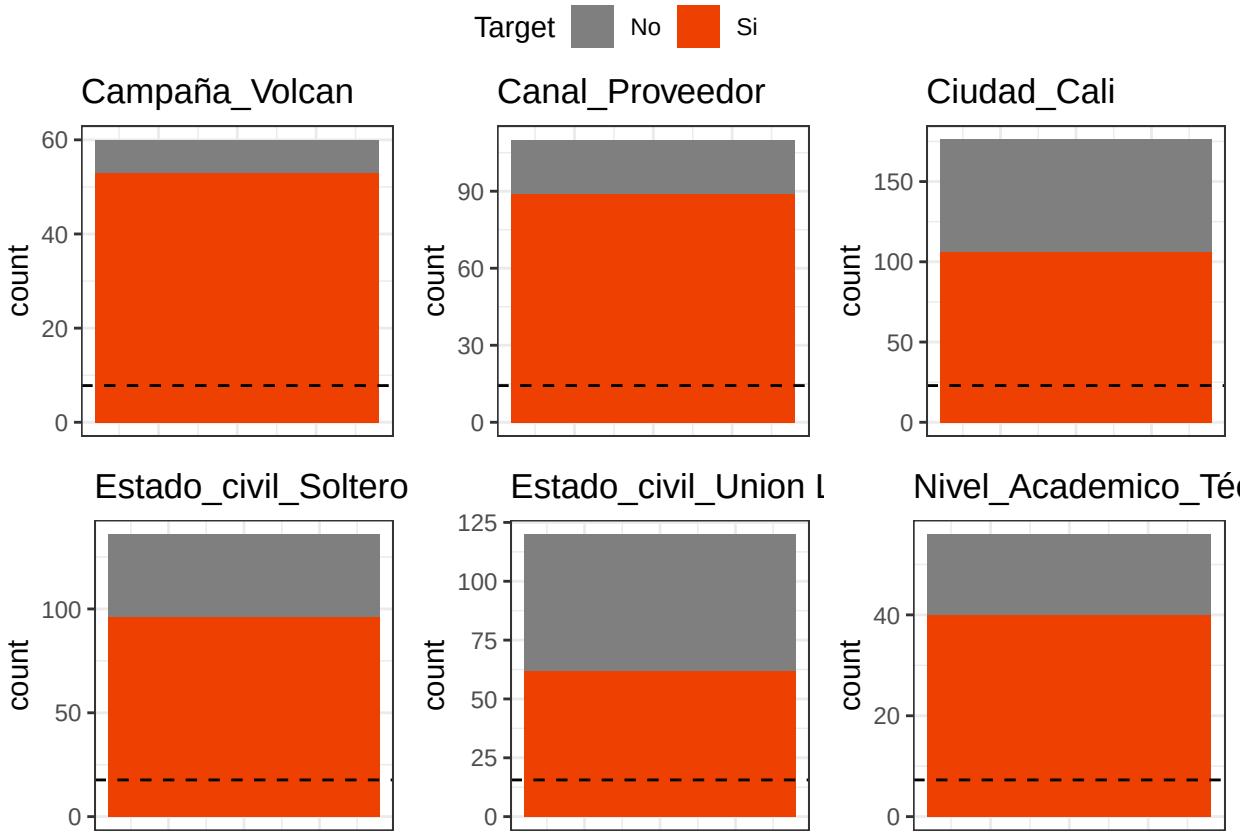
```
## # A tibble: 58 x 3
## # Groups:   variable_grupo [58]
##    variable_grupo             p_value prop_si
##    <chr>                        <dbl>   <dbl>
##  1 Canal_Proveedor           3.03e-98   0.809
##  2 Estado_civil_Soltero      1.29e-87   0.706
##  3 Ciudad_Cali               1.47e-76   0.602
##  4 Campaña_Volcan            5.35e-66   0.883
##  5 Nivel_Academico_Técnico   1.58e-37   0.714
##  6 Estado_civil_Union Libre  1.25e-35   0.517
##  7 Franquicia_Karate         9.91e-31   0.363
##  8 Contrato_Sin Informacion  9.96e-31   0.667
##  9 Nivel_Academico_Tegnologo 4.79e-23   0.281
## 10 Ciudad_Atlantico          9.29e-22   0.941
## # ... with 48 more rows
```

```
top6_grupos <- analisis_prop %>% pull(variable_grupo) %>%
  head(6)
```

Representación gráfica de la distribución de los 6 grupos con menor p-value

```
plot_grupo <- function(grupo, df, threshold_line = 0.13){
  p <- ggplot(data = df, aes(x = 1, y = ..count.., fill = Target)) +
    geom_bar() +
  scale_fill_manual(values = c("gray50", "orangered2")) +
  # Se añade una línea horizontal en el nivel basal
    geom_hline(yintercept = nrow(df) * threshold_line,
               linetype = "dashed") + labs(title = grupo) +
    theme_bw() +
    theme(legend.position = "bottom",
          axis.text.x = element_blank(),
          axis.title.x = element_blank(),
          axis.ticks.x = element_blank())
  return(p) }



datos_graficos <- datos_cualitativos_tidy %>%
```

```
  filter(variable_grupo %in% top6_grupos) %>%
  group_by(variable_grupo) %>% nest() %>%
  arrange(variable_grupo)

plots <- map2(datos_graficos$variable_grupo,
              .y = datos_graficos$data, .f = plot_grupo)

ggarrange(plotlist = plots, common.legend = TRUE)
```



El listado obtenido muestra, ordenados de menor a mayor p-value, cada uno de los posibles grupos simples en los que se puede diferenciar a los clientes.

Hay que tener precaución a la hora de interpretarlo, la idea es la siguiente: cuanto menor es el p-value de un grupo, mayor la evidencia de que la proporción de Target en dicho grupo se aleja de lo esperado (0.13), tanto por encima como por debajo, si no existiese relación entre esa variable y la supervivencia.

Random forest

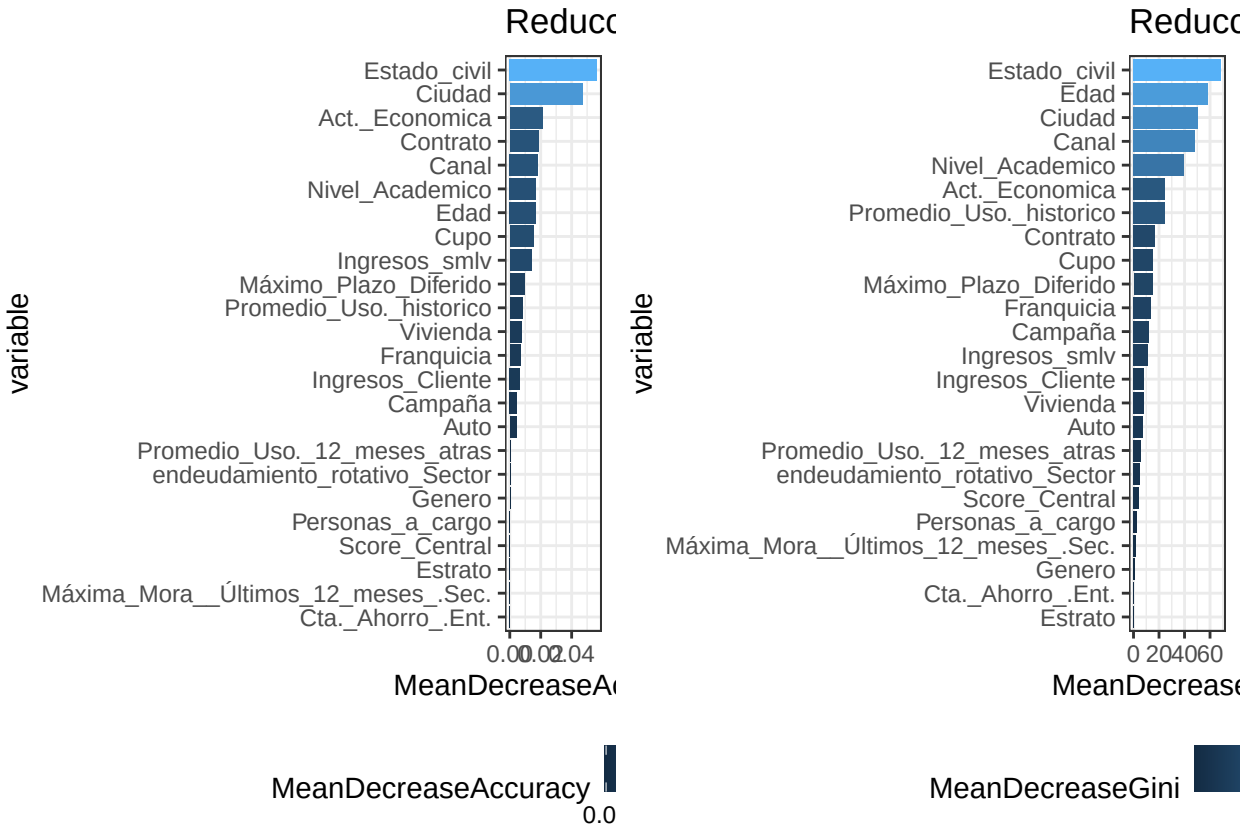
Otra estrategia ampliamente extendida para estudiar la importancia de variables es el empleo de Random Forest. El problema de aplicarlo a este ejemplo es que no acepta valores ausentes, tema que todavía no se ha tratado (visto más adelante). Por el momento, se excluyen las variables incompletas. Para información detallada sobre los fundamentos de este método, leer la sección Importancia de los predictores del capítulo Árboles de predicción.

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
datos_rf <- data %>% select(-Tipo_documento, -ID, -Fecha_de_Corte,
 -Fecha_nacimiento, -Fecha_activación,-Corte_AAAAMM,-Motivo_Retiro)



datos_rf <- map_if(.x = datos_rf, .p = is.character, .f = as.factor) %>%
  as.data.frame()

modelo_randforest <- randomForest(formula = Target ~ . ,
                                  data =datos_rf,
                                  mtry = 5, importance = TRUE,
                                  ntree = 100)

importancia <- as.data.frame(modelo_randforest$importance)
importancia <- rownames_to_column(importancia,var = "variable")

p1 <- ggplot(data = importancia,
             aes(x = reorder(variable, MeanDecreaseAccuracy),
                 y = MeanDecreaseAccuracy,
                 fill = MeanDecreaseAccuracy)) +
  labs(x = "variable", title = "Reducción de Accuracy") +
  geom_col() + coord_flip() + theme_bw() +theme(legend.position = "bottom")

p2 <- ggplot(data = importancia,
             aes(x = reorder(variable, MeanDecreaseGini),
                 y = MeanDecreaseGini, fill = MeanDecreaseGini)) +
  labs(x = "variable", title = "Reducción de pureza (Gini)") +
  geom_col() + coord_flip() + theme_bw() +
  theme(legend.position = "bottom")
library("ggpubr")
ggarrange(p1, p2)
```



Ambos análisis apuntan a que las variables Ciudad, Estado_civil, Edad, Contrato y Nivel_Academico tienen una influencia alta sobre las probabilidades de Target.

Conclusión análisis exploratorio

La exploración de los datos, el estudio de su distribución, y su posible relación con la variable respuesta parecen indicar que los factores que más influyeron en Target de los clientes fueron: Ciudad, Estado_civil y Edad .

División de los datos en entrenamiento y test

Evaluar la capacidad predictiva de un modelo consiste en comprobar cómo de próximas son sus predicciones a los verdaderos valores de la variable respuesta. Para poder cuantificar de forma correcta este error, se necesita disponer de un conjunto de observaciones, de las que se conozca la variable respuesta, pero que el modelo no haya "visto", es decir, que no hayan participado en su ajuste. Con esta finalidad, se dividen los datos disponibles en un conjunto de entrenamiento y un conjunto de test. El tamaño adecuado de las particiones depende en gran medida de la cantidad de datos disponibles y la seguridad que se necesite en la estimación del error, 80%-20% suele dar buenos resultados. El reparto debe hacerse de forma aleatoria o aleatoria-estratificada. Evaluación de modelos.

Se crean los índices de las observaciones de entrenamiento

```
set.seed(342)
train <- createDataPartition(y = data$Target,
                             p = 0.7, list = FALSE, times = 1)
datos_train <- data[train, ]
datos_test <- data[-train, ]
```

Es importante verificar que la distribución de la variable respuesta es similar en el conjunto de entrenamiento y en el de test. Por defecto, la función createDataPartition() garantiza una distribución aproximada.

```
prop.table(table(datos_train$Target))
```

```
##
##        No        Si
## 0.8736617 0.1263383
```

```
prop.table(table(datos_test$Target))
```

```
##
##        No        Si
## 0.8747913 0.1252087
```

Este tipo de reparto estratificado asegura que el conjunto de entrenamiento y el de test sean similares en cuanto a la variable respuesta, sin embargo, no garantiza que ocurra lo mismo con los predictores.

Si esto ocurre en el conjunto de entrenamiento, algunos algoritmos darán error al aplicarlos al conjunto de test, ya que no entenderán el valor que se les está pasando. Este problema puede evitarse eliminando variables con varianza próxima a cero.

Preprocesado de los datos

El preprocesado de datos engloba aquellas transformaciones de los datos hechas con la finalidad de que puedan ser aceptados por el algoritmo de machine learning o que mejoren sus resultados. Todo preprocesado de datos debe aprenderse de las observaciones de entrenamiento y luego aplicarse al conjunto de entrenamiento y al de test. Esto es muy importante para no violar la condición de que ninguna información procedente de las observaciones de test puede participar o influir en el ajuste del modelo. Aunque no es posible crear un único listado, algunos de los pasos de preprocesado que más suelen aplicarse en la práctica son:

1. Imputación de valores ausentes.
2. Exclusión de variables con varianza próxima a cero.
3. Reducción de dimensionalidad.
4. Estandarización de las variables numéricas.
5. Binarización de las variables cualitativas.

Se crea un objeto recipe() con la variable respuesta y los predictores. Las variables que no parecen, no aportar información relevante sobre Target.

```
library(recipes)
```

```
##
## Attaching package: 'recipes'
```

```
## The following object is masked from 'package:stringr':
##
##     fixed
```

```
## The following object is masked from 'package:stats':
##
##     step
```

```
objeto_recipe <- recipe(formula = Target ~Ciudad+Estado_civil+Edad+
                        Contrato+Nivel_Academico, data = datos_train)
objeto_recipe
```

```
## Data Recipe
##
## Inputs:
```

```
##
##       role #variables
##    outcome          1
##  predictor          5
```

Variables con varianza próxima a cero

No se deben incluir en el modelo predictores que contengan un único valor (cero-varianza) ya que no aportan información. Tampoco es conveniente incluir predictores que tengan una varianza próxima a cero, es decir, predictores que toman solo unos pocos valores, de los cuales, algunos aparecen con muy poca frecuencia.

El problema con estos últimos es que pueden convertirse en predictores con varianza cero cuando se dividen las observaciones por validación cruzada o bootstrap.

La función nearZeroVar() del paquete caret y step_nzv() del paquete recipe identifican como predictores potencialmente problemáticos aquellos que tienen un único valor (cero varianza) o que cumplen las dos siguientes condiciones:

1. Ratio de frecuencias: ratio entre la frecuencia del valor más común y la frecuencia del segundo valor más común. Este ratio tiende a 1 si las frecuencias están equidistribuidas y a valores grandes cuando la frecuencia del valor mayoritario supera por mucho al resto (el denominador es un número decimal pequeño). Valor por defecto freqCut = 95/5.
2. Porcentaje de valores únicos: número de valores únicos dividido entre el total de muestras (multiplicado por 100). Este porcentaje se aproxima a cero cuanto mayor es la variedad de valores. Valor por defecto uniqueCut = 10.

```
data %>% select(Ciudad,Estado_civil,Edad,Contrato,Nivel_Academico) %>%
  nearZeroVar(saveMetrics = TRUE)
```

```
##                 freqRatio percentUnique zeroVar   nzv
## Ciudad           4.600639          0.25   FALSE FALSE
## Estado_civil     1.025552          0.20   FALSE FALSE
## Edad             1.071429         24.85   FALSE FALSE
## Contrato         2.288000          0.25   FALSE FALSE
## Nivel_Academico  1.382199          0.30   FALSE FALSE
```

Entre los predictores incluidos en el modelo, no se detecta ninguno con varianza cero o próxima a cero.

Estandarización y escalado

Cuando los predictores son numéricos, la escala en la que se miden, así como la magnitud de su varianza pueden influir en gran medida en el modelo. Muchos algoritmos de machine learning (SVM, redes neuronales, lasso...) son sensibles a esto, de forma que, si no se igualan de alguna forma los predictores, aquellos que se midan en una escala mayor o que tengan más varianza, dominarán el modelo aunque no sean los que más relación tienen con la variable respuesta. Existen principalmente 2 estrategias para evitarlo:

1. Centrado: consiste en restarle a cada valor la media del predictor al que pertenece. Si los datos están almacenados en un dataframe, el centrado se consigue restándole a cada valor la media de la columna en la que se encuentra. Como resultado de esta transformación, todos los predictores pasan a tener una media de cero, es decir, los valores se centran en torno al origen.
2. Normalización (estandarización): consiste en transformar los datos de forma que todos los predictores estén aproximadamente en la misma escala. Hay dos formas de lograrlo:

Para este análisis se normalizan todas las variables numéricas.

```
objeto_recipe <- objeto_recipe %>% step_center(all_numeric())
objeto_recipe <- objeto_recipe %>% step_scale(all_numeric())
```

Binarización de variables cualitativas

La binarización consiste en crear nuevas variables dummy con cada uno de los niveles de las variables cualitativas. A este proceso también se le conoce como one hot encoding. Por ejemplo, una variable llamada color que contenga los niveles rojo, verde y azul, se convertirá en tres nuevas variables (color_rojo, color_verde, color_azul), todas con el valor 0 excepto la que coincide con la observación, que toma el valor 1.

Por defecto, la función step_dummy(all_nominal()) binariza todas las variables almacenadas como tipo factor o character. Además, elimina uno de los niveles para evitar redundancias. Volviendo al ejemplo anterior, no es necesario almacenar las tres variables, ya que, si color_rojo y color_verde toman el valor 0, la variable color_azul toma necesariamente el valor 1. Si color_rojo o color_verde toman el valor 1, entonces color_azul es necesariamente 0.

```
objeto_recipe <- objeto_recipe %>% step_dummy(all_nominal(), -all_outcomes())
```

Una vez que se ha creado el objeto recipe con todas las transformaciones de preprocesado, se aprenden con los datos de entrenamiento y se aplican a los dos conjuntos.

Se entrena el objeto recipe

```
trained_recipe <- prep(objeto_recipe, training = datos_train)
trained_recipe
```

```
## Data Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor          5
##
## Training data contained 1401 data points and no missing data.
##
## Operations:
##
## Centering for Edad [trained]
## Scaling for Edad [trained]
## Dummy variables from Ciudad, Estado_civil, Contrato, Nivel_Academico [trained]
```

Se aplican las transformaciones al conjunto de entrenamiento y de test

```
datos_train_prep <- bake(trained_recipe, new_data = datos_train)
datos_test_prep <- bake(trained_recipe, new_data = datos_test)
glimpse(datos_train_prep)
```

```
## Rows: 1,401
## Columns: 18
## $ Edad                          <dbl> 0.16523041, 0.79418097, -0.897186...
## $ Target                        <fct> No, No, No, No, No, No, Si, Si, N...
## $ Ciudad_B.Manga                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Ciudad_Bogotá                 <dbl> 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, ...
## $ Ciudad_Cali                   <dbl> 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, ...
## $ Ciudad_Medellín               <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, ...
## $ Estado_civil_Soltero          <dbl> 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, ...
## $ Estado_civil_Union.Libre      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Estado_civil_Viudo            <dbl> 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, ...
## $ Contrato_Indefinido           <dbl> 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, ...
## $ Contrato_Otro                 <dbl> 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, ...
```

```
## $ Contrato_Prestacion..de.Servicios <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...
## $ Contrato_Sin.Informacion          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Nivel_Academico_Posgrado          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Nivel_Academico_Profesional       <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Nivel_Academico_Técnico           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Nivel_Academico_Tegnologo         <dbl> 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, ...
## $ Nivel_Academico_Universitario     <dbl> 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, ...
```

Modelos

En los siguientes apartados se entrenan diferentes modelos de machine learning con el objetivo de compararlos e identificar el que mejor resultado obtiene prediciendo la supervivencia de los pasajeros. Tal y como se mencionó al inicio del documento, caret hace una llamada a los modelos implementados en otros paquetes, por lo que, para conocer en detalle cómo funciona cada uno, se debe consultar la documentación del paquete correspondiente. Se puede encontrar un listado de todos los modelos disponibles, así como el paquete de origen y los hiperparámetros en este link http://topepo.github.io/caret/available-models.html.

K-Nearest Neighbor (kNN)

K-Nearest Neighbor es uno de los algoritmos de machine learning más simples. Se fundamenta en la idea de identificar observaciones en el conjunto de entrenamiento que se asemejen a la observación de test (observaciones vecinas) y asignarle como valor predicho la clase predominante entre dichas observaciones. A pesar de su sencillez, en muchos escenarios consigue resultados aceptables.

Ajuste, optimización y validación del modelo El método knn de caret emplea la función knn3() con un único hiperparámetro:

k: número de observaciones vecinas empleadas

```
set.seed(342)

#particiones:10
#repeticiones:5
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)

hiperparametros <- data.frame(k =  5)




set.seed(342)

modelo_knn <- train(Target~ ., data = datos_train_prep,
                    method = "knn",
                    tuneGrid = hiperparametros,
                    metric = "Accuracy",
                    trControl = control_train)
modelo_knn
```

```
## k-Nearest Neighbors
##
## 1401 samples
##   17 predictor
```

```
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9614755  0.8033686
##
## Tuning parameter 'k' was held constant at a value of 5
```

```
print(modelo_knn$finalModel)
```

```
## 5-nearest neighbor model
## Training set outcome distribution:
##
##   No   Si
## 1224  177
```

```
#Predicion

predicciones_knn <- predict(modelo_knn,
                            newdata = datos_test_prep,
                           type = "raw")

#Error de test

confusionMatrix(data = predicciones_knn,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 518  31
##         Si   6  44
##
##                Accuracy : 0.9382
##                  95% CI : (0.9159, 0.9561)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 2.262e-07
##
##                   Kappa : 0.6711
##
##  Mcnemar's Test P-Value : 7.961e-05
##
##             Sensitivity : 0.9885
##             Specificity : 0.5867
##          Pos Pred Value : 0.9435
##          Neg Pred Value : 0.8800
##              Prevalence : 0.8748
##          Detection Rate : 0.8648
##    Detection Prevalence : 0.9165
##       Balanced Accuracy : 0.7876
```

```
##
##        'Positive' Class : No
##
```

```
#Error de test
error_test <- mean(predicciones_knn != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 6.18 %"
```

Naive Bayes

Empleando el teorema de Bayes, el algoritmo de Naive Bayes calcula las probabilidades condicionales de que una observación pertenezca a cada una de las clases dadas unas evidencias (valores de los predictores). El término naive (ingenuo en inglés) se debe a que el algoritmo asume que las variables son independientes, es decir, que el valor que toman no está influenciado por las demás.

Aunque en la práctica, esta asunción raramente es cierta, permite calcular la probabilidad cuando hay múltiples predictores simplemente multiplicando las probabilidades individuales de cada uno (regla de eventos independientes). A pesar de que es solo una aproximación, puede resultar muy efectiva. Más información sobre Naive Bayes.

Ajuste, optimización y validación del modelo El método nb de caret emplea la función NaiveBayes() del paquete klaR con tres hiperparámetros:

usekernel: TRUE para emplear un kernel que estime la densidad o FALSE para asumir una distribución de densidad gaussiana.

fL: factor de corrección de Laplace, 0 para no aplicar ninguna corrección.

adjust: parámetro pasado a la función density si usekernel = TRUE.

```
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)


hiperparametros <- data.frame(usekernel = TRUE, fL = 0 ,
                            adjust= 5)


set.seed(342)

modelo_nb <- train(Target ~ ., data = datos_train_prep,
                   method = "nb",
                   metric = "Accuracy",
                   tuneGrid = hiperparametros,
                   trControl = control_train)
modelo_nb
```

```
## Naive Bayes
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
```

```
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8970717  0.2763683
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'usekernel' was held constant at a value of TRUE
## Tuning
##  parameter 'adjust' was held constant at a value of 5
```

```
#predicion

predicciones_nb<- predict(modelo_nb,
                          newdata = datos_test_prep,
                          type = "raw")


confusionMatrix(data = predicciones_nb,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 523  61
##         Si   1  14
##
##                Accuracy : 0.8965
##                  95% CI : (0.8693, 0.9197)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 0.05858
##
##                   Kappa : 0.2811
##
##  Mcnemar's Test P-Value : 6.731e-14
##
##             Sensitivity : 0.9981
##             Specificity : 0.1867
##          Pos Pred Value : 0.8955
##          Neg Pred Value : 0.9333
##              Prevalence : 0.8748
##          Detection Rate : 0.8731
##    Detection Prevalence : 0.9750
##       Balanced Accuracy : 0.5924
##
##        'Positive' Class : No
##
```

```
#Error de test
error_test <- mean(predicciones_nb != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 10.35 %"
```

Regresión logística

Ajuste, optimización y validación del modelo El método glm de caret emplea la función glm() del paquete básico de R. Este algoritmo no tiene ningún hiperparámetro pero, para que efectúe una regresión logística, hay que indicar family = "binomial".

```
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)

set.seed(342)

modelo_logistic <- train(Target~ ., data = datos_train_prep,
                         method = "glm",
                         metric = "Accuracy",
                         trControl = control_train,
                         family = "binomial")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
modelo_logistic
```

```
## Generalized Linear Model
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9644766  0.8314779
```

```
summary(modelo_logistic$finalModel)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.1963  -0.1319  -0.0432  -0.0196   4.1752
##
## Coefficients:
##                                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)                         4.2215     2.0033   2.107  0.03509 *
## Edad                               -1.0591     0.1684  -6.290 3.17e-10 ***
## Ciudad_B.Manga                     -3.7353     2.0945  -1.783  0.07453 .
## Ciudad_Bogotá                      -6.3074     1.9490  -3.236  0.00121 **
## Ciudad_Cali                        -1.9946     1.9484  -1.024  0.30598
## Ciudad_Medellín                    -5.2844     1.9815  -2.667  0.00766 **
## Estado_civil_Soltero                4.5162     0.6633   6.808 9.87e-12 ***
## Estado_civil_Union.Libre            2.1998     0.5294   4.155 3.26e-05 ***
## Estado_civil_Viudo                 -0.1626     0.4213  -0.386  0.69959
## Contrato_Indefinido                -1.0092     0.5413  -1.864  0.06227 .
## Contrato_Otro                       0.3859     0.5359   0.720  0.47150
## Contrato_Prestacion..de.Servicios  -0.2525     0.5004  -0.505  0.61386
## Contrato_Sin.Informacion           -0.5257     0.8358  -0.629  0.52938
## Nivel_Academico_Posgrado            6.7749   882.7440   0.008  0.99388
## Nivel_Academico_Profesional        -4.3837     0.6926  -6.330 2.46e-10 ***
## Nivel_Academico_Técnico             1.6827     0.7123   2.362  0.01816 *
## Nivel_Academico_Tegnologo          -0.8427     0.5600  -1.505  0.13237
## Nivel_Academico_Universitario      -5.6880     0.9639  -5.901 3.61e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1062.98  on 1400  degrees of freedom
## Residual deviance:  263.37  on 1383  degrees of freedom
## AIC: 299.37
##
## Number of Fisher Scoring iterations: 13
```

```r
#predicion

predicciones_logistic<- predict(modelo_logistic,
                          newdata = datos_test_prep,
                          type = "raw")

confusionMatrix(data = predicciones_logistic,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 517  27
##         Si   7  48
##
##                Accuracy : 0.9432
##                  95% CI : (0.9216, 0.9604)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 1.876e-08
##
##                   Kappa : 0.7075
```

```
##
##  Mcnemar's Test P-Value : 0.00112
##
##             Sensitivity : 0.9866
##             Specificity : 0.6400
##          Pos Pred Value : 0.9504
##          Neg Pred Value : 0.8727
##              Prevalence : 0.8748
##          Detection Rate : 0.8631
##    Detection Prevalence : 0.9082
##       Balanced Accuracy : 0.8133
##
##        'Positive' Class : No
##
```

```
#Error de test
error_test <- mean(predicciones_logistic != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 5.68 %"
```

Árbol de clasificación simple

Información detallada sobre árboles de clasificación en Árboles de predicción: bagging, random forest, boosting y C5.0.

Ajuste, optimización y validación del modelo El método C5.0Tree de caret emplea la función C5.0.default() del paquete C50 para crear un árbol de clasificación simple. Este algoritmo no tiene ningún hiperparámetro.

```
set.seed(342)
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)




modelo_C50Tree <- train(Target ~ ., data = datos_train_prep,
                        method = "C5.0Tree",
                        metric = "Accuracy",
                        trControl = control_train)
modelo_C50Tree
```

```
## Single C5.0 Tree
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9613194  0.8119204
```

```r
summary(modelo_C50Tree$finalModel)
```

```
##
## Call:
## C50:::C5.0.default(x = x, y = y, weights = wts)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sat Nov 07 23:10:55 2020
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 1401 cases (18 attributes) from undefined.data
##
## Decision tree:
##
## Edad <= -1.35615:
## :...Nivel_Academico_Universitario > 0:
## :   :...Estado_civil_Viudo > 0: No (4)
## :   :   Estado_civil_Viudo <= 0:
## :   :   :...Edad <= -1.636628: Si (4/1)
## :   :       Edad > -1.636628: No (4)
## :   Nivel_Academico_Universitario <= 0:
## :   :...Nivel_Academico_Profesional <= 0: Si (83/7)
## :       Nivel_Academico_Profesional > 0:
## :       :...Estado_civil_Soltero > 0: Si (7)
## :           Estado_civil_Soltero <= 0:
## :           :...Estado_civil_Union.Libre <= 0: No (11)
## :               Estado_civil_Union.Libre > 0: Si (4/1)
## Edad > -1.35615:
## :...Ciudad_Cali > 0:
##     :...Nivel_Academico_Universitario > 0: No (16)
##     :   Nivel_Academico_Universitario <= 0:
##     :   :...Nivel_Academico_Profesional > 0:
##     :       :...Estado_civil_Soltero <= 0: No (31/2)
##     :       :   Estado_civil_Soltero > 0: Si (2)
##     :       Nivel_Academico_Profesional <= 0:
##     :       :...Contrato_Indefinido <= 0: Si (40)
##     :           Contrato_Indefinido > 0:
##     :           :...Edad <= 0.2927204: Si (6)
##     :               Edad > 0.2927204: No (2)
##     Ciudad_Cali <= 0:
##     :...Estado_civil_Soltero <= 0:
##         :...Estado_civil_Union.Libre <= 0: No (1107/18)
##         :   Estado_civil_Union.Libre > 0:
##         :   :...Contrato_Otro <= 0: No (37/1)
##         :       Contrato_Otro > 0: Si (5)
##         Estado_civil_Soltero > 0:
##         :...Nivel_Academico_Profesional > 0: No (16/1)
##             Nivel_Academico_Profesional <= 0:
##             :...Contrato_Prestacion..de.Servicios <= 0: Si (10/1)
##                 Contrato_Prestacion..de.Servicios > 0:
##                 :...Nivel_Academico_Universitario > 0: No (6/1)
##                     Nivel_Academico_Universitario <= 0:
```

```
##                     :...Edad <= -0.2172395: Si (2)
##                         Edad > -0.2172395: No (4/1)
##
##
## Evaluation on training data (1401 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##      21   34( 2.4%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##    1214    10    (a): class No
##      24   153    (b): class Si
##
##
##  Attribute usage:
##
##  100.00% Edad
##   91.65% Ciudad_Cali
##   88.65% Estado_civil_Soltero
##   83.08% Estado_civil_Union.Libre
##   16.13% Nivel_Academico_Universitario
##   15.99% Nivel_Academico_Profesional
##    3.43% Contrato_Indefinido
##    3.00% Contrato_Otro
##    1.57% Contrato_Prestacion..de.Servicios
##    0.86% Estado_civil_Viudo
##
##
## Time: 0.0 secs
```

```r
#predicion
set.seed(342)
predicciones_C50Tree<- predict(modelo_C50Tree,
                               newdata = datos_test_prep,
                               type = "raw")

confusionMatrix(data = predicciones_C50Tree,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 511  22
##         Si  13  53
##
##                Accuracy : 0.9416
##                  95% CI : (0.9197, 0.959)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 4.429e-08
```

```
##
##                   Kappa : 0.7188
##
##  Mcnemar's Test P-Value : 0.1763
##
##             Sensitivity : 0.9752
##             Specificity : 0.7067
##          Pos Pred Value : 0.9587
##          Neg Pred Value : 0.8030
##              Prevalence : 0.8748
##          Detection Rate : 0.8531
##    Detection Prevalence : 0.8898
##       Balanced Accuracy : 0.8409
##
##        'Positive' Class : No
##
```

```
#Error de test
error_test <- mean(predicciones_C50Tree != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 5.84 %"
```

RandomForest

Información detallada sobre random forest en Árboles de predicción: bagging, random forest, boosting y C5.0

Ajuste, optimización y validación del modelo El método ranger de caret emplea la función ranger() del paquete ranger. Este algoritmo tiene 3 hiperparámetros:

mtry: número predictores seleccionados aleatoriamente en cada árbol. min.node.size: tamaño mínimo que tiene que tener un nodo para poder ser dividido.

splitrule: criterio de división. Aunque caret también incluye el método rf con la función rf() del paquete randomForest, este último solo permite optimizar el hiperparámetro mtry.

```
set.seed(342)
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)

hiperparametros <- expand.grid(mtry = 7,
                     min.node.size = 20,
                     splitrule = "gini")



set.seed(342)
modelo_rf <- train(Target~ .,
                   data = datos_train_prep,
                   method = "ranger",
                   metric = "Accuracy",
                   tuneGrid = hiperparametros,
                   trControl = control_train,
                   num.trees = 100)
modelo_rf
```

```
## Random Forest
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9641807  0.8292328
##
## Tuning parameter 'mtry' was held constant at a value of 7
## Tuning
##  parameter 'splitrule' was held constant at a value of gini
## Tuning
##  parameter 'min.node.size' was held constant at a value of 20
```

```
summary(modelo_rf$finalModel)
```

```
##                             Length Class         Mode
## predictions                 1401   factor        numeric
## num.trees                      1   -none-        numeric
## num.independent.variables      1   -none-        numeric
## mtry                           1   -none-        numeric
## min.node.size                  1   -none-        numeric
## prediction.error               1   -none-        numeric
## forest                         9   ranger.forest list
## confusion.matrix               4   table         numeric
## splitrule                      1   -none-        character
## treetype                       1   -none-        character
## call                           9   -none-        call
## importance.mode                1   -none-        character
## num.samples                    1   -none-        numeric
## replace                        1   -none-        logical
## xNames                        17   -none-        character
## problemType                    1   -none-        character
## tuneValue                      3   data.frame    list
## obsLevels                      2   -none-        character
## param                          1   -none-        list
```

```
#predicion
set.seed(342)
predicciones_rf<- predict(modelo_rf,
                          newdata = datos_test_prep,
                          type = "raw")

confusionMatrix(data = predicciones_rf,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  No  Si
##         No 510  23
##         Si  14  52
##
##                Accuracy : 0.9382
##                  95% CI : (0.9159, 0.9561)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 2.262e-07
##
##                   Kappa : 0.7027
##
##  Mcnemar's Test P-Value : 0.1884
##
##             Sensitivity : 0.9733
##             Specificity : 0.6933
##          Pos Pred Value : 0.9568
##          Neg Pred Value : 0.7879
##              Prevalence : 0.8748
##          Detection Rate : 0.8514
##    Detection Prevalence : 0.8898
##       Balanced Accuracy : 0.8333
##
##        'Positive' Class : No
##
```

```
#Error de test
error_test <- mean(predicciones_rf != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 6.18 %"
```

Gradient Boosting

n.trees: número de iteraciones del algoritmo de boosting, es decir, número de modelos que forman el ensemble. Cuanto mayor es este valor, más se reduce el error de entrenamiento, pudiendo llegar generarse overfitting.

interaction.depth: complejidad de los árboles empleados como weak learner, en concreto, el número total de divisiones que tiene el árbol. Emplear árboles con ente 1 y 6 nodos suele dar buenos resultados.

shrinkage: este parámetro, también conocido como learning rate, controla la influencia que tiene cada modelo sobre el conjunto del ensemble.

n.minobsinnode: número mínimo de observaciones que debe tener un nodo para poder ser dividido. Al igual que interaction.depth, permite controlar la complejidad de los weak learners basados en árboles.

```
set.seed(342)
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)


hiperparametros <- expand.grid(interaction.depth = 2,
                               n.trees = 200,
                               shrinkage = 0.1,
                               n.minobsinnode = 15)
```

```r
set.seed(342)

modelo_boost <- train(Target ~ .,
                      data = datos_train_prep, method = "gbm",
                      tuneGrid = hiperparametros,
                      metric = "Accuracy",
                      trControl = control_train,
                      # Número de árboles ajustados distribution = "adaboost",
                      verbose = FALSE)
```

```
## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 13: Nivel_Academico_Posgrado has no variation.

## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 13: Nivel_Academico_Posgrado has no variation.

## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 13: Nivel_Academico_Posgrado has no variation.

## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 13: Nivel_Academico_Posgrado has no variation.

## Warning in (function (x, y, offset = NULL, misc = NULL, distribution =
## "bernoulli", : variable 13: Nivel_Academico_Posgrado has no variation.
```

```r
modelo_boost
```

```
## Stochastic Gradient Boosting
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9647643  0.8317184
##
## Tuning parameter 'n.trees' was held constant at a value of 200
## Tuning
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 15
```

```r
#predicion
set.seed(342)
predicciones_boost<- predict(modelo_boost,
                         newdata = datos_test_prep,
                         type = "raw")
```

```r
confusionMatrix(data = predicciones_boost,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 512  25
##         Si  12  50
##
##                Accuracy : 0.9382
##                  95% CI : (0.9159, 0.9561)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 2.262e-07
##
##                   Kappa : 0.6954
##
##  Mcnemar's Test P-Value : 0.04852
##
##             Sensitivity : 0.9771
##             Specificity : 0.6667
##          Pos Pred Value : 0.9534
##          Neg Pred Value : 0.8065
##              Prevalence : 0.8748
##          Detection Rate : 0.8548
##    Detection Prevalence : 0.8965
##       Balanced Accuracy : 0.8219
##
##        'Positive' Class : No
##
```

```r
#Error de test
error_test <- mean(predicciones_boost != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 6.18 %"
```

SVM Máquinas de Vector Soporte

sigma: coeficiente del kernel radial. C: penalización por violaciones del margen del hiperplano.

```r
set.seed(342)
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)
# Hiperparámetros

hiperparametros <- expand.grid(sigma =  0.01,
                               C = 100)


set.seed(342)

modelo_svmrad <- train(Target ~ .,
                       data = datos_train_prep,
```

```
                        method = "svmRadial",
                        tuneGrid = hiperparametros,
                        metric = "Accuracy",
                        trControl = control_train)
```

```
## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.

## Warning in .local(x, ...): Variable(s) `' constant. Cannot scale data.
```

```
modelo_svmrad
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9716033  0.8670657
##
## Tuning parameter 'sigma' was held constant at a value of 0.01
## Tuning
##  parameter 'C' was held constant at a value of 100
```

```
#predicion
set.seed(342)
predicciones_svmrad<- predict(modelo_svmrad,
                              newdata = datos_test_prep,
                              type = "raw")

confusionMatrix(data = predicciones_svmrad,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 517  27
##         Si   7  48
##
##                Accuracy : 0.9432
##                  95% CI : (0.9216, 0.9604)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 1.876e-08
```

```
##
##                   Kappa : 0.7075
##
##  Mcnemar's Test P-Value : 0.00112
##
##             Sensitivity : 0.9866
##             Specificity : 0.6400
##          Pos Pred Value : 0.9504
##          Neg Pred Value : 0.8727
##              Prevalence : 0.8748
##          Detection Rate : 0.8631
##    Detection Prevalence : 0.9082
##       Balanced Accuracy : 0.8133
##
##        'Positive' Class : No
##
```

```
#Error de test
error_test <- mean(predicciones_svmrad != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 5.68 %"
```

Redes neuronales (NNET)

size: número de neuronas en la capa oculta. decay: controla la regularización durante el entrenamiento de la red.

```
# Hiperparámetros

set.seed(342)
control_train <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5)


hiperparametros <- expand.grid(size = 80,
                               decay = 0.1)




set.seed(342)
modelo_nnet <- train(Target~ ., data = datos_train_prep,
                     method = "nnet",
                     tuneGrid = hiperparametros,
                     metric = "Accuracy",
                     trControl = control_train,
                     # Se aumenta el número máximo de pesos
                     MaxNWts = 2000,
                     # Para que no se muestre cada iteración por pantalla
                     trace = FALSE)

modelo_nnet
```

```
## Neural Network
```

```
##
## 1401 samples
##   17 predictor
##    2 classes: 'No', 'Si'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1261, 1262, 1260, 1261, 1262, 1260, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9721788  0.8703133
##
## Tuning parameter 'size' was held constant at a value of 80
## Tuning
##  parameter 'decay' was held constant at a value of 0.1
```

```
#predicion
set.seed(342)
predicciones_nnet<- predict(modelo_nnet,
                            newdata = datos_test_prep,
                            type = "raw")

#Error de test

confusionMatrix(data = predicciones_nnet,
                reference = datos_test_prep$Target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No  Si
##         No 515  21
##         Si   9  54
##
##                Accuracy : 0.9499
##                  95% CI : (0.9293, 0.966)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 4.42e-10
##
##                   Kappa : 0.7545
##
##  Mcnemar's Test P-Value : 0.04461
##
##             Sensitivity : 0.9828
##             Specificity : 0.7200
##          Pos Pred Value : 0.9608
##          Neg Pred Value : 0.8571
##              Prevalence : 0.8748
##          Detection Rate : 0.8598
##    Detection Prevalence : 0.8948
##       Balanced Accuracy : 0.8514
##
##        'Positive' Class : No
##
```

```r
#Error de test
error_test <- mean(predicciones_nnet != datos_test_prep$Target)
paste("El error de test del modelo:", round(error_test*100, 2), "%")
```

```
## [1] "El error de test del modelo: 5.01 %"
```

Comparación de modelos

Una vez que se han entrenado y optimizado distintos modelos, se tiene que identificar cuál de ellos consigue mejores resultados para el problema en cuestión, en este caso, predecir la supervivencia de los pasajeros. Con los datos disponibles, existen dos formas de comparar los modelos. Si bien las dos no tienen por qué dar los mismos resultados, son complementarias a la hora de tomar una decisión final.

La función resamples() permite extraer, de uno o varios modelos creados con train(), las métricas obtenidas para cada repetición del proceso de validación. Es importante tener en cuenta que esta función recupera todos los resultados, por lo que, si no se especifica en el control de entrenamiento returnResamp = "final", se devuelven los valores para todos los hiperparámetros, no solo los del modelo final.

```r
modelos <- list(KNN = modelo_knn,
                NB = modelo_nb,
                logistic = modelo_logistic,
                arbol = modelo_C50Tree,
                rf = modelo_rf,
                boosting = modelo_boost,
                SVMradial = modelo_svmrad,
                NNET = modelo_nnet)

resultados_resamples <- resamples(modelos)



metricas_resamples <- resultados_resamples$values %>%
  gather(key = "modelo", value = "valor", -Resample) %>%
  separate(col = "modelo", into = c("modelo", "metrica"),
           sep = "~", remove = TRUE)


metricas_resamples %>%
  group_by(modelo, metrica) %>%
  summarise(media = mean(valor)) %>%
  spread(key = metrica, value = media) %>%
  arrange(desc(Accuracy))
```

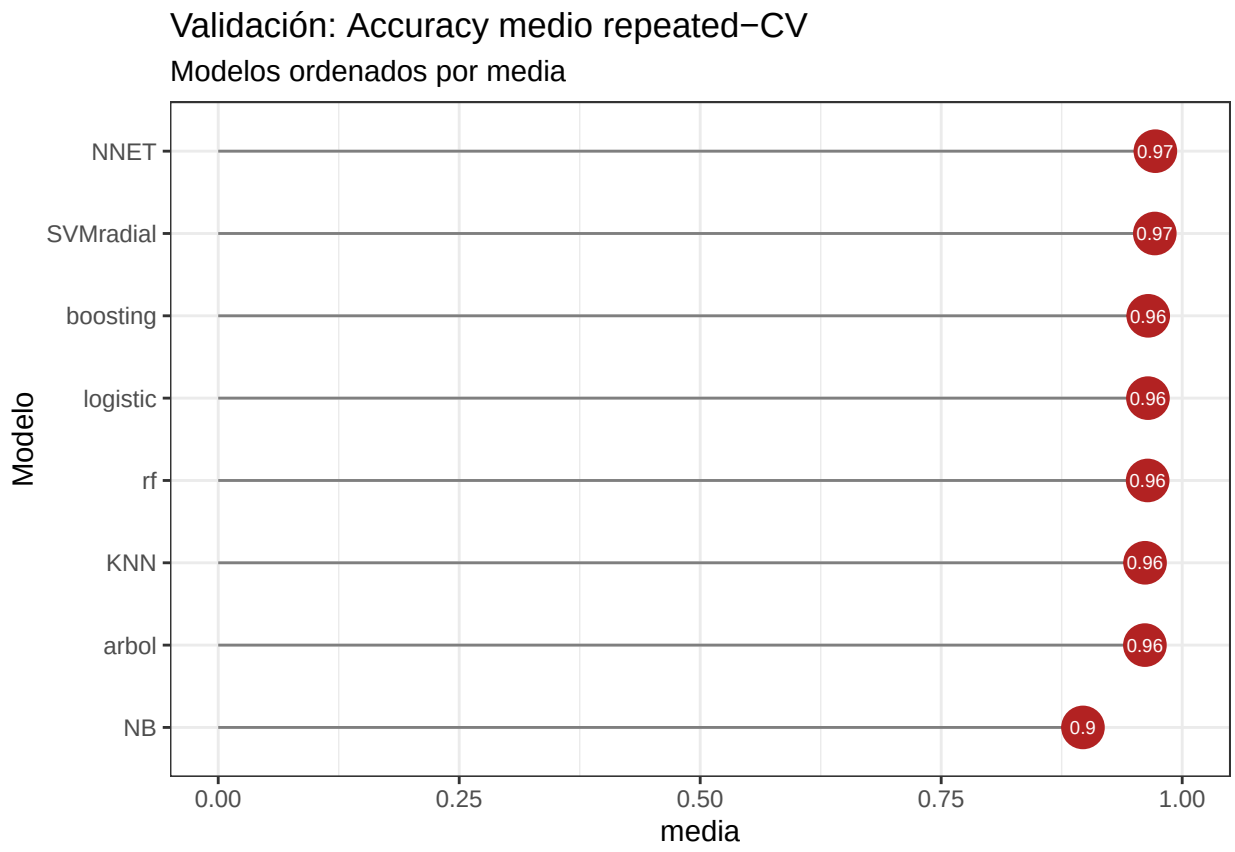
```
## `summarise()` regrouping output by 'modelo' (override with `.groups` argument)
```

```
## # A tibble: 8 x 3
## # Groups:   modelo [8]
##   modelo    Accuracy Kappa
##   <chr>        <dbl> <dbl>
## 1 NNET         0.972 0.870
## 2 SVMradial    0.972 0.867
## 3 boosting     0.965 0.832
## 4 logistic     0.964 0.831
## 5 rf           0.964 0.829
## 6 KNN          0.961 0.803
## 7 arbol        0.961 0.812
```

```
## 8 NB           0.897 0.276
```

```
metricas_resamples %>%
  filter(metrica == "Accuracy") %>%
  group_by(modelo) %>%
  summarise(media = mean(valor)) %>%
  ggplot(aes(x = reorder(modelo, media),
             y = media, label = round(media, 2))) +
  geom_segment(aes(x = reorder(modelo, media),
                   y = 0, xend = modelo, yend = media),
                color = "grey50") +
  geom_point(size = 7, color = "firebrick") +
  geom_text(color = "white", size = 2.5) +
  scale_y_continuous(limits = c(0, 1)) +
  # Accuracy basal geom_hline(yintercept = 0.62, linetype = "dashed") +
  labs(title = "Validación: Accuracy medio repeated-CV",
       subtitle = "Modelos ordenados por media", x = "Modelo") +
  coord_flip() + theme_bw()
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```



Error de test

Aunque está demostrado que los métodos de validación tipo CV, bootstrapping, LOOCV... consiguen estimaciones muy buenas del error que comente un modelo, es conveniente hacer una medición final con nuevas observaciones para asegurar que, durante la optimización, no se haya generado overfitting. Esta es la razón por la que, al inicio de un análisis, se separa un conjunto de test que se mantiene aislado de todo el

proceso de transformaciones, entrenamiento y optimización.

Tal y como se describió anteriormente, si se desea obtener predicciones para varios modelos, es conveniente emplear la función extractPrediction(). Esta función devuelve un dataframe con las predicciones de cada uno de los modelos, tanto para las observaciones de entrenamiento como para las de test. Además, muestra el verdadero valor de cada observación.

```
predicciones <- extractPrediction( models = modelos,
                                   testX = datos_test_prep[, -2],
                                   testY = datos_test_prep$Target)

predicciones %>% head()
```

```
##   obs pred model dataType object
## 1  No   No   knn Training    KNN
## 2  No   No   knn Training    KNN
## 3  No   No   knn Training    KNN
## 4  No   No   knn Training    KNN
## 5  No   No   knn Training    KNN
## 6  No   No   knn Training    KNN
```

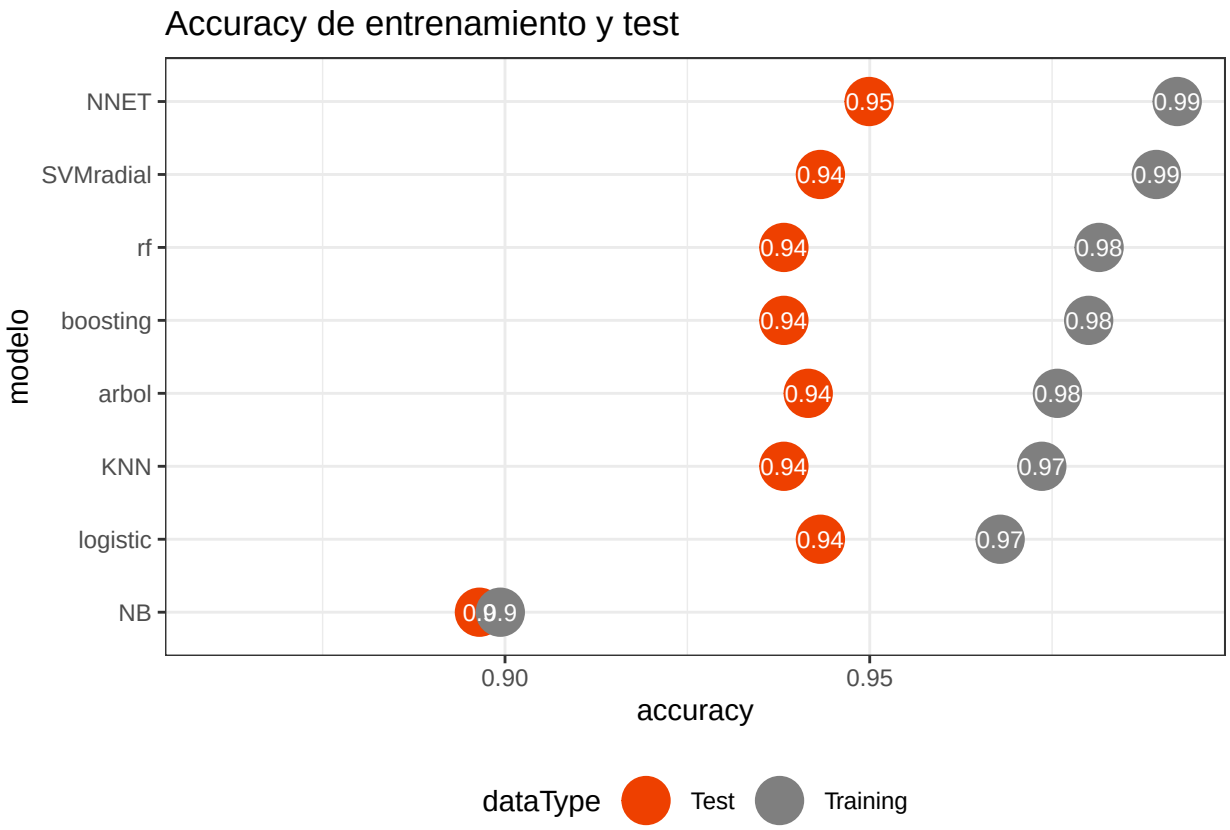
```
metricas_predicciones <- predicciones %>%
  mutate(acierto = ifelse(obs == pred, TRUE, FALSE)) %>%
  group_by(object, dataType) %>% summarise(accuracy = mean(acierto))
```

```
## `summarise()` regrouping output by 'object' (override with `.groups` argument)
```

```
metricas_predicciones %>%
  spread(key = dataType, value = accuracy) %>%
  arrange(desc(Test))
```

```
## # A tibble: 8 x 3
## # Groups:   object [8]
##   object     Test Training
##   <chr>     <dbl>    <dbl>
## 1 NNET      0.950    0.992
## 2 logistic  0.943    0.968
## 3 SVMradial 0.943    0.989
## 4 arbol     0.942    0.976
## 5 boosting  0.938    0.980
## 6 KNN       0.938    0.974
## 7 rf        0.938    0.981
## 8 NB        0.896    0.899
```

```
ggplot(data = metricas_predicciones,
       aes(x = reorder(object, accuracy),
           y = accuracy, color = dataType,
           label = round(accuracy, 2))) + geom_point(size = 8) +
  scale_color_manual(values = c("orangered2", "gray50")) +
  geom_text(color = "white", size = 3) +
 # geom_hline(yintercept = 0.85, linetype = "dashed") +
  annotate(geom = "text", y = 0.86, x = 8.5, label = "") +
  coord_flip() +
  labs(title = "Accuracy de entrenamiento y test", x = "modelo") +
  theme_bw() + theme(legend.position = "bottom")
```



Puede verse que, todos los modelos, consiguen más predicciones correctas en el conjunto de entrenamiento que en el de test, de ahí que las métricas obtenidas en el entrenamiento no deban utilizarse para evaluar los modelos, son excesivamente optimistas.