Trabajo ML1: Parte 3

Jose Castro

2024-07-13

Contents

Analisis Descriptivo de Datos y Preparacion de Datos

Realizo la importacion de librerias que ocuparé para el analisis descriptivo, el modelo k-NN de clasificacion y el modelo Naive Bayes.

```
# importacion de librerias
library(ggplot2)
library(dplyr)
library(readr)
library(vcd)
library(reshape2)
library(ltm)
library(gmodels)
library(pROC)
library(kknn)
library(caret)
library(e1071)
library(rsample)
library(naivebayes)
library(doParallel)
```

Cargo la base de datos, que fue descargada del siguiente link:

https://www.kaggle.com/datasets/joebeachcapital/loan-default

```
loan_data <- read_csv("Anonymize_Loan_Default_data.csv")
```

```
## New names:
## Rows: 38480 Columns: 37
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (14): term, emp_length, home_ownership, verification_status, issue_d, lo... dbl
## (23): ...1, id, member_id, loan_amnt, funded_amnt, funded_amnt_inv, int_...
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

```
loan_data <- as.data.frame(loan_data)
```

El siguiente bloque de código tiene como objetivo asegurarse de que todos los datos de tipo carácter en el conjunto de datos loan_data estén codificados en UTF-8.

```
# Recorro todas las columnas del dataframe
for (i in seq_along(loan_data)) {
  # Verifico si la columna es de tipo caracter
  if (is.character(loan_data[[i]])) {
    # Convierto la columna a UTF-8
    loan_data[[i]] <- iconv(loan_data[[i]], to = "UTF-8")
  }
}
```

Este paso es crucial para evitar problemas de codificación que podrían surgir durante el análisis o procesamiento de los datos. La codificación UTF-8 es un estándar que garantiza la compatibilidad de caracteres especiales y acentos, especialmente útil si el conjunto de datos incluye texto en diferentes idiomas o caracteres especiales.

1. **Recorrido de columnas**: Se utiliza un bucle for para iterar sobre todas las columnas del dataframe loan_data. seq_along(loan_data) genera una secuencia de índices basada en el número de columnas en el dataframe.

2. **Verificación de tipo de columna**: Dentro del bucle, se verifica si la columna actual es de tipo carácter utilizando `is.character(loan_data[[i]])`.
3. **Conversión a UTF-8**: Si la columna es de tipo carácter, se convierte a UTF-8 utilizando la función iconv, lo cual asegura que todos los caracteres sean compatibles y se puedan manejar correctamente en análisis posteriores.

Este paso es una medida preventiva importante para evitar errores relacionados con la codificación de caracteres en etapas posteriores del análisis.

La siguiente tiene como objetivo limpiar el conjunto de datos loan_data eliminando columnas con altos porcentajes de valores faltantes (NA) y columnas irrelevantes. La limpieza de datos es un paso esencial para asegurar que los análisis y modelos posteriores se basen en datos de calidad y evitar sesgos o errores.

```
#borro la columna que contiene el muchos NA, 91% de los datos
#columna next_pymnt_d
loan_data <- loan_data[, -which(names(loan_data) %in% c("next_pymnt_d"))]
#borro la primera columna que se llama ...1
loan_data <- loan_data[, -which(names(loan_data) %in% c("...1"))]
#borro la columna que tiene 61% de sus valores como NA
loan_data <- loan_data[, -which(names(loan_data) %in%
                                  c("mths_since_last_delinq"))]
```

La siguiente celda se utiliza para eliminar filas con valores faltantes (NA) del conjunto de datos loan_data, asegurando así que los datos sean completos para el análisis y modelado posterior. La eliminación de valores NA es un paso importante en la preparación de datos para evitar problemas durante el análisis.

```
#numero de filas y columnas antes de eliminar los NA
dim(loan_data)
```

```
## [1] 38480    34
```

```
#elimino las filas con valores NA de todas las columnas
loan_data <- na.omit(loan_data)
#numero de filas y columnas despues de eliminar los NA
dim(loan_data)
```

```
## [1] 38345    34
```

El resultado **38345 34** indica que el conjunto de datos ahora tiene 38345 filas y 34 columnas, lo que muestra que se eliminaron 135 filas con valores faltantes.

Eliminar las filas con valores faltantes es crucial para asegurar la integridad de los datos y evitar errores o sesgos en el análisis y modelado. Este paso garantiza que los algoritmos de machine learning puedan trabajar con datos completos y confiables.

```
#reviso si hay columnas con valores NA
loan_data %>% summarise(across(everything(), ~sum(is.na(.))))
```

```
##   id member_id loan_amnt funded_amnt funded_amnt_inv term int_rate installment
## 1  0         0         0           0               0    0        0           0
##   emp_length home_ownership annual_inc verification_status issue_d loan_status
## 1          0              0          0                   0       0           0
##   purpose zip_code addr_state dti delinq_2yrs earliest_cr_line inq_last_6mths
## 1       0        0          0   0           0                0              0
##   open_acc pub_rec revol_bal revol_util total_acc total_pymnt total_pymnt_inv
## 1        0       0         0          0         0           0               0
##   total_rec_prncp total_rec_int last_pymnt_d last_pymnt_amnt last_credit_pull_d
## 1               0             0            0               0                  0
##   repay_fail
```

```
## 1           0
```

Con esto queda comprobado que ya no hay NA en las columnas, por lo que esto me asegura que no ocurriran errores al entrenar los modelos de ML provenientes de NA.

El siguiente código calcula el coeficiente de Cramér para evaluar la fuerza de asociación entre variables categóricas y la variable objetivo repay_fail, que es binaria. El coeficiente de Cramér es una medida de asociación entre dos variables categóricas y es útil en el análisis exploratorio de datos.

```
# Función para calcular el coeficiente de Cramér en relacion a 2 variables
# una variable categórica y target binaria
cramers_v <- function(x, y) {
  tbl <- table(x, y)
  chi2 <- chisq.test(tbl, correct = FALSE)$statistic
  n <- sum(tbl)
  min_dim <- min(dim(tbl)) - 1
  return(sqrt(chi2 / (n * min_dim)))
}
```

```
# Calculo el coeficiente de Cramér para cada variable categórica
# con respecto a 'repay_fail'
vars_to_test <- c("zip_code", "addr_state", "last_pymnt_d",
                  "last_credit_pull_d", "earliest_cr_line", "issue_d",
                  "emp_length", "home_ownership", "loan_status", "purpose")
correlacion <- sapply(vars_to_test, function(var) {
  suppressWarnings(cramers_v(loan_data[[var]], loan_data$repay_fail))
})
```

```
# Muestro los resultados
correlacion
```

```
##           zip_code.X-squared         addr_state.X-squared
##                   0.16901610                   0.06630926
##       last_pymnt_d.X-squared last_credit_pull_d.X-squared
##                   0.26395049                   0.22507931
##   earliest_cr_line.X-squared           issue_d.X-squared
##                   0.12621675                   0.08353522
##         emp_length.X-squared     home_ownership.X-squared
##                   0.03816881                   0.02315577
##        loan_status.X-squared           purpose.X-squared
##                   1.00000000                   0.09867921
```

zip_code (0.1690):

- Baja correlación: Un valor de 0.169 indica que hay una asociación baja entre zip_code y repay_fail. Aunque no es insignificante, la alta cardinalidad (834 clases) de esta variable puede introducir ruido más que valor predictivo significativo.
- Recomendación: Borrar esta columna.

addr_state (0.0663):

- Muy baja correlación: Un valor de 0.066 indica una correlación muy baja entre addr_state y repay_fail. Esto sugiere que el estado de residencia no es relevante para la predicción de incumplimiento.
- Recomendación: Borrar esta columna.

last_pymnt_d (0.2639):

- Moderada correlación: Con un valor de 0.2639, esta variable muestra una correlación moderada con repay_fail. Esto sugiere que la fecha del último pago tiene cierta relevancia para la predicción.
- Recomendación: Mantener esta columna.

last_credit_pull_d (0.2250):

- Baja a moderada correlación: Un valor de 0.225 indica una baja a moderada asociación entre last_credit_pull_d y repay_fail. A pesar de esto, puede no ser suficiente para justificar su inclusión debido a la complejidad de manejar fechas.
- Recomendación: Mantener esta columna.

earliest_cr_line (0.1262):

- Baja correlación: Un valor de 0.1262 indica una asociación baja entre earliest_cr_line y repay_fail. Esto sugiere que la fecha de la línea de crédito más antigua no tiene una relevancia significativa para predecir el incumplimiento.
- Recomendación: Borrar esta columna.

issue_d (0.0835):

- Muy baja correlación: Un valor de 0.0835 indica una asociación muy baja entre issue_d y repay_fail. Esto sugiere que la fecha de emisión del préstamo no es muy relevante para predecir el incumplimiento.
- Recomendación: Borrar esta columna.

emp_length (0.0381)

- Muy baja correlación
- Recomendación: Borrar esta columna

home_ownership (0.0231)

- Muy baja correlación
- Recomendación: Borrar esta columna

loan_status (1)

- correlacion 1 a 1 con repay_fail
- Recomendación: Borrar esta columna porque es basicamente lo mismo que la columna a predecir.

purpose (0.0986)

- Muy baja correlación
- Recomendación: Borrar esta columna

La siguiente celda de código se enfoca en eliminar varias columnas del conjunto de datos loan_data que han mostrado tener baja correlación con la variable objetivo repay_fail. Eliminar variables con baja correlación es una práctica común en la preparación de datos para reducir la dimensionalidad y simplificar los modelos sin perder información significativa.

```
#ejecuto las recomendaciones de borrar 2 columnas con baja correlacion
#elimino la columna zip_code
loan_data <- loan_data[, -which(names(loan_data) %in% c("zip_code"))]
#elimino la columna addr_state
loan_data <- loan_data[, -which(names(loan_data) %in% c("addr_state"))]
#elimino la columna earliest_cr_line
loan_data <- loan_data[, -which(names(loan_data) %in% c("earliest_cr_line"))]
#elimino la columna issue_d
loan_data <- loan_data[, -which(names(loan_data) %in% c("issue_d"))]
#elimino la columna emp_length
loan_data <- loan_data[, -which(names(loan_data) %in% c("emp_length"))]
#elimino la columna home_ownership
loan_data <- loan_data[, -which(names(loan_data) %in% c("home_ownership"))]
#elimino la columna loan_status
loan_data <- loan_data[, -which(names(loan_data) %in% c("loan_status"))]
#elimino la columna purpose
```

```r
loan_data <- loan_data[, -which(names(loan_data) %in% c("purpose"))]
dim(loan_data)
```

```
## [1] 38345    26
```

El siguiente bloque de código se centra en la conversión y formateo adecuado de columnas de fecha.

```r
#Indico a R que trabajaremos con fechas en inglés
Sys.setlocale("LC_TIME", "English")
```

```
## [1] "English_United States.1252"
```

```r
# Convierto las columnas de fecha al formato de fecha adecuado
loan_data$last_pymnt_d <- as.Date(paste0("01-", loan_data$last_pymnt_d),
                                  format = "%d-%b-%y")
loan_data$last_credit_pull_d <- as.Date(paste0("01-", loan_data$last_credit_pull_d), format = "%d-%b-%y
# Verifico las conversiones de fecha
head(loan_data[, c("last_pymnt_d", "last_credit_pull_d")])
```

```
##   last_pymnt_d last_credit_pull_d
## 2   2013-07-01         2016-06-01
## 3   2011-11-01         2012-03-01
## 4   2014-03-01         2014-03-01
## 5   2014-02-01         2016-06-01
## 6   2013-05-01         2016-06-01
## 7   2014-04-01         2014-04-01
```

Los siguientes 2 bloques de código se encargan de analizar las fechas mínimas y máximas de las columnas last_pymnt_d y last_credit_pull_d, y de crear nuevas columnas categorizando estas fechas en intervalos específicos. La categorización de fechas en intervalos será útil para disminuir los numeros de clases de cada columna de fechas.

```r
#fecha minimas y maximas
min(loan_data$last_pymnt_d)
```

```
## [1] "2007-12-01"
```

```r
max(loan_data$last_pymnt_d)
```

```
## [1] "2016-06-01"
```

```r
min(loan_data$last_credit_pull_d)
```

```
## [1] "2007-05-01"
```

```r
max(loan_data$last_credit_pull_d)
```

```
## [1] "2016-06-01"
```

Creo columnas nuevas que agrupen fecha en intervalos, para las columnas `last_pymnt_d` y `last_credit_pull_d`, estos intervalos seran 2007-2010, 2011-2013 y 2014-2016.

```r
# Creo nuevas columnas con los intervalos de fechas especificados
loan_data <- loan_data %>%
  mutate(
    last_pymnt_d2 = case_when(
      last_pymnt_d >= as.Date("2007-01-01") & last_pymnt_d <= as.Date("2010-12-31") ~ "2007-2010",
      last_pymnt_d >= as.Date("2011-01-01") & last_pymnt_d <= as.Date("2013-12-31") ~ "2011-2013",
      last_pymnt_d >= as.Date("2014-01-01") & last_pymnt_d <= as.Date("2016-12-31") ~ "2014-2016",
      TRUE ~ NA_character_
```

```
    ),
    last_credit_pull_d2 = case_when(
      last_credit_pull_d >= as.Date("2007-01-01") & last_credit_pull_d <= as.Date("2010-12-31") ~ "2007
      last_credit_pull_d >= as.Date("2011-01-01") & last_credit_pull_d <= as.Date("2013-12-31") ~ "2011
      last_credit_pull_d >= as.Date("2014-01-01") & last_credit_pull_d <= as.Date("2016-12-31") ~ "2014
      TRUE ~ NA_character_
    )
  )

# Verifico las nuevas columnas
head(loan_data[, c("last_pymnt_d", "last_pymnt_d2", "last_credit_pull_d", "last_credit_pull_d2")])
```

```
##   last_pymnt_d last_pymnt_d2 last_credit_pull_d last_credit_pull_d2
## 2   2013-07-01     2011-2013         2016-06-01           2014-2016
## 3   2011-11-01     2011-2013         2012-03-01           2011-2013
## 4   2014-03-01     2014-2016         2014-03-01           2014-2016
## 5   2014-02-01     2014-2016         2016-06-01           2014-2016
## 6   2013-05-01     2011-2013         2016-06-01           2014-2016
## 7   2014-04-01     2014-2016         2014-04-01           2014-2016
```

Ahora que ya generé nuevas columnas con fechas agrupadas en intervalos, procedo a borrar las originales que ya no me sirven y proceso a renombrar las nuevas con el nombre de las antiguas columnas.

```
#borro columnas last_pymnt_d y last_credit_pull_d
loan_data <- loan_data[, -which(names(loan_data) %in% c("last_pymnt_d"))]
loan_data <- loan_data[, -which(names(loan_data) %in% c("last_credit_pull_d"))]
#cambio el nombre de las columnas last_pymnt_d2 y last_credit_pull_d2
colnames(loan_data)[colnames(loan_data) == "last_pymnt_d2"] <- "last_pymnt_d"
colnames(loan_data)[colnames(loan_data) == "last_credit_pull_d2"] <- "last_credit_pull_d"
```

La siguiente celda se encarga de limpiar y convertir los valores de la columna revol_util del conjunto de datos loan_data, eliminando el signo de porcentaje y convirtiendo los valores a formato numérico.

```
#elimino el signo de porcentaje de la columna revol_util
loan_data$revol_util <- as.numeric(gsub("%", "", loan_data$revol_util))
```

```
# selecciono las columnas categoricas
# luego veo la cantidad de valores por cada categoria o clase
categorical_columns <- c("term",
                         "verification_status",
                         "last_pymnt_d", "last_credit_pull_d",
                         "repay_fail")
valores_por_categoria <- lapply(loan_data[categorical_columns], table)
valores_por_categoria
```

```
## $term
##
## 36 months 60 months
##     28478      9867
##
## $verification_status
##
##    Not Verified Source Verified        Verified
##           16885            9321           12139
##
## $last_pymnt_d
```

```
##
## 2007-2010 2011-2013 2014-2016
##      3222     22539     12584
##
## $last_credit_pull_d
##
## 2007-2010 2011-2013 2014-2016
##      1431     11041     25873
##
## $repay_fail
##
##     0     1
## 32606  5739
```

```
#nombre de las columnas
names(loan_data)
```

```
##  [1] "id"                  "member_id"           "loan_amnt"
##  [4] "funded_amnt"         "funded_amnt_inv"     "term"
##  [7] "int_rate"            "installment"         "annual_inc"
## [10] "verification_status" "dti"                 "delinq_2yrs"
## [13] "inq_last_6mths"      "open_acc"            "pub_rec"
## [16] "revol_bal"           "revol_util"          "total_acc"
## [19] "total_pymnt"         "total_pymnt_inv"     "total_rec_prncp"
## [22] "total_rec_int"       "last_pymnt_amnt"     "repay_fail"
## [25] "last_pymnt_d"        "last_credit_pull_d"
```

```
# selecciono que columnas son numericas y cuales son categoricas
vars_num <- loan_data %>% select_if(is.numeric) %>% colnames()
vars_cat <- loan_data %>% select_if(is.character) %>% colnames()
# veo las variables numericas y categoricas (respectivamente)
vars_num
```

```
##  [1] "id"              "member_id"       "loan_amnt"       "funded_amnt"
##  [5] "funded_amnt_inv" "int_rate"        "installment"     "annual_inc"
##  [9] "dti"             "delinq_2yrs"     "inq_last_6mths"  "open_acc"
## [13] "pub_rec"         "revol_bal"       "revol_util"      "total_acc"
## [17] "total_pymnt"     "total_pymnt_inv" "total_rec_prncp" "total_rec_int"
## [21] "last_pymnt_amnt" "repay_fail"
```

```
vars_cat
```

```
## [1] "term"                "verification_status" "last_pymnt_d"
## [4] "last_credit_pull_d"
```

1. Describa cada una de las variables que componen la data e indique si corresponden a variables numéricas o categóricas. En caso de tener un número excesivo de variables (o variables irrelevantes) mantener solamente las de mayor interés.

Variables Numericas

- id: Identificador único del préstamo.
- member_id: Identificador único del miembro solicitante del préstamo.
- loan_amnt: Monto del préstamo solicitado.
- funded_amnt: Monto del préstamo financiado.
- funded_amnt_inv: Monto del préstamo financiado por los inversionistas.
- int_rate: Tasa de interés del préstamo.
- installment: Pago mensual de la cuota del préstamo.

- annual_inc: Ingreso anual del solicitante.
- dti: Ratio deuda-ingreso (Debt-to-Income Ratio) del solicitante.
- delinq_2yrs: Número de incidencias de morosidad del solicitante en los últimos 2 años.
- inq_last_6mths: Número de consultas de crédito del solicitante en los últimos 6 meses.
- open_acc: Número de cuentas abiertas del solicitante.
- pub_rec: Número de registros públicos adversos del solicitante.
- revol_bal: Saldo rotativo del solicitante.
- revol_util: Utilización del crédito rotativo del solicitante.
- total_acc: Número total de cuentas del solicitante.
- total_pymnt: Pago total realizado por el solicitante.
- total_pymnt_inv: Pago total realizado por los inversionistas.
- total_rec_prncp: Total de capital principal recibido.
- total_rec_int: Total de interés recibido.
- last_pymnt_amnt: Monto del último pago.

Variables Categoricas

- term: Plazo del préstamo.
- emp_length: Duración del empleo del solicitante.
- home_ownership: Estado de propiedad de la vivienda del solicitante.
- verification_status: Estado de verificación del ingreso del solicitante.
- loan_status: Estado del préstamo.
- purpose: Propósito del préstamo
- repay_fail: Indicador de fallo de pago (1 = fallo, 0 = no fallo).
- last_pymnt_d: Fecha del último pago.
- last_credit_pull_d: Fecha de la última consulta de crédito.

Procedo a eliminar columnas que me parecen no relevantes para el analisis y modelo de ML que implementaré para ver si un prestamo se paga o no.

```
# Elimino las columnas id, member_id, open_acc, total_acc y verification_status
loan_data <- loan_data[, -which(names(loan_data) %in% c("id"))]
loan_data <- loan_data[, -which(names(loan_data) %in% c("member_id"))]
loan_data <- loan_data[, -which(names(loan_data) %in% c("open_acc"))]
loan_data <- loan_data[, -which(names(loan_data) %in% c("total_acc"))]
loan_data <- loan_data[, -which(names(loan_data) %in% c("verification_status"))]
# Reseteo los nombres de las filas para que comiencen en 1
row.names(loan_data) <- NULL
```

La siguiente celda se utiliza para generar un resumen estadístico de las variables numéricas seleccionadas en el conjunto de datos loan_data. El resumen incluye medidas como la media, mediana, mínimo, máximo y los cuartiles (Q1 y Q3), que son útiles para entender la distribución de los datos.

```
# veo un resumen de las variables numericas
# como la media, mediana (Q2), minimo, maximo, Q1, Q3
summary(loan_data[c("loan_amnt", "funded_amnt",
                    "funded_amnt_inv", "int_rate", "installment", "annual_inc",
                    "dti", "delinq_2yrs", "inq_last_6mths",
                    "pub_rec", "revol_bal", "revol_util",
                    "total_pymnt", "total_pymnt_inv", "total_rec_prncp",
                    "total_rec_int", "last_pymnt_amnt")])
```

```
##    loan_amnt      funded_amnt    funded_amnt_inv    int_rate
##  Min.   :  500   Min.   :  500   Min.   :    0   Min.   : 5.42
##  1st Qu.: 5300   1st Qu.: 5125   1st Qu.: 4973   1st Qu.: 9.62
##  Median : 9800   Median : 9600   Median : 8500   Median :11.99
##  Mean   :11109   Mean   :10845   Mean   :10166   Mean   :12.15
```

```
##  3rd Qu.:15000   3rd Qu.:15000   3rd Qu.:14019   3rd Qu.:14.72
##  Max.   :35000   Max.   :35000   Max.   :35000   Max.   :24.11
##   installment        annual_inc           dti         delinq_2yrs
##  Min.   :  15.67   Min.   :   1896   Min.   : 0.00   Min.   : 0.0000
##  1st Qu.: 165.91   1st Qu.:  40000   1st Qu.: 8.21   1st Qu.: 0.0000
##  Median : 278.48   Median :  58880   Median :13.49   Median : 0.0000
##  Mean   : 323.52   Mean   :  69058   Mean   :13.38   Mean   : 0.1519
##  3rd Qu.: 429.86   3rd Qu.:  82100   3rd Qu.:18.69   3rd Qu.: 0.0000
##  Max.   :1305.19   Max.   :6000000   Max.   :29.99   Max.   :11.0000
##  inq_last_6mths      pub_rec         revol_bal         revol_util
##  Min.   : 0.000   Min.   :0.0000   Min.   :      0   Min.   :  0.00
##  1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.:   3667   1st Qu.: 25.70
##  Median : 1.000   Median :0.0000   Median :   8863   Median : 49.60
##  Mean   : 1.082   Mean   :0.0579   Mean   :  14309   Mean   : 49.11
##  3rd Qu.: 2.000   3rd Qu.:0.0000   3rd Qu.:  17291   3rd Qu.: 72.60
##  Max.   :33.000   Max.   :5.0000   Max.   :1207359   Max.   :119.00
##   total_pymnt       total_pymnt_inv total_rec_prncp total_rec_int
##  Min.   :   35.71   Min.   :    0   Min.   :    0   Min.   :    3.54
##  1st Qu.: 5487.53   1st Qu.: 4851   1st Qu.: 4485   1st Qu.:  662.38
##  Median : 9708.77   Median : 8981   Median : 8000   Median : 1339.16
##  Mean   :12012.95   Mean   :11307   Mean   : 9673   Mean   : 2238.91
##  3rd Qu.:16427.20   3rd Qu.:15517   3rd Qu.:13450   3rd Qu.: 2803.51
##  Max.   :58563.68   Max.   :58564   Max.   :35000   Max.   :23611.10
##  last_pymnt_amnt
##  Min.   :    0.0
##  1st Qu.:  213.8
##  Median :  528.5
##  Mean   : 2622.5
##  3rd Qu.: 3184.3
##  Max.   :36115.2
```

Medidas resumen incluidas:

- Media: El promedio de los valores en cada columna.
- Mediana (Q2): El valor central cuando los datos están ordenados.
- Mínimo: El valor más pequeño en la columna.
- Máximo: El valor más grande en la columna.
- Primer cuartil (Q1): El valor por debajo del cual se encuentra el 25% de los datos.
- Tercer cuartil (Q3): El valor por debajo del cual se encuentra el 75% de los datos.

Analisis Univariado Variables Numericas Seleccionadas

Considero que las variables numericas más importantes son 3: loan_amnt, annual_inc y dti.

```
#chequeo si hay valores faltantes en las columnas
# y cuantos son
valores_faltantes <- sapply(loan_data, function(x) sum(is.na(x)))
valores_faltantes
```

```
##          loan_amnt        funded_amnt    funded_amnt_inv               term
##                  0                  0                  0                  0
##           int_rate        installment         annual_inc                dti
##                  0                  0                  0                  0
##        delinq_2yrs     inq_last_6mths            pub_rec          revol_bal
##                  0                  0                  0                  0
##         revol_util        total_pymnt    total_pymnt_inv    total_rec_prncp
##                  0                  0                  0                  0
```

```
##      total_rec_int    last_pymnt_amnt         repay_fail       last_pymnt_d
##                  0                  0                  0                  0
## last_credit_pull_d
##                  0
```

```
# veo un resumen de las variables numericas
# como la media, mediana (Q2), minimo, maximo, Q1, Q3
summary(loan_data[c("loan_amnt", "annual_inc", "dti")])
```

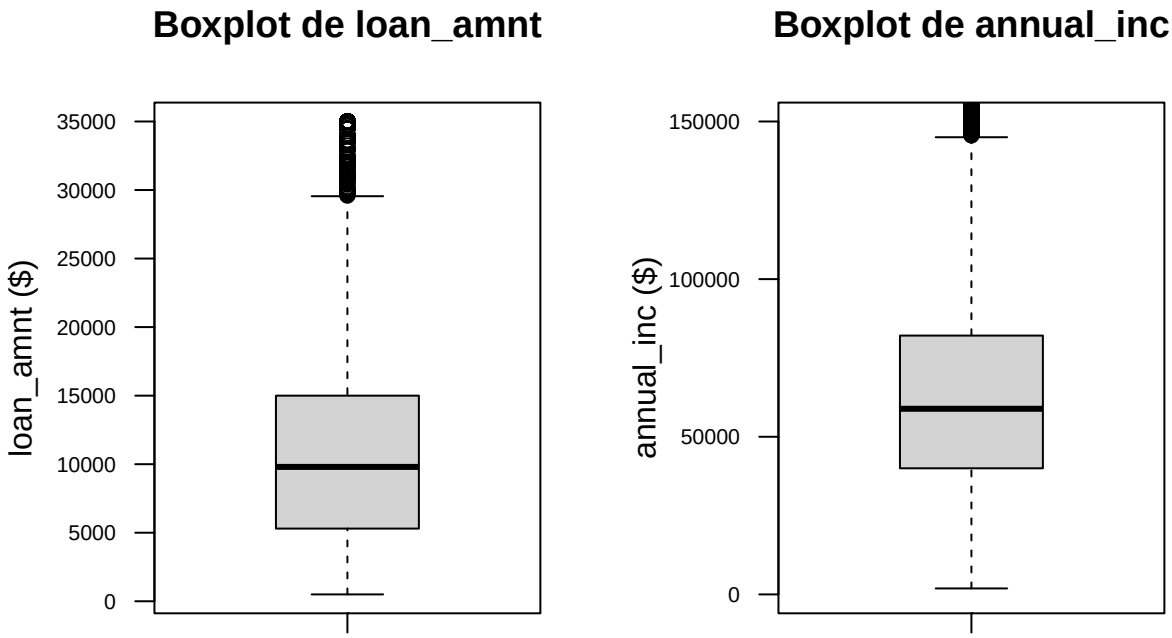
```
##    loan_amnt       annual_inc           dti
##  Min.   :  500   Min.   :   1896   Min.   : 0.00
##  1st Qu.: 5300   1st Qu.:  40000   1st Qu.: 8.21
##  Median : 9800   Median :  58880   Median :13.49
##  Mean   :11109   Mean   :  69058   Mean   :13.38
##  3rd Qu.:15000   3rd Qu.:  82100   3rd Qu.:18.69
##  Max.   :35000   Max.   :6000000   Max.   :29.99
```

```
# selecciono las columnas numericas que me interesan
# luego calculo el rango intercuartilico de cada una
col_interes_num2 <- c("loan_amnt", "annual_inc", "dti")
valores_iqr <- sapply(loan_data[col_interes_num2], IQR)
valores_iqr
```

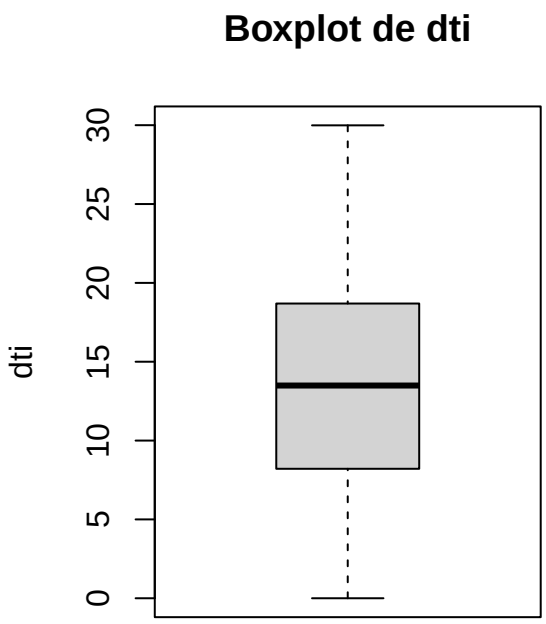
```
##  loan_amnt annual_inc        dti
##    9700.00   42100.00      10.48
```

```
# Ajusto el tamaño de la ventana gráfica
options(repr.plot.width = 10, repr.plot.height = 5)

# Divido el área de gráficos en una disposición de 1 fila por 2 columnas
par(mfrow = c(1, 2))

# Creo el primer boxplot para loan_amnt
boxplot(loan_data$loan_amnt,
        main = "Boxplot de loan_amnt",
        ylab = "loan_amnt ($)",
        xlab = "",
        outline = TRUE,
        axes = FALSE)  # Oculto ejes para personalizarlos después

# Añado de nuevo los ejes
axis(2, las = 1, cex.axis = 0.7,
     at = pretty(loan_data$loan_amnt),
     labels = format(pretty(loan_data$loan_amnt), scientific = FALSE))
axis(1, at = 1, labels = "")
box()  # Añado el cuadro alrededor del gráfico

# Creo el segundo boxplot para annual_inc con rango limitado
boxplot(loan_data$annual_inc,
        main = "Boxplot de annual_inc",
        ylab = "annual_inc ($)",
        xlab = "",
        ylim = c(0, 150000),  # Limito el rango del eje y
        outline = TRUE,
        axes = FALSE)  # Oculto ejes para personalizarlos después
```

```r
# Añado de nuevo los ejes
axis(2, las = 1, cex.axis = 0.7,
     at = pretty(loan_data$annual_inc[loan_data$annual_inc <= 150000]),
     labels = format(pretty(loan_data$annual_inc
                            [loan_data$annual_inc <= 150000]),
                     scientific = FALSE))
axis(1, at = 1, labels = "")
box()  # Añado el cuadro alrededor del gráfico
```




```r
# Creo un boxplot para dti
par(mfrow = c(1, 2))  # Configuro el área de gráficos para un solo gráfico
boxplot(loan_data$dti,
        main = "Boxplot de dti",
        ylab = "dti",
        xlab = "",
        outline = TRUE)  # Mantengo los outliers
```




```r
# Cálculo del rango intercuartílico de loan_amnt
q1 <- quantile(loan_data$loan_amnt, 0.25)
q3 <- quantile(loan_data$loan_amnt, 0.75)
iqr <- q3 - q1
lim_inf <- q1 - 1.5 * iqr
lim_sup <- q3 + 1.5 * iqr
# Identifico los outliers
outliers <- loan_data %>%
  filter(loan_amnt < lim_inf | loan_amnt > lim_sup)
# Número de outliers
nrow(outliers)
```

```
## [1] 1111
```

```r
# Cálculo del rango intercuartílico de annual_inc
q1 <- quantile(loan_data$annual_inc, 0.25)
q3 <- quantile(loan_data$annual_inc, 0.75)
iqr <- q3 - q1
lim_inf <- q1 - 1.5 * iqr
lim_sup <- q3 + 1.5 * iqr
# Identifico los outliers
outliers <- loan_data %>%
  filter(annual_inc < lim_inf | annual_inc > lim_sup)
# Número de outliers
nrow(outliers)
```

```
## [1] 1838
```

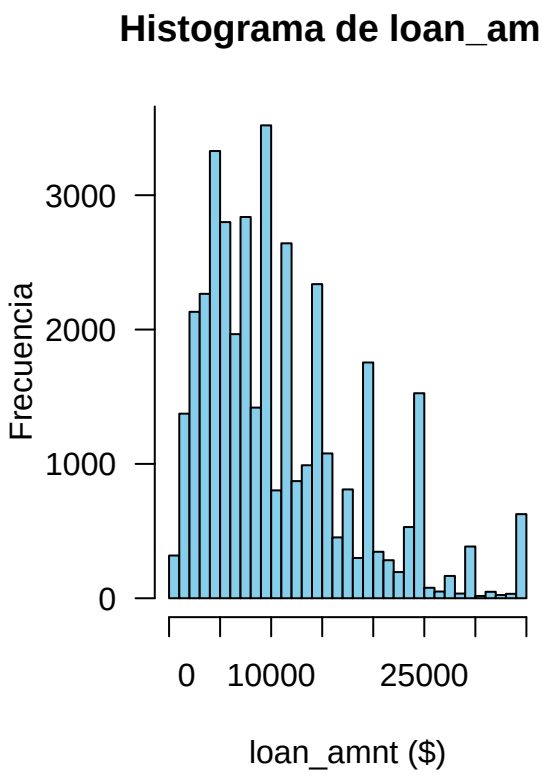
```r
#mediana y promedio de annual_inc
median(loan_data$annual_inc)
```

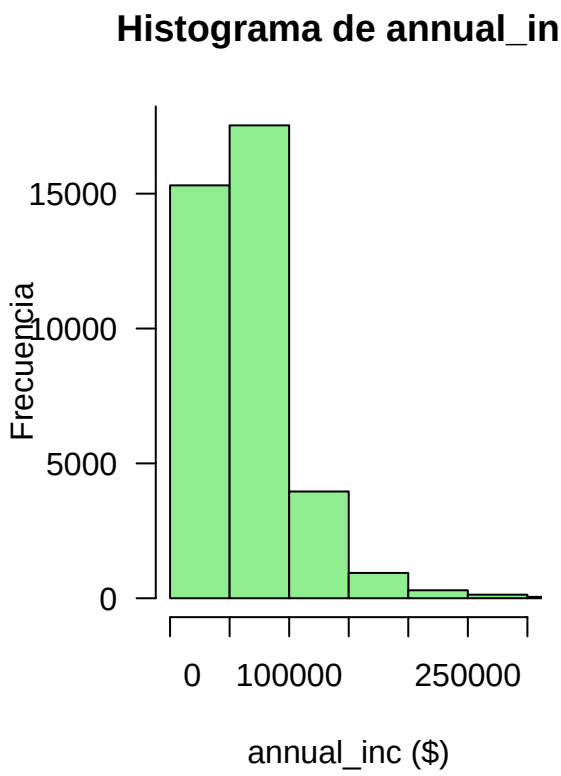
```
## [1] 58880
```

```r
mean(loan_data$annual_inc)
```

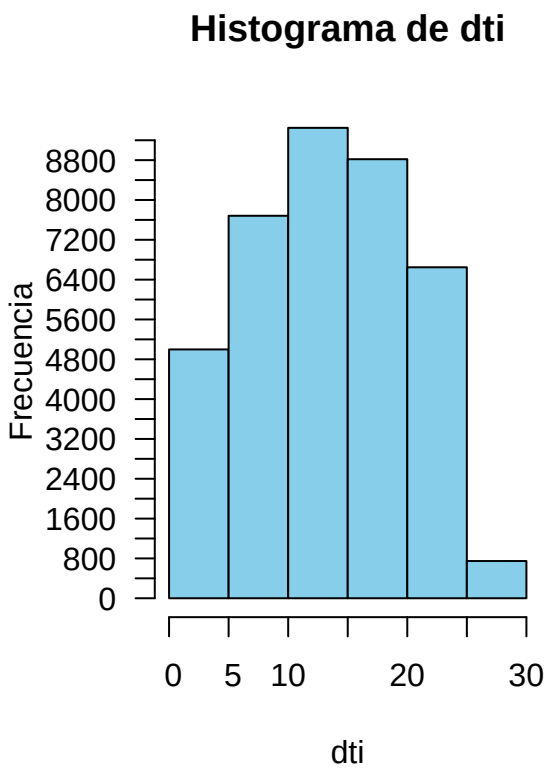
```
## [1] 69057.61
```

```r
# Ajusto el tamaño de la ventana gráfica
options(repr.plot.width = 10, repr.plot.height = 5)

# Divido el área de gráficos en una disposición de 1 fila por 2 columnas
par(mfrow = c(1, 2))

# Creo el histograma para loan_amnt
hist(loan_data$loan_amnt,
     main = "Histograma de loan_amnt",
     xlab = "loan_amnt ($)",
     ylab = "Frecuencia",
     col = "skyblue",
     breaks = 30,  # Ajusto el número de bins
     axes = FALSE)  # Oculto ejes para personalizarlos después

# Añado los ejes personalizados para loan_amnt
axis(1, at = pretty(loan_data$loan_amnt),
     labels = format(pretty(loan_data$loan_amnt), scientific = FALSE))
axis(2, las = 1, at = pretty(range(hist(loan_data$loan_amnt, breaks = 30, plot = FALSE)$counts)))

# Creo el histograma para annual_inc con más bins
hist(loan_data$annual_inc,
     main = "Histograma de annual_inc",
     xlab = "annual_inc ($)",
     ylab = "Frecuencia",
     col = "lightgreen",
     breaks = 100,  # Incremento el número de bins
     xlim = c(0, 300000),  # Limito el rango del eje x
     axes = FALSE)  # Oculto ejes para personalizarlos después

# Añado los ejes personalizados para annual_inc
axis(1, at = pretty(loan_data$annual_inc[loan_data$annual_inc <= 300000]),
     labels = format(pretty(loan_data$annual_inc[loan_data$annual_inc <= 300000]), scientific = FALSE))
axis(2, las = 1, at = pretty(range(hist(loan_data$annual_inc, breaks = 100, plot = FALSE)$counts)))
```






```r
# Configuro el área de gráficos para un solo gráfico
par(mfrow = c(1, 2))
# Obtengo las frecuencias del histograma de dti sin dibujarlo
hist_info_dti <- hist(loan_data$dti, breaks = 10, plot = FALSE)

# Calculo la frecuencia máxima para dti
max_freq_dti <- max(hist_info_dti$counts)

# Creo el histograma para dti con el ajuste del eje y
hist(loan_data$dti,
     main = "Histograma de dti",
     xlab = "dti",
     ylab = "Frecuencia",
     col = "skyblue",
     breaks = 10,
     ylim = c(0, max_freq_dti + 50),  # Ajusto el rango del eje y
     axes = FALSE)  # Oculto ejes para personalizarlos después

# Añado los ejes personalizados para dti
axis(1, at = hist_info_dti$breaks, labels =
       format(hist_info_dti$breaks, scientific = FALSE))
axis(2, las = 1, at = seq(0, max_freq_dti + 50, by = 400))
```




```r
# Varianza y desviación estándar de loan_amnt
var_loan_amnt <- var(loan_data$loan_amnt)
sd_loan_amnt <- sd(loan_data$loan_amnt)
var_loan_amnt
```

```
## [1] 54811627
```

```r
sd_loan_amnt
```

```
## [1] 7403.487
```

```r
# Varianza y desviación estándar de annual_inc
var_annual_inc <- var(loan_data$annual_inc)
sd_annual_inc <- sd(loan_data$annual_inc)
var_annual_inc
```

```
## [1] 4160751266
```

```r
sd_annual_inc
```

```
## [1] 64503.89
```

```r
# Varianza y desviación estándar de dti
var_dti <- var(loan_data$dti)
sd_dti <- sd(loan_data$dti)
var_dti
```

```
## [1] 45.25706
```

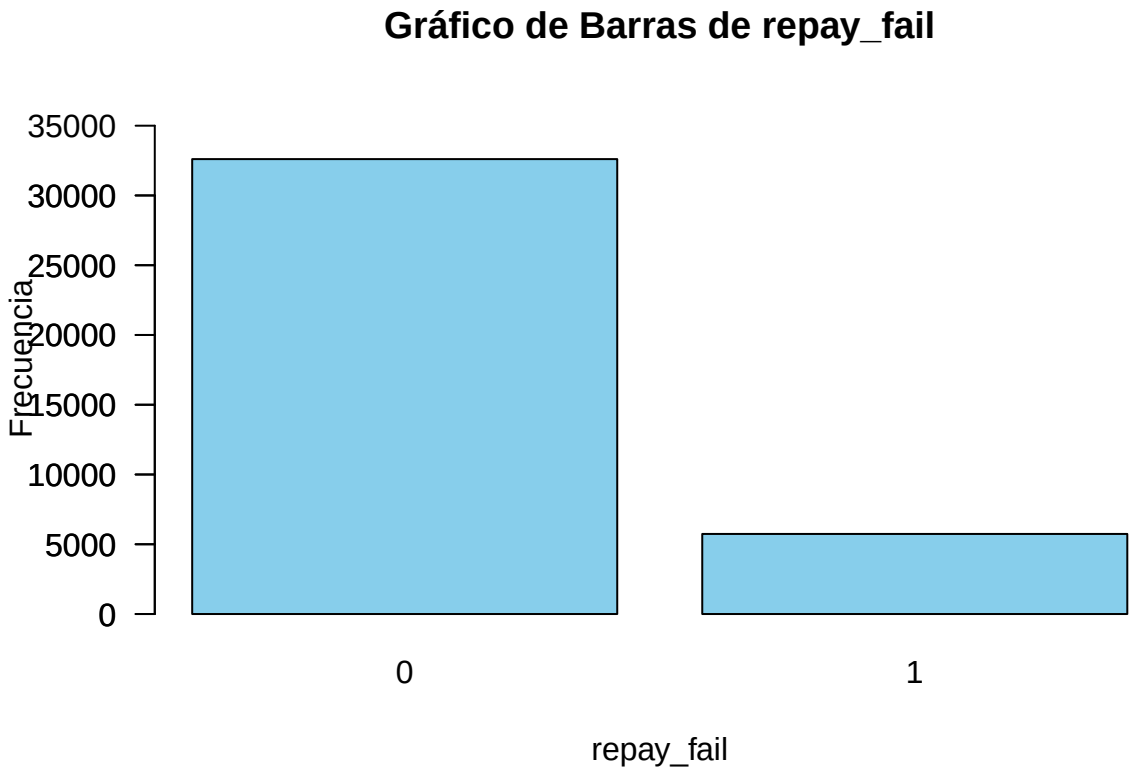
```r
sd_dti
```

```
## [1] 6.727337
```

Analisis Univariado Variables Categoricas Seleccionadas

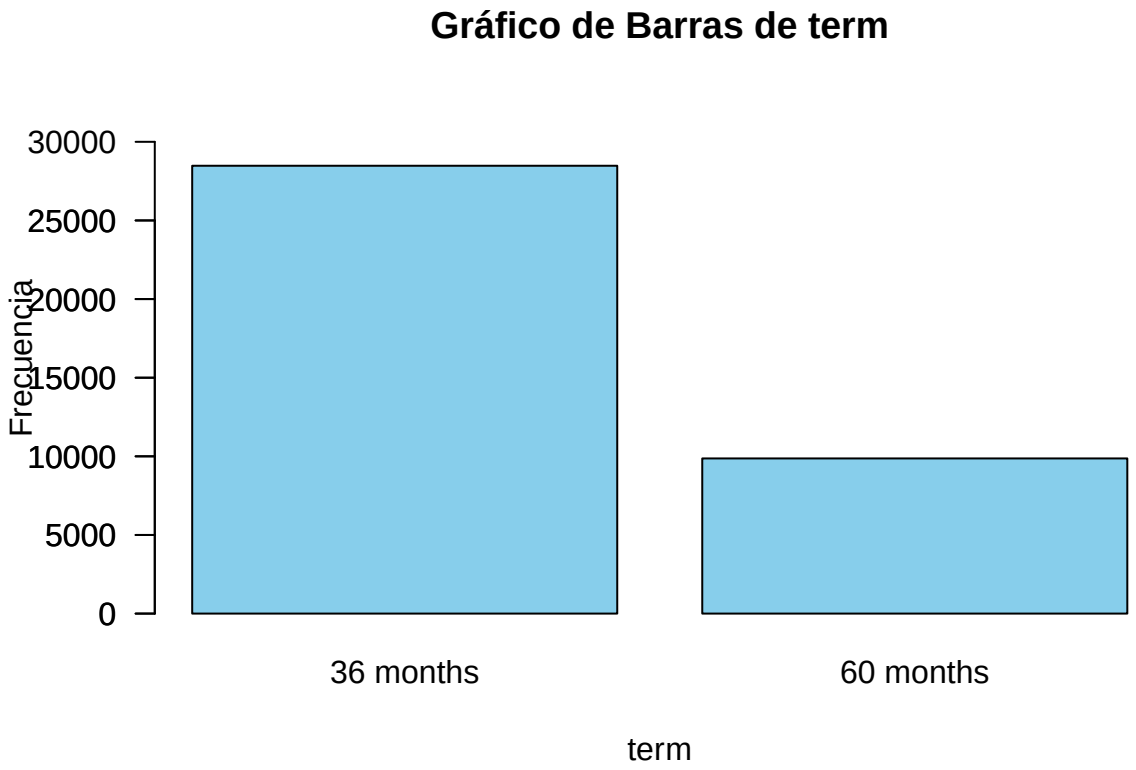
Considero que las variables categoricas más importantes son 2: repay_fail y term

```r
# Creo la tabla de frecuencias para repay_fail
freq_table_repay_fail <- table(loan_data$repay_fail)

# Creo el gráfico de barras
barplot(freq_table_repay_fail,
        main = "Gráfico de Barras de repay_fail",
        xlab = "repay_fail",
        ylab = "Frecuencia",
        col = "skyblue",
        ylim = c(0, max(freq_table_repay_fail) + 4000),
        las = 1)  # Hacer que las etiquetas del eje y sean horizontales

# Añado más etiquetas en el eje y
y_ticks <- seq(0, max(freq_table_repay_fail) + 50, by = 5000)
axis(2, at = y_ticks, labels = y_ticks, las = 1)
```



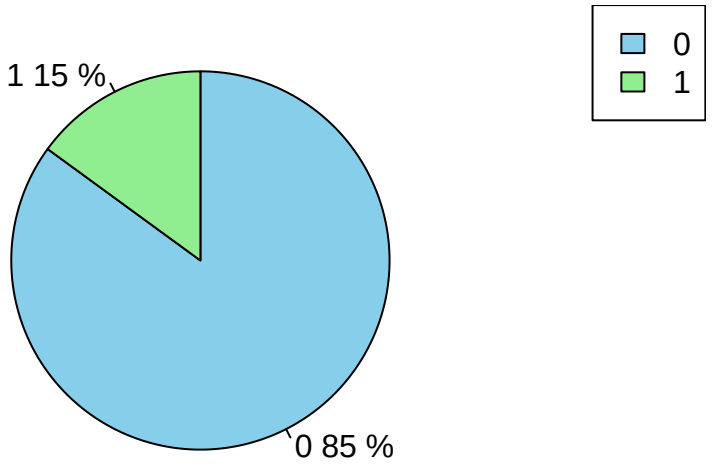

```r
# Creo la tabla de frecuencias para term
freq_table_term <- table(loan_data$term)

# Creo el gráfico de barras
barplot(freq_table_term,
        main = "Gráfico de Barras de term",
        xlab = "term",
        ylab = "Frecuencia",
        col = "skyblue",
        ylim = c(0, max(freq_table_term) + 4000),
        las = 1)  # Hago que las etiquetas del eje y sean horizontales
```

```r
# Añado más etiquetas en el eje y
y_ticks <- seq(0, max(freq_table_term) + 50, by = 5000)
axis(2, at = y_ticks, labels = y_ticks, las = 1)
```



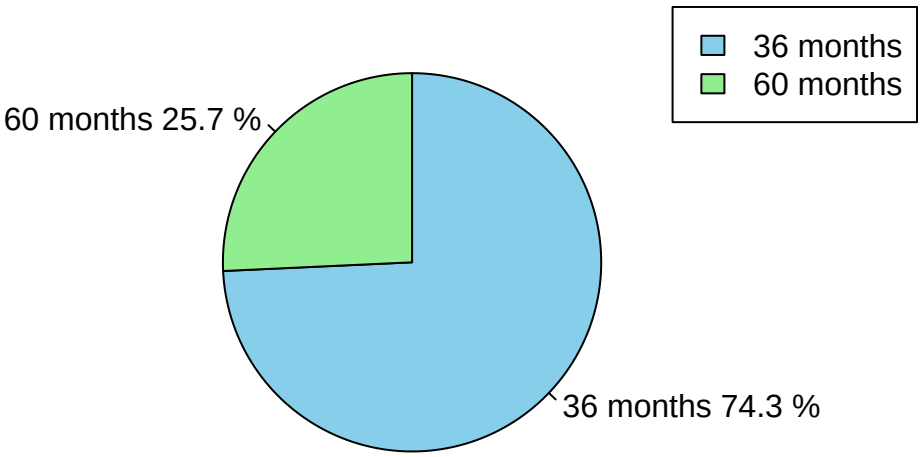

```r
# Creo la tabla de frecuencias para repay_fail
freq_table_repay_fail <- table(loan_data$repay_fail)

# Calculo los porcentajes
percentages <- round(100 * freq_table_repay_fail / sum(freq_table_repay_fail), 1)
labels <- paste(names(freq_table_repay_fail), percentages, "%")

# Creo el gráfico de torta
pie(freq_table_repay_fail,
    main = "Gráfico de Torta de repay_fail",
    col = c("skyblue", "lightgreen"),
    labels = labels,
    clockwise = TRUE)

# Añado leyenda
legend("topright", legend = names(freq_table_repay_fail), fill = c("skyblue", "lightgreen"))
```

Gráfico de Torta de repay_fail




```
# Creo la tabla de frecuencias para term
freq_table_term <- table(loan_data$term)

# Calculo los porcentajes
percentages <- round(100 * freq_table_term / sum(freq_table_term), 1)
labels <- paste(names(freq_table_term), percentages, "%")

# Creo el gráfico de torta
pie(freq_table_term,
    main = "Gráfico de Torta de term",
    col = c("skyblue", "lightgreen"),
    labels = labels,
    clockwise = TRUE)

# Añado leyenda
legend("topright", legend = names(freq_table_term), fill = c("skyblue", "lightgreen"))
```

Gráfico de Torta de term




```
## Creo la tabla de frecuencias para repay_fail
freq_table_repay_fail <- table(loan_data$repay_fail)
```

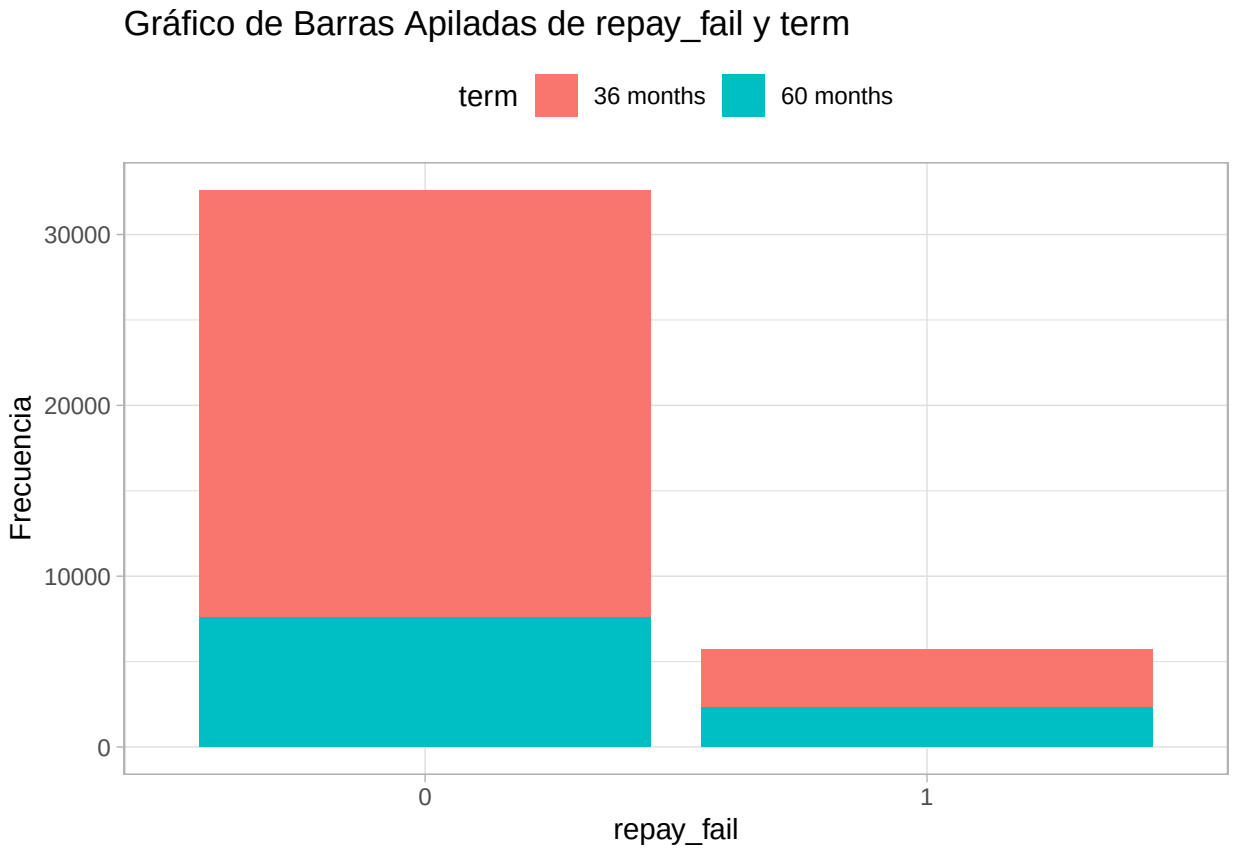
```
freq_table_repay_fail
```

```
##
##     0     1
## 32606  5739
```

```
## Creo la tabla de frecuencias para term
freq_table_term <- table(loan_data$term)
freq_table_term
```

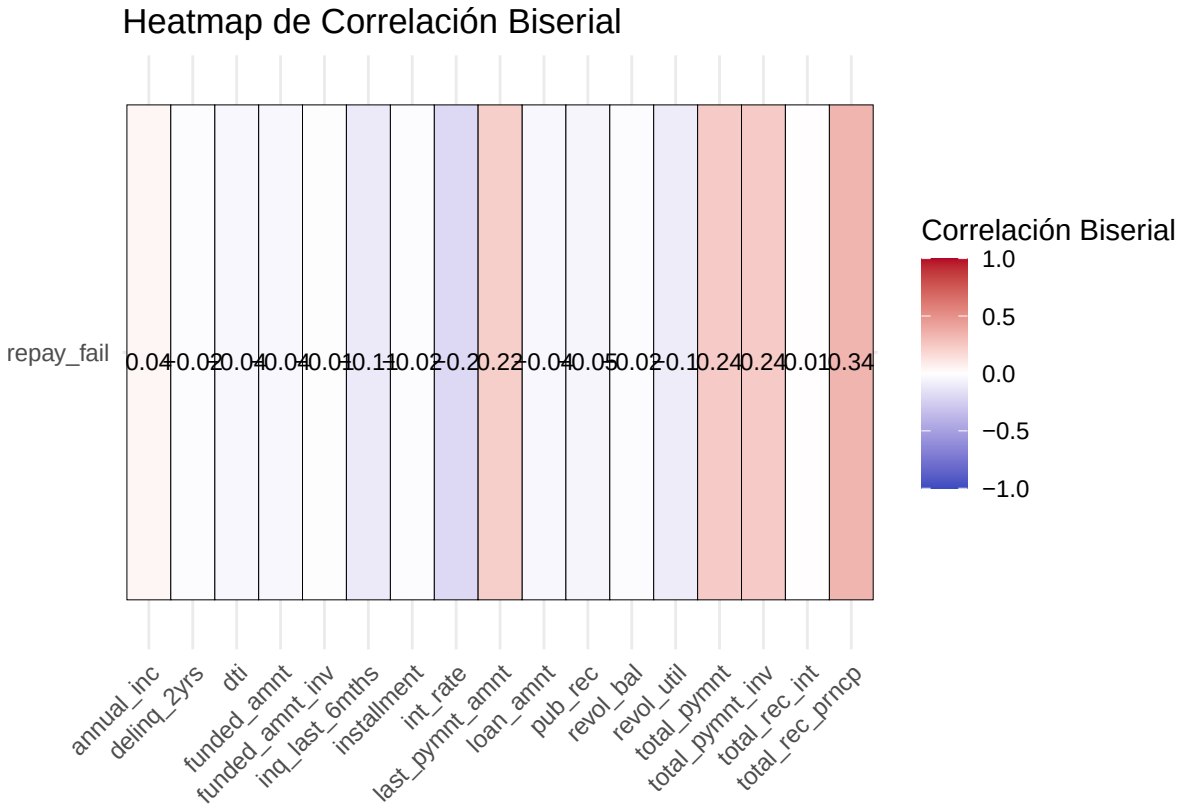
```
##
## 36 months 60 months
##     28478      9867
```

```
# Creo el gráfico de barras apiladas usando ggplot2
# para las variables repay_fail y purpose
ggplot(loan_data, aes(x = factor(repay_fail), fill = factor(term))) +
  geom_bar(position = "stack") +
  labs(title = "Gráfico de Barras Apiladas de repay_fail y term",
       x = "repay_fail",
       y = "Frecuencia",
       fill = "term") +
  theme_light() +
  theme(legend.position = "top")
```



La siguiente celda tiene como objetivo calcular la correlación biserial entre las variables numéricas de un conjunto de datos y la variable de interés (repay_fail), y luego visualizar estas correlaciones utilizando un heatmap.

```r
# Selecciono solo las columnas numéricas
numeric_vars <- sapply(loan_data, is.numeric)
datos_numericos <- loan_data[, numeric_vars]

# Añado 'repay_fail' a los datos numéricos
datos_numericos$repay_fail <- loan_data$repay_fail

# creo función para calcular la correlación biserial
cor_biserial <- function(x, y) {
  biserial.cor(x, y)
}

# Calculo la correlación biserial entre cada variable numérica y 'repay_fail'
correlaciones <- sapply(datos_numericos[, -ncol(datos_numericos)], cor_biserial, y = datos_numericos$re

# Convierto las correlaciones a un data frame para ggplot
correlaciones_df <- data.frame(Variable = names(correlaciones), Correlacion = correlaciones)

# Ajusto tamaño de etiquetas para mejorar la visualización
correlaciones_df$Correlacion <- round(correlaciones_df$Correlacion, 2)

# Creo el heatmap con etiquetas mejoradas
ggplot(correlaciones_df, aes(x = Variable, y = "repay_fail", fill = Correlacion)) +
  geom_tile(color = "black") +
  geom_text(aes(label = Correlacion), vjust = 1, color = "black", size = 3) +
  scale_fill_gradient2(low = "#3B4CC0", mid = "#FFFFFF", high = "#B40426",
                       midpoint = 0, limit = c(-1, 1), space = "Lab",
                       name = "Correlación Biserial") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(x = "", y = "", title = "Heatmap de Correlación Biserial")
```



Propósito y Utilidad

El propósito del código es identificar qué variables numéricas tienen una relación significativa con la variable repay_fail. Esta información es crucial para:

Selección de Características:

Identificar las variables más relevantes para incluir en el modelo de Naive Bayes.

Preprocesamiento:

Determinar qué variables podrían necesitar transformaciones adicionales (como binning) antes de ser utilizadas en el modelo.

La siguiente celda se encarga de eliminar las variables que tienen una baja correlación con la variable objetivo repay_fail.

```
#por ende procedo a eliminar las variables que tienen baja
# correlación con la variable objetivo
#osea elimino funded_amnt_inv, installment, revol_bal, total_rec_int
# delinq_2yrs, funded_amnt, pub_rec, inq_last_6mths, loan_amnt, revol_util
loan_data <- loan_data %>%
  dplyr::select(-funded_amnt_inv, -installment, -revol_bal,
                -total_rec_int, -delinq_2yrs, -funded_amnt, -pub_rec,
                -inq_last_6mths, -loan_amnt, -revol_util)
```

Este paso se enfoca en eliminar variables que no aportan información significativa en relación con la variable objetivo repay_fail. Reducir el número de variables simplifica el modelo y mejora su rendimiento, evitando el sobreajuste.

Mejora en la Eficiencia del Modelo:

Al eliminar las variables irrelevantes, el modelo de Naive Bayes podrá centrarse en las variables que realmente tienen una influencia significativa en la predicción de repay_fail.

```
#creo 2 copias de loan_data, una para el modelo de
# k-NN de clasificación y otra para el modelo de naives bayes
loan_data_knn <- loan_data
loan_data2 <- loan_data
```

```
# Elimino todas las variables excepto loan_data_knn y loan_data2
rm(list = setdiff(ls(), c("loan_data_knn", "loan_data2")))
```

Analisis k-NN de Clasificacion

```
# Selecciono las columnas categóricas específicas
col_cate <- c("term", "last_pymnt_d", "last_credit_pull_d")

# Calculo los valores por clase para las columnas especificadas en loan_data_knn
valores_por_clase <- lapply(loan_data_knn[col_cate], table)

# Muestro los valores por clase
valores_por_clase
```

```
## $term
##
## 36 months 60 months
##     28478      9867
##
## $last_pymnt_d
##
## 2007-2010 2011-2013 2014-2016
##      3222     22539     12584
##
## $last_credit_pull_d
##
## 2007-2010 2011-2013 2014-2016
##      1431     11041     25873
```

Transformaciones para el Modelo k-NN

El objetivo principal del siguiente código es preparar las variables categóricas en el conjunto de datos para que puedan ser utilizadas por el algoritmo k-NN. Esto se logra convirtiendo las variables categóricas en variables dummy (también conocidas como variables indicadoras), un proceso crucial porque el k-NN requiere que todas las variables de entrada sean numéricas para calcular distancias entre observaciones.

```
# las variables categoricas las paso a dummies con clases n-1
# Especifico las columnas a convertir
col_cate <- c("term", "last_pymnt_d", "last_credit_pull_d")

# Defino el objeto con las variables categóricas a convertir, usando
# solo las columnas especificadas de loan_data_knn
dv <- dummyVars("~ .", data = loan_data_knn[, col_cate], fullRank = TRUE)

# Creo el diseño de la matriz con las columnas especificadas
design_matrix <- data.frame(predict(dv, newdata = loan_data_knn[, col_cate]))

#reviso el nombre de las columnas de mi diseño de matriz
colnames(design_matrix)
```

```
## [1] "term60.months"               "last_pymnt_d2011.2013"
## [3] "last_pymnt_d2014.2016"       "last_credit_pull_d2011.2013"
## [5] "last_credit_pull_d2014.2016"
```

Razón para Usar n-1 Clases al Convertir Columnas Categóricas a Dummies

Al convertir variables categóricas en variables dummy, se utiliza n-1 clases para evitar la colinealidad. La colinealidad ocurre cuando una variable puede ser perfectamente predicha por una combinación lineal de otras variables, lo cual puede causar problemas en muchos modelos de machine learning y análisis estadístico.

Por ejemplo, si una variable categórica tiene tres niveles: A, B y C, la conversión a variables dummy creará tres columnas: A, B y C. Si se incluyen las tres columnas en el modelo, la suma de las tres será siempre 1 (ya que cada observación pertenecerá a exactamente uno de los niveles), lo que introduce una dependencia perfecta entre las variables. Para evitar esto, se elimina una de las columnas (por ejemplo, la columna C), dejando solo las columnas A y B. Esto permite que el modelo utilice las variables dummy sin introducir colinealidad.

Relevancia en el Contexto del Algoritmo k-NN

El algoritmo k-NN utiliza distancias para identificar los vecinos más cercanos y, por lo tanto, requiere que todas las variables de entrada sean numéricas. Las variables categóricas, si no se transforman, no pueden ser utilizadas directamente para calcular estas distancias. Al convertir las variables categóricas en dummies, garantizamos que el algoritmo pueda considerar todas las características relevantes en sus cálculos de distancia.

```
# Elimino las columnas originales de loan_data_knn
loan_data_knn <- loan_data_knn[, -which(names(loan_data_knn) %in% col_cate)]
# Agrego las columnas de diseño de matriz a loan_data_knn
loan_data_knn <- cbind(loan_data_knn, design_matrix)
# compruebo que las columnas se han añadido correctamente
dim(loan_data_knn)
```

```
## [1] 38345    13
```

Ahora eliminé las columnas originales de las cuales transformé a columnas con valores dummies

La siguientes 2 celdas normalizan todas las variables en el conjunto de datos para que estén en un rango de 0 a 1. Este paso es crucial para el algoritmo k-NN, ya que las diferentes escalas de las variables pueden afectar los cálculos de distancia y, por lo tanto, la precisión del modelo.

```
# Función de normalización
normalizar <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
```

```
# Normalizo todas las columnas de loan_data_knn
loan_data_knn <- as.data.frame(lapply(loan_data_knn, normalizar))
# verifico que las columnas se han normalizado correctamente
# Quiero verificar que todos mis columnas tengan como mínimo 0 y como máximo 1
summary(loan_data_knn)[c(1, 6), ]
```

```
##     int_rate        annual_inc            dti          total_pymnt
##  Min.   :0.0000   Min.   :0.000000   Min.   :0.0000   Min.   :0.00000
##  Max.   :1.0000   Max.   :1.000000   Max.   :1.0000   Max.   :1.00000
##  total_pymnt_inv   total_rec_prncp  last_pymnt_amnt      repay_fail
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.000000   Min.   :0.0000
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.000000   Max.   :1.0000
##  term60.months    last_pymnt_d2011.2013 last_pymnt_d2014.2016
##  Min.   :0.0000   Min.   :0.0000        Min.   :0.0000
##  Max.   :1.0000   Max.   :1.0000        Max.   :1.0000
##  last_credit_pull_d2011.2013 last_credit_pull_d2014.2016
##  Min.   :0.0000              Min.   :0.0000
##  Max.   :1.0000              Max.   :1.0000
```

El algoritmo k-NN calcula distancias entre observaciones para identificar los vecinos más cercanos. Si las variables tienen diferentes escalas, aquellas con valores más grandes pueden dominar los cálculos de distancia, lo que puede llevar a resultados sesgados. Normalizar las variables asegura que cada una contribuya de manera equitativa a los cálculos de distancia, mejorando la precisión del modelo.

La siguiente celda convierte la variable objetivo repay_fail en un factor. Este paso es crucial para los

algoritmos de clasificación como k-NN, que requiere que la variable objetivo esté en formato categórico (factor) para realizar predicciones correctamente

```
#quiero dejar la variable target como factor
# Convierto 'repay_fail' a formato factor
loan_data_knn$repay_fail <- as.factor(loan_data_knn$repay_fail)
```

El objetivo de la siguiente celda de código es dividir el conjunto de datos en conjuntos de entrenamiento y prueba, asegurando que la proporción de la variable objetivo (repay_fail) se mantenga igual en ambos conjuntos. Este paso es crucial para evaluar correctamente el rendimiento del modelo, ya que garantiza que ambos conjuntos sean representativos de la distribución original de los datos.

```
### Splitting inicial de los datos
library(rsample)
set.seed(123) # Semilla
# Muestreo Estratificado
split <- initial_split(loan_data_knn, prop = 0.7, strata = "repay_fail")
loan_data_knn_train <- training(split)
loan_data_knn_test <- testing(split)

# Compruebo proporción de "repay_fail"
round(prop.table(table(loan_data_knn_train$repay_fail)), 2)
```

```
##
##    0    1
## 0.85 0.15
```

```
round(prop.table(table(loan_data_knn_test$repay_fail)), 2)
```

```
##
##    0    1
## 0.85 0.15
```

```
# Vectores variable objetivo
repay_fail_train <- loan_data_knn_train$repay_fail
repay_fail_test  <- loan_data_knn_test$repay_fail
```

La división adecuada de los datos es esencial para evaluar el rendimiento de cualquier modelo de machine learning, incluido k-NN. Usar muestreo estratificado asegura que la proporción de clases en la variable objetivo se mantenga constante entre los conjuntos de entrenamiento y prueba, lo que es crucial para obtener una evaluación precisa del modelo. Esto ayuda a prevenir problemas como el sesgo en la evaluación del rendimiento del modelo.

Los resultados obtenidos muestran que tanto el conjunto de entrenamiento como el de prueba tienen una proporción de 85% para la clase 0 y 15% para la clase 1, lo cual indica que la estratificación se ha realizado correctamente.

1era forma: Evaluación del Modelo k-NN sin busqueda de Parámetros

1era forma: Paquete kknn que permite configuracion de parametros k, distancia y kernel, para esta version se colocarán parametros suboptimos de forma manual, sin saber hiperparametro y parametros optimos que se obtendran mediante remuestreo y grilla, lo que se realizará posteriormente en la 2da forma.

1.1 Evaluacion del modelo, metrica de accuracy, con output un vector de niveles

El objetivo del siguiente trozo de código es evaluar el rendimiento de un modelo k-NN de clasificación utilizando el paquete kknn en R. En esta primera forma, se configuran manualmente algunos parámetros del modelo (k, distancia y kernel) para realizar una evaluación preliminar mediante la métrica de precisión (accuracy).

```
# Evaluación del modelo con kknn
set.seed(12345)
kknn_level <- kknn(repay_fail ~ ., loan_data_knn_train,
                   loan_data_knn_test
                   [, colnames(loan_data_knn_test) != "repay_fail"],
                   k = 7, distance = 2, kernel = "optimal")
kknn_level <- kknn_level$fitted.values

# Resultados
results <- confusionMatrix(kknn_level, repay_fail_test)
results$table
```

```
##            Reference
## Prediction    0    1
##          0 9395  674
##          1  387 1048
```

```
results$overall[names(results$overall) == "Accuracy"]
```

```
##  Accuracy
## 0.9077712
```

Configuración Inicial y Entrenamiento del Modelo:

Se establece una semilla para asegurar la reproducibilidad de los resultados.

Se entrena el modelo k-NN con parámetros configurados manualmente:

- k=7: número de vecinos.
- distance=2: distancia euclidiana.
- kernel="optimal": tipo de kernel utilizado.

Se obtienen los valores predichos del modelo.

Evaluación del Modelo:

- Se utiliza la matriz de confusión para evaluar el rendimiento del modelo.
- Se calcula la precisión (accuracy) del modelo, que es la proporción de predicciones correctas sobre el total de predicciones.
- **True Negatives (TN)**: Indican el número de clientes correctamente identificados como no fallidos en el pago del préstamo. Esto es crucial para evitar la pérdida de buenos clientes.
- **False Positives (FP)**: Indican el número de clientes incorrectamente identificados como fallidos en el pago del préstamo. Este error puede llevar a la pérdida de clientes potencialmente valiosos.

- **False Negatives (FN)**: Indican el número de clientes incorrectamente identificados como no fallidos en el pago del préstamo. Este error es especialmente costoso, ya que representa clientes que representan un riesgo pero no fueron identificados como tales.
- **True Positives (TP)**: Indican el número de clientes correctamente identificados como fallidos en el pago del préstamo. Identificar correctamente estos clientes ayuda a mitigar riesgos financieros.

Interpretación de los Resultados

Matriz de Confusión:

- **True Negatives (TN)**: 9395 (Clientes predichos correctamente como que no fallarán en el pago del préstamo).
- **False Positives (FP)**: 387 (Clientes predichos incorrectamente como que fallarán en el pago del préstamo, pero en realidad no fallarán).
- **False Negatives (FN)**: 674 (Clientes predichos incorrectamente como que no fallarán en el pago del préstamo, pero en realidad sí fallarán).
- **True Positives (TP)**: 1048 (Clientes predichos correctamente como que fallarán en el pago del préstamo).

Precisión (Accuracy):

- La precisión del modelo es aproximadamente 90.78%, lo que indica que el modelo clasifica correctamente el 90.78% de las observaciones en el conjunto de prueba.

1.2 Evaluacion de modelo mediante ROC, con output un vector de probabilidades

La siguiente celda evalua el rendimiento de un modelo k-NN de clasificación utilizando la curva ROC y las probabilidades de pertenencia a las clases. Esto permite una evaluación más detallada y precisa del modelo comparada con solo usar la matriz de confusión y la precisión (accuracy).

```
# Evaluación del modelo con kknn y salida de probabilidades
set.seed(12345)
kknn_prob <- kknn(repay_fail ~ ., loan_data_knn_train,
                  loan_data_knn_test
                  [, colnames(loan_data_knn_test) != "repay_fail"],
                  k = 7, distance = 2, kernel = "optimal")
kknn_prob <- kknn_prob$prob
kknn_prob <- as.numeric(kknn_prob[, colnames(kknn_prob) == "1"])
```

```
# Umbral nivel/probabilidad
level_prob <- data.frame(kknn_level, kknn_prob)
level_prob <- min(level_prob[level_prob$kknn_level == "1", "kknn_prob"])
level_prob
```

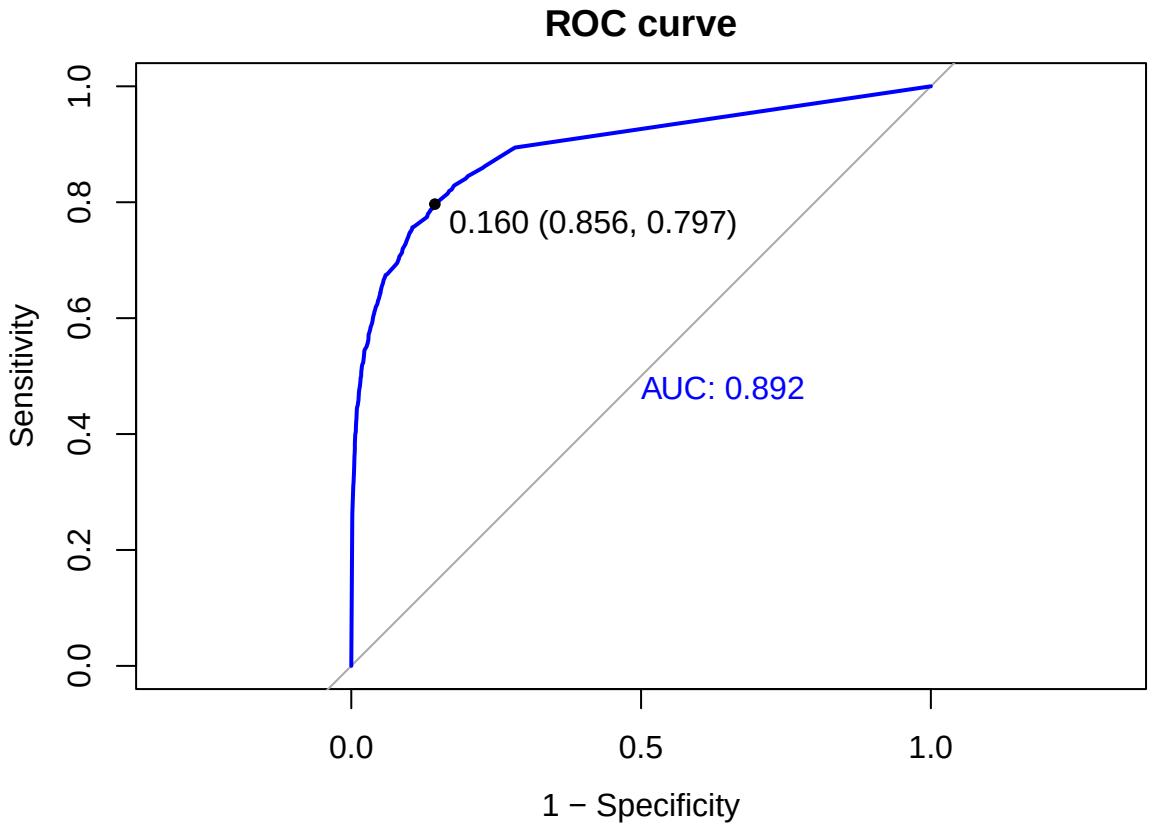
```
## [1] 0.5022038
```

Umbral de Probabilidad:

El umbral calculado (0.502) indica la probabilidad mínima a partir de la cual las observaciones se clasifican como Class_1 (clientes que fallarán en el pago del préstamo). Este umbral se puede ajustar para mejorar la sensibilidad o la especificidad del modelo, osea que ajustar este umbral puede ayudar a balancear la tasa de falsos positivos y falsos negativos, lo cual es crucial para la toma de decisiones en un contexto financiero.

El objetivo del siguiente código es evaluar el rendimiento del modelo k-NN utilizando la curva ROC para determinar el punto de corte óptimo que maximiza la suma de sensibilidad y especificidad. Luego, se convierte el vector de probabilidades en clases binarias usando este punto de corte óptimo y se evalúa el rendimiento del modelo mediante la matriz de confusión.

```
# ROC
knn_roc_curve <- roc(repay_fail_test, knn_prob,
                     levels = c("0", "1"), direction = "<")
plot.roc(knn_roc_curve, main = "ROC curve", col = "blue", lwd = 2,
         legacy.axes = TRUE, print.thres = "best", print.auc = TRUE)
```




```
# Calculo el punto de corte óptimo basado en la suma
# de sensibilidades y especificidades
aux <- knn_roc_curve$sensitivities + knn_roc_curve$specificities
corte <- knn_roc_curve$thresholds[which(aux == max(aux))]

# Convierto las probabilidades a clases binarias usando el punto de corte óptimo
knn_prob <- ifelse(knn_prob < corte, 0, 1)
knn_prob <- factor(knn_prob, levels = c(0, 1),
                   labels = c("0", "1"))

# Resultados
results <- confusionMatrix(knn_prob, repay_fail_test)
results$table
```

```
##            Reference
## Prediction    0    1
##          0 8371  350
##          1 1411 1372
```

```
sensitivity(knn_prob, repay_fail_test) +
  specificity(knn_prob, repay_fail_test)
```

```
## [1] 1.652503
```

Interpretación de los Resultados Matriz de Confusión:

- True Negatives (TN): 8371 (Clientes predichos correctamente como que no fallarán en el pago del préstamo).
- False Positives (FP): 1411 (Clientes predichos incorrectamente como que fallarán en el pago del préstamo, pero en realidad no fallarán).
- False Negatives (FN): 350 (Clientes predichos incorrectamente como que no fallarán en el pago del préstamo, pero en realidad sí fallarán).
- True Positives (TP): 1372 (Clientes predichos correctamente como que fallarán en el pago del préstamo).

Curva ROC y AUC:

- La curva ROC muestra la relación entre la tasa de verdaderos positivos (sensibilidad) y la tasa de falsos positivos (1 - especificidad).
- El área bajo la curva (AUC) es 0.892, lo que indica una buena capacidad del modelo para distinguir entre clientes que fallarán y no fallarán en el pago del préstamo.
- La sensibilidad (True Positive Rate) es la proporción de verdaderos positivos correctamente identificados por el modelo.
- La especificidad (True Negative Rate) es la proporción de verdaderos negativos correctamente identificados por el modelo.

Suma de Sensibilidad y Especificidad:

- La suma de sensibilidad y especificidad es 1.65, lo que refleja un buen equilibrio entre ambos aspectos en el punto de corte óptimo.

```
# Elimino objetos
rm(kknn_level, kknn_prob, level_prob, results, kknn_roc_curve, aux, corte)
```

2da forma: Evaluación del Modelo k-NN con busqueda de Parámetros

2da forma: Paquete kknn que permite configuracion de parametros k, distancia y kernel, para esta version se buscarán los mejores 3 parametros (k, distancia y kernel), para estos efectos se utilizará el remuestreo k-fold CV y grilla.

2.1 Evaluacion del modelo, mediante accuracy, con output un vector de niveles

El objetivo de la siguiente celda de código es encontrar los mejores hiperparámetros para el modelo k-NN utilizando el paquete kknn en R. Se realiza una búsqueda exhaustiva de los hiperparámetros óptimos (k, distancia y kernel) utilizando el remuestreo k-fold cross-validation y una grilla de hiperparámetros. Finalmente, se evalúa el modelo utilizando la métrica de precisión (accuracy).

```
# Estrategia
cv <- trainControl(method = "cv", number = 2)

# Parámetros
hyper_grid <- expand.grid(kmax = seq(13, 25, 4), distance = seq(1, 3, 1),
                          kernel = c(
                                     "triangular", "gaussian",
                                     "rank", "optimal"))
```

Configuración de la Estrategia de Remuestreo:

Se define una estrategia de remuestreo utilizando k-fold cross-validation con 2 folds.

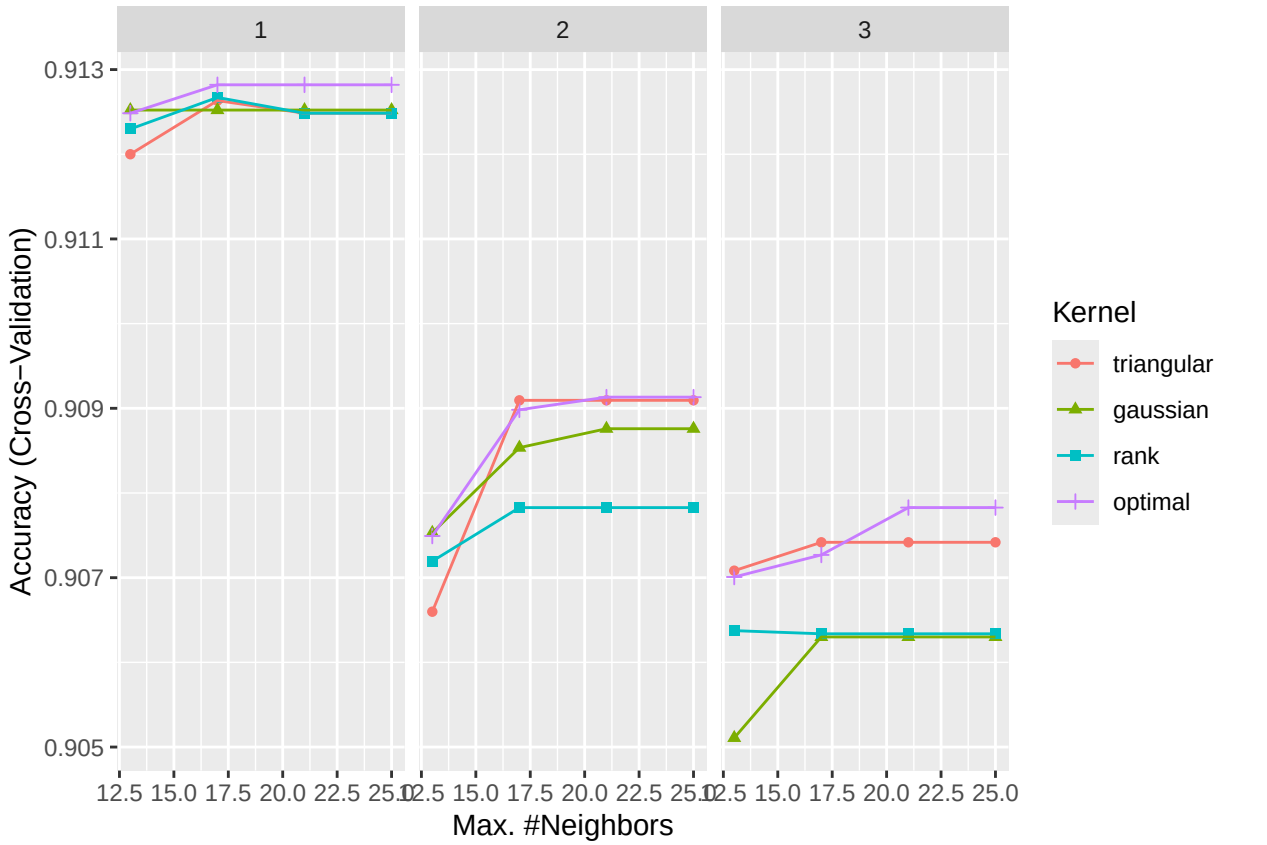
Se crea una grilla de hiperparámetros con diferentes valores para kmax, distance y kernel.

```
# Configuración del procesamiento en paralelo
# Detecta el número de núcleos disponibles
num_cores <- detectCores()
# Crear cluster
# Usar todos menos uno para no sobrecargar el sistema
cl <- makeCluster(num_cores)
# Registrar el cluster
registerDoParallel(cl)

# Entrenamiento
# tomó 39 minutos
set.seed(12345)
kknn_fit <- train(repay_fail ~ ., data = loan_data_knn_train, method = "kknn",
                  metric = "Accuracy", trControl = cv, tuneGrid = hyper_grid)
kknn_fit
```

```
## k-Nearest Neighbors
##
## 26841 samples
##    12 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 13421, 13420
## Resampling results across tuning parameters:
##
##   kmax  distance  kernel      Accuracy   Kappa
##   13    1         triangular  0.9120003  0.6087660
```

```
##   13    1         gaussian    0.9125219  0.6075232
##   13    1         rank        0.9122984  0.6062314
##   13    1         optimal     0.9124847  0.6114094
##   13    2         triangular  0.9065982  0.5865323
##   13    2         gaussian    0.9075296  0.5859040
##   13    2         rank        0.9071942  0.5833756
##   13    2         optimal     0.9074923  0.5880011
##   13    3         triangular  0.9070825  0.5835144
##   13    3         gaussian    0.9051079  0.5749984
##   13    3         rank        0.9063746  0.5756154
##   13    3         optimal     0.9070080  0.5845428
##   17    1         triangular  0.9126337  0.6095689
##   17    1         gaussian    0.9125219  0.6075232
##   17    1         rank        0.9126710  0.6038420
##   17    1         optimal     0.9128200  0.6090507
##   17    2         triangular  0.9090943  0.5900153
##   17    2         gaussian    0.9085355  0.5853722
##   17    2         rank        0.9078276  0.5827376
##   17    2         optimal     0.9089826  0.5898564
##   17    3         triangular  0.9074178  0.5802446
##   17    3         gaussian    0.9063001  0.5729316
##   17    3         rank        0.9063373  0.5746880
##   17    3         optimal     0.9072688  0.5843149
##   21    1         triangular  0.9124847  0.6073318
##   21    1         gaussian    0.9125219  0.6075232
##   21    1         rank        0.9124847  0.6027521
##   21    1         optimal     0.9128200  0.6090507
##   21    2         triangular  0.9090943  0.5900153
##   21    2         gaussian    0.9087590  0.5857962
##   21    2         rank        0.9078276  0.5827376
##   21    2         optimal     0.9091316  0.5886004
##   21    3         triangular  0.9074178  0.5802446
##   21    3         gaussian    0.9063001  0.5729316
##   21    3         rank        0.9063373  0.5746880
##   21    3         optimal     0.9078276  0.5798650
##   25    1         triangular  0.9124847  0.6073318
##   25    1         gaussian    0.9125219  0.6075232
##   25    1         rank        0.9124847  0.6027521
##   25    1         optimal     0.9128200  0.6090507
##   25    2         triangular  0.9090943  0.5900153
##   25    2         gaussian    0.9087590  0.5857962
##   25    2         rank        0.9078276  0.5827376
##   25    2         optimal     0.9091316  0.5886004
##   25    3         triangular  0.9074178  0.5802446
##   25    3         gaussian    0.9063001  0.5729316
##   25    3         rank        0.9063373  0.5746880
##   25    3         optimal     0.9078276  0.5798650
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were kmax = 25, distance = 1 and kernel
##  = optimal.
```

```
ggplot(kknn_fit)
```




```
# Detener el cluster
stopCluster(cl)
registerDoSEQ() # Registramos la secuencia por defecto
```

Interpretación de los Resultados

Gráfico de Resultados:

El gráfico muestra la precisión (accuracy) obtenida para diferentes combinaciones de hiperparámetros (k, distancia y kernel).

Hiperparámetros Óptimos:

Aunque el modelo seleccionó kmax = 25, distancia = 1 y kernel = "optimal", se observó que la precisión fue la misma para kmax = 17 y kmax = 25. Por lo tanto, se seleccionó kmax = 17 para mayor simplicidad y eficiencia computacional.

Relevancia en el Contexto del Algoritmo k-NN

La búsqueda exhaustiva de los hiperparámetros óptimos es crucial para mejorar el rendimiento del modelo k-NN. Utilizar técnicas de remuestreo como k-fold cross-validation asegura que el modelo se evalúe de manera robusta y generalice bien a datos nuevos.

En el proyecto se está utilizando el k-NN para predecir si un cliente fallará en el pago de su préstamo (repay_fail). La optimización de los hiperparámetros (k, distancia y kernel) mediante una búsqueda en grilla y k-fold cross-validation permite mejorar la precisión del modelo, asegurando que las predicciones sean más confiables.

- Eficiencia Computacional: Seleccionar un valor de k más pequeño cuando la precisión es la misma puede reducir el tiempo de cómputo y la complejidad del modelo sin sacrificar precisión.

- Remuestreo: Usar k-fold cross-validation proporciona una evaluación más fiable del rendimiento del modelo al reducir la varianza que podría resultar de una sola división del conjunto de datos.

En la siguiente celda se evalua el rendimiento del modelo k-NN utilizando los mejores hiperparámetros (k = 17, distance = 1, kernel = "optimal") que se identificaron mediante la búsqueda en grilla y k-fold cross-validation. La métrica utilizada para esta evaluación es la precisión (accuracy)

```
# Best Model - Accuracy
set.seed(12345)
kknn_acc <- kknn(repay_fail ~ ., loan_data_knn_train,
                 loan_data_knn_test[, colnames(loan_data_knn_test) !=
                                    "repay_fail"],
                 k = 17, distance = 1, kernel = "optimal")
kknn_acc <- kknn_acc$fitted.values
```

```
## Resultados
results <- confusionMatrix(kknn_acc, repay_fail_test)
results$table
```

```
##            Reference
## Prediction    0    1
##          0 9571  704
##          1  211 1018
```

```
results$overall[names(results$overall) == "Accuracy"]
```

```
##  Accuracy
## 0.9204624
```

Interpretación de los Resultados

Matriz de Confusión:

- True Negatives (TN): 9571 (Clientes predichos correctamente como que no fallarán en el pago del préstamo).
- False Positives (FP): 211 (Clientes predichos incorrectamente como que fallarán en el pago del préstamo, pero en realidad no fallarán).
- False Negatives (FN): 704 (Clientes predichos incorrectamente como que no fallarán en el pago del préstamo, pero en realidad sí fallarán).
- True Positives (TP): 1018 (Clientes predichos correctamente como que fallarán en el pago del préstamo).

Precisión (Accuracy):

La precisión del modelo es aproximadamente 92.05%, lo que indica que el modelo clasifica correctamente el 92.05% de las observaciones en el conjunto de prueba.

- True Negatives (TN) y True Positives (TP) son cruciales para identificar correctamente a los clientes que no representan un riesgo y aquellos que sí lo representan, respectivamente.
- False Positives (FP) y False Negatives (FN) deben minimizarse para evitar la pérdida de buenos clientes y no subestimar el riesgo de clientes problemáticos.

2.2 Evaluacion del modelo mediante ROC, con output un vector de probabilidades

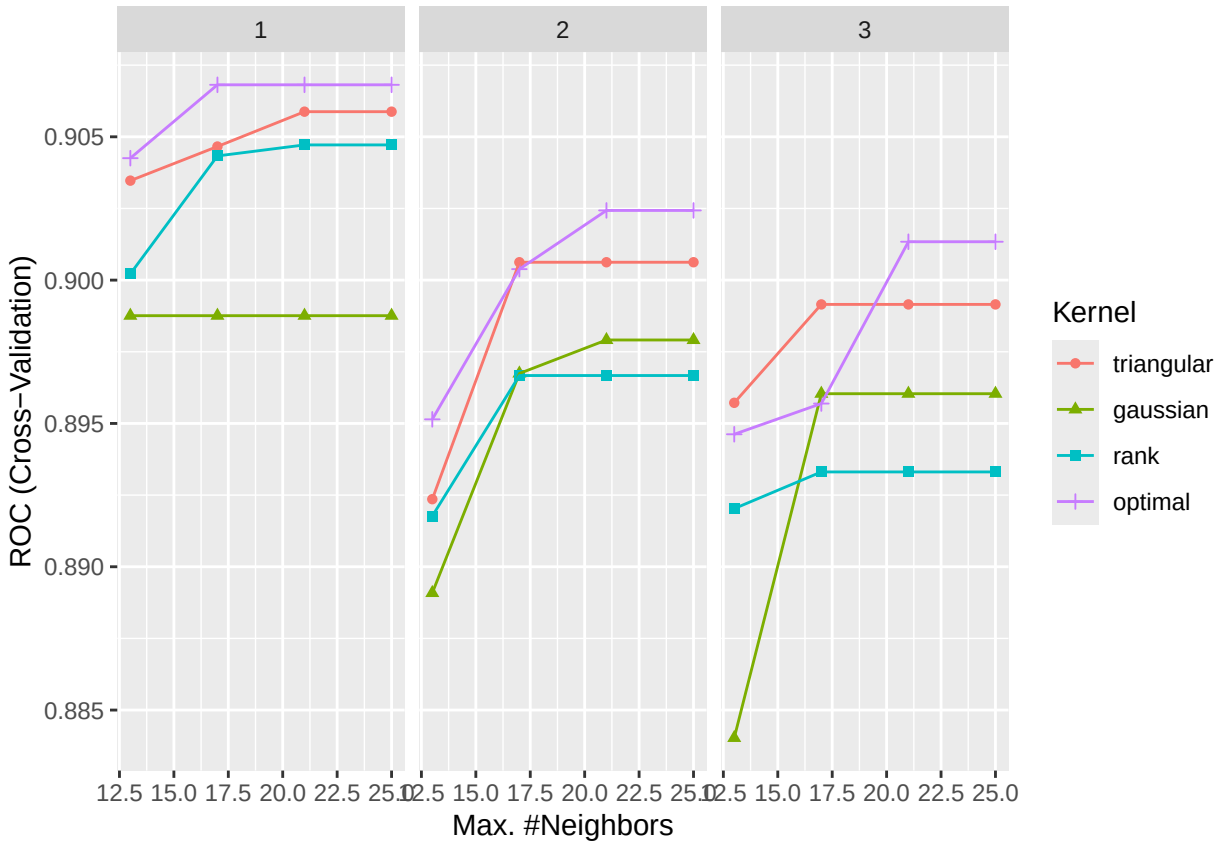
El siguiente código configura una estrategia para evaluar el rendimiento del modelo k-NN utilizando la curva ROC, teniendo como output un vector de probabilidades. Se define una estrategia de remuestreo k-fold cross-validation y se crea una grilla de hiperparámetros para encontrar la mejor combinación de k, distancia y kernel.

```
# Estrategia
cv <- trainControl(method = "cv", number = 2, classProbs = TRUE,
```

```
                    summaryFunction = twoClassSummary)

# Parámetros
hyper_grid <- expand.grid(kmax = seq(13, 25, 4), distance = seq(1, 3, 1),
                          kernel = c("triangular", "gaussian", "rank",
                                     "optimal"))
```

Lo que hace el siguiente código es entrenar y evaluar un modelo k-NN utilizando k-fold cross-validation con una grilla de hiperparámetros para optimizar el ROC. La evaluación se realiza utilizando el área bajo la curva ROC (AUC), que mide la capacidad del modelo para distinguir entre clases. Además, se busca identificar los mejores hiperparámetros (k, distancia y kernel).

```
# Configuración del procesamiento en paralelo
# Detecta el número de núcleos disponibles
num_cores <- detectCores()
# Crear cluster
# Usar todos menos uno para no sobrecargar el sistema
cl <- makeCluster(num_cores)
# Registrar el cluster
registerDoParallel(cl)

# Me aseguro de que los niveles de 'repay_fail' sean nombres válidos
loan_data_knn_train$repay_fail <- factor(loan_data_knn_train$repay_fail,
                                         levels = c("0", "1"),
                                         labels = c("Class_0", "Class_1"))
loan_data_knn_test$repay_fail <- factor(loan_data_knn_test$repay_fail,
                                        levels = c("0", "1"),
                                        labels = c("Class_0", "Class_1"))
# Entrenamiento
#tomó 52 minutos
set.seed(12345)
kknn_fit <- train(repay_fail ~ ., data = loan_data_knn_train, method = "kknn",
                  metric = "ROC", trControl = cv, tuneGrid = hyper_grid)
ggplot(kknn_fit)
```




```
# Detener el cluster
stopCluster(cl)
registerDoSEQ() # Registramos la secuencia por defecto
```

Interpretación de los Resultados Evaluación de Hiperparámetros:

- kmax = 17 y kmax = 25: Los valores de kmax de 17 y 25 muestran un rendimiento similar en términos de AUC, por lo que se seleccionó k = 17 para mayor simplicidad y eficiencia computacional.
- Distance = 1: La distancia óptima identificada es 1.
- Kernel = "optimal": El kernel óptimo es "optimal".

Métricas de Rendimiento:

- ROC: La métrica ROC fue utilizada para seleccionar el modelo óptimo. Un ROC más alto indica una mejor capacidad del modelo para distinguir entre las clases.
- Sensibilidad y Especificidad:
- La sensibilidad (True Positive Rate) es la proporción de verdaderos positivos correctamente identificados por el modelo.
- La especificidad (True Negative Rate) es la proporción de verdaderos negativos correctamente identificados por el modelo.

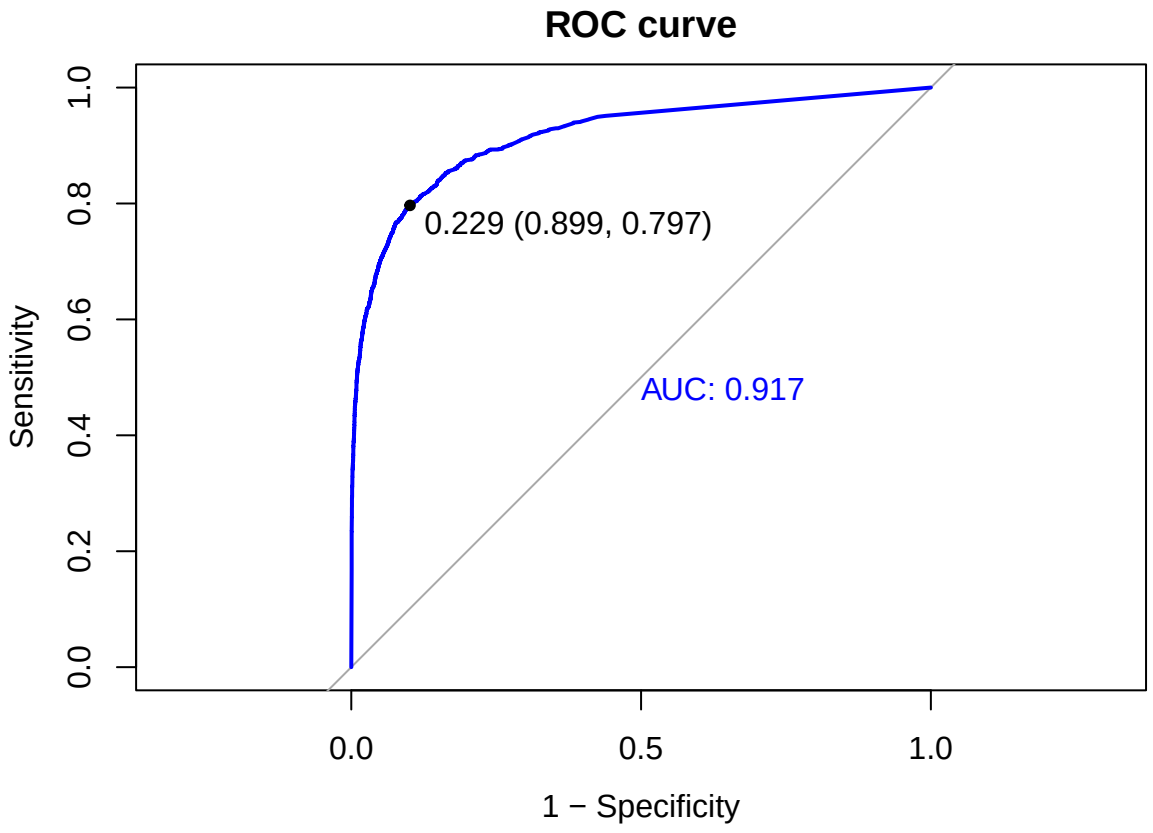
El proposito del siguiente código es evaluar el rendimiento del modelo k-NN utilizando la curva ROC, teniendo como output un vector de probabilidades y determinando el punto de corte óptimo basado en la suma de sensibilidades y especificidades.

```
### Splitting inicial de los datos
library(rsample)
set.seed(123) # Semilla
```

```r
# Muestreo Estratificado
split <- initial_split(loan_data_knn, prop = 0.7, strata = "repay_fail")
loan_data_knn_train <- training(split)
loan_data_knn_test <- testing(split)

# Vectores variable objetivo
repay_fail_train <- loan_data_knn_train$repay_fail
repay_fail_test  <- loan_data_knn_test$repay_fail
```

```r
# Best Model - ROC
set.seed(12345)
kknn_roc <- kknn(repay_fail ~ ., loan_data_knn_train,
                 loan_data_knn_test[, colnames(loan_data_knn_test) !=
                                      "repay_fail"],
                 k = 17, distance = 1, kernel = "optimal")
kknn_roc <- kknn_roc$prob
kknn_roc <- as.numeric(kknn_roc[, colnames(kknn_roc) == "1"])
```

```r
# ROC
kknn_roc_curve <- roc(repay_fail_test, kknn_roc,
                      levels = c("0", "1"), direction = "<")
plot.roc(kknn_roc_curve, main = "ROC curve", col = "blue", lwd = 2,
         legacy.axes = TRUE, print.thres = "best", print.auc = TRUE)
```




```r
# Calculo el punto de corte óptimo basado en la
# suma de sensibilidades y especificidades
aux <- kknn_roc_curve$sensitivities + kknn_roc_curve$specificities
corte <- kknn_roc_curve$thresholds[which(aux == max(aux))]
```

```r
# Convierto las probabilidades a clases binarias usando el punto de corte óptimo
kknn_roc <- ifelse(kknn_roc < corte, 0, 1)
kknn_roc <- factor(kknn_roc, levels = c(0, 1), labels = c("0", "1"))
```

Curva ROC

- La curva ROC muestra la relación entre la tasa de verdaderos positivos (sensibilidad) y la tasa de falsos positivos (1 - especificidad).
- El área bajo la curva (AUC) es 0.917, lo que indica una buena capacidad del modelo para distinguir entre clientes que fallarán y no fallarán en el pago del préstamo.

Interpretación de los Resultados

Curva ROC y AUC:

- Área Bajo la Curva (AUC): Un AUC de 0.917 indica una excelente capacidad del modelo para distinguir entre clientes que fallarán y no fallarán en el pago del préstamo. Un AUC cercano a 1 indica un modelo muy eficaz.
- Curva ROC: La curva ROC proporciona una visión gráfica de la trade-off entre la sensibilidad (tasa de verdaderos positivos) y la especificidad (tasa de verdaderos negativos). Un punto en la curva más cercano a la esquina superior izquierda indica un mejor rendimiento del modelo.
- Umbral de Decisión: El punto en la curva con coordenadas (0.229, 0.899, 0.797) indica el umbral de probabilidad óptimo de 0.229, con una sensibilidad de 0.899 y una especificidad de 0.797.

```r
# Asegurarse de que repay_fail_test sea un factor con los niveles correctos
repay_fail_test <- factor(repay_fail_test, levels = c(0, 1))

# Resultados
results <- confusionMatrix(kknn_roc, repay_fail_test)
results$table
```

```
##           Reference
## Prediction    0    1
##          0 8791  350
##          1  991 1372
```

```r
sensitivity(kknn_roc, repay_fail_test) + specificity(kknn_roc, repay_fail_test)
```

```
## [1] 1.695439
```

Interpretación de los Resultados

Matriz de Confusión:

- True Negatives (TN): 8791 (Clientes predichos correctamente como que no fallarán en el pago del préstamo).
- False Positives (FP): 991 (Clientes predichos incorrectamente como que fallarán en el pago del préstamo, pero en realidad no fallarán).
- False Negatives (FN): 350 (Clientes predichos incorrectamente como que no fallarán en el pago del préstamo, pero en realidad sí fallarán).
- True Positives (TP): 1372 (Clientes predichos correctamente como que fallarán en el pago del préstamo).

Sensibilidad y Especificidad:

- Sensibilidad (True Positive Rate): Es la proporción de verdaderos positivos correctamente identificados por el modelo. En este caso, 1372 clientes que realmente fallaron en pagar el préstamo fueron correctamente identificados.
- Especificidad (True Negative Rate): Es la proporción de verdaderos negativos correctamente identificados por el modelo. En este caso, 8791 clientes que realmente no fallaron en pagar el préstamo fueron

correctamente identificados.

Suma de Sensibilidad y Especificidad:

La suma de sensibilidad y especificidad es 1.6954, lo que refleja un buen equilibrio entre ambos aspectos en el punto de corte óptimo.

Conclusiones y Comparación de Resultados Modelo k-NN

Comparación de Resultados y Conclusiones

1. Precisión (Accuracy):

- Primera Configuración: La precisión del modelo es 90.78%.
- Segunda Configuración: La precisión del modelo es 92.05%.

Conclusión: La optimización de los hiperparámetros mediante remuestreo y grilla resultó en una mejora en la precisión del modelo.

2. Matriz de Confusión:

Primera Configuración:

- True Negatives (TN): 9395
- False Positives (FP): 387
- False Negatives (FN): 674
- True Positives (TP): 1048

Segunda Configuración:

- True Negatives (TN): 9571
- False Positives (FP): 211
- False Negatives (FN): 704
- True Positives (TP): 1018

Conclusión: La segunda configuración reduce significativamente el número de falsos positivos (FP) y ligeramente los verdaderos positivos (TP), lo que mejora la precisión general del modelo.

3. Evaluación mediante ROC:

Primera Configuración:

- AUC: 0.892
- Mejor Umbral de Corte: 0.160
- Suma de Sensibilidad + Especificidad: 1.65250343670887

Segunda Configuración:

- AUC: 0.917
- Umbral de Corte: 0.229
- Suma de Sensibilidad + Especificidad: 1.695439

Conclusión: La segunda configuración también mejora el AUC, lo que indica una mejor capacidad del modelo para distinguir entre clientes que fallarán y no fallarán en el pago de sus préstamos. La mejora en la suma de sensibilidad y especificidad también refleja un mejor balance entre estos dos aspectos críticos.

4. Sensibilidad y Especificidad:

- Primera Configuración: Sensibilidad y especificidad combinadas dan un valor de 1.65250343670887.
- Segunda Configuración: Sensibilidad y especificidad combinadas dan un valor de 1.695439.

Conclusión: La segunda configuración proporciona un equilibrio ligeramente mejor entre la sensibilidad y la especificidad, lo cual es crucial en el contexto del problema para identificar correctamente tanto a los clientes que no fallarán como a los que sí fallarán.

Conclusiones Finales

- La optimización de los hiperparámetros utilizando remuestreo k-fold cross-validation y una grilla de parámetros ha demostrado ser más efectiva que el uso de parámetros subóptimos configurados manualmente. - Los resultados indican mejoras en la precisión (accuracy), el AUC, y la suma de sensibilidad y especificidad. En el contexto del problema de predecir si los clientes fallarán en el pago de

sus préstamos, estas mejoras significan una mayor capacidad del modelo para identificar correctamente tanto a los clientes que representen un riesgo como a aquellos que no.

```
# elimino todas las variables excepto loan_data2
rm(list = setdiff(ls(), "loan_data2"))
```

Analisis Naive Bayes

Transformaciones para el Modelo Naive Bayes

En el preprocesamiento de datos, especialmente cuando se trabaja con modelos de machine learning como naive bayes que no pueden manejar variables categóricas directamente, se convierten estas variables en variables dummy. Este proceso implica crear nuevas columnas binarias para representar las categorías originales de la variable.

En el contexto de Naive Bayes para clasificación, la creación de variables dummy sigue el principio de n - 1 columnas por razones específicas a cómo este algoritmo maneja las variables categóricas.

Razón para n - 1 Columnas Dummy en Naive Bayes

Evitar la Multicolinealidad:

Aunque Naive Bayes no se ve afectado por la multicolinealidad de la misma manera que otros modelos lineales (como la regresión lineal o logística), crear n columnas para n categorías introduce redundancia innecesaria. Esto puede llevar a cálculos innecesarios y complicaciones en la interpretación.

Simplicidad y Eficiencia Computacional:

Naive Bayes asume que las características son independientes unas de otras (suposición de independencia condicional). Para mantener este supuesto de manera eficiente, se crean n - 1 columnas dummy. Esto simplifica los cálculos de probabilidad y evita el sobreajuste, ya que la redundancia puede hacer que el modelo se ajuste demasiado a los datos de entrenamiento.

Categoría de Referencia:

Al crear n - 1 dummies, una categoría se toma como referencia implícita. Esta referencia no necesita ser representada explícitamente porque su información está contenida en las otras variables dummy. En términos de Naive Bayes, esto significa que la probabilidad de la categoría de referencia se puede calcular implícitamente cuando todas las otras dummies son 0.

```
#tratamiento de variable categórica term
table(loan_data2$term) # 2 levels (1 dummy)
```

```
##
## 36 months 60 months
##     28478      9867
```

```
loan_data2$term_60_months <- ifelse(loan_data2$term == "60 months", 1, 0)
loan_data2$term <- ifelse(loan_data2$term == "36 months", 1, 0)
```

En este caso:

term_60_months = 1 si el término es “60 months”, 0 en caso contrario. term = 1 si el término es “36 months”, 0 en caso contrario. Aquí, una de las categorías (term para “36 months”) actúa como la categoría de referencia implícita.

Para Naive Bayes:

Probabilidades Condicionales:

Naive Bayes calculará las probabilidades condicionales de cada categoría de la variable objetivo dado cada variable dummy. Por ejemplo, calculará P(repay_fail , term_60_months = 1) y P(repay_fail , term = 1).

Independencia Condicional:

Asumiendo independencia condicional, Naive Bayes combinará estas probabilidades para predecir la categoría de la variable objetivo. La eliminación de una dummy asegura que no haya redundancia en estas combinaciones.

Cálculo de Probabilidades:

La categoría de referencia ("36 months" en este caso) no necesita una columna dummy explícita. Su probabilidad se deduce cuando term_60_months = 0. Naive Bayes utiliza esta información para calcular P(repay_fail , term = 1) implícitamente.

Para las otras columnas categoricas se aplicará lo mismo

```
# last_pymnt_d
table(loan_data2$last_pymnt_d) # 3 levels (2 dummies)
```

```
##
## 2007-2010 2011-2013 2014-2016
##      3222     22539     12584
```

```
loan_data2$last_pymnt_d_2011_2013 <- ifelse(loan_data2$last_pymnt_d ==
                                              "2011-2013", 1, 0)
loan_data2$last_pymnt_d_2014_2016 <- ifelse(loan_data2$last_pymnt_d ==
                                              "2014-2016", 1, 0)
loan_data2$last_pymnt_d <- ifelse(loan_data2$last_pymnt_d ==
                                    "2007-2010", 1, 0)
```

```
# last_credit_pull_d
table(loan_data2$last_credit_pull_d) # 3 levels (2 dummies)
```

```
##
## 2007-2010 2011-2013 2014-2016
##      1431     11041     25873
```

```
loan_data2$last_credit_pull_d_2011_2013 <- ifelse(loan_data2$last_credit_pull_d ==
                                                    "2011-2013", 1, 0)
loan_data2$last_credit_pull_d_2014_2016 <- ifelse(loan_data2$last_credit_pull_d ==
                                                    "2014-2016", 1, 0)
loan_data2$last_credit_pull_d <- ifelse(loan_data2$last_credit_pull_d ==
                                          "2007-2010", 1, 0)
```

Tratamiento variables numericas (Binning)

En el contexto de Naive Bayes para clasificación, las variables numéricas se convierten en categóricas mediante un proceso llamado binning. Este proceso divide el rango de valores de una variable numérica en varios intervalos discretos, transformándolos en variables categóricas que el algoritmo puede manejar eficientemente.

```
#Variable numerica int_rate

# int_rate
#Min. : 5.42
#1st Qu.: 9.62
#Median :11.99
#Mean :12.15
#3rd Qu.:14.72
#Max. :24.11

loan_data2$int_rate_low <- ifelse(loan_data2$int_rate < 9.62, 1, 0)
loan_data2$int_rate_mid_low <- ifelse(loan_data2$int_rate >= 9.62 &
                                        loan_data2$int_rate < 11.99, 1, 0)
loan_data2$int_rate_mid_high <- ifelse(loan_data2$int_rate >= 11.99 &
                                         loan_data2$int_rate < 14.72, 1, 0)
loan_data2$int_rate_high <- ifelse(loan_data2$int_rate >= 14.72, 1, 0)
```

Transformación de Variables Numéricas en Categóricas:

El algoritmo Naive Bayes requiere variables 0 y 1 para funcionar correctamente. El binning convierte una variable numérica (int_rate en este caso) en varias variables binarias categóricas.

Simplificación y Interpretación

Dividir una variable numérica en intervalos discretos puede simplificar el modelo y mejorar la interpretación, ya que los intervalos pueden representar categorías significativas (por ejemplo, tasas de interés bajas, medias y altas).

Detalles del Binning

Definición de Intervalos

Los intervalos se definen basándose en los percentiles (cuartiles) de la distribución de la variable `int_rate`:

- `int_rate_low`: Tasa de interés menor a 9.62 (primer cuartil).
- `int_rate_mid_low`: Tasa de interés entre 9.62 y 11.99 (primer cuartil a la mediana).
- `int_rate_mid_high`: Tasa de interés entre 11.99 y 14.72 (mediana al tercer cuartil).
- `int_rate_high`: Tasa de interés mayor o igual a 14.72 (tercer cuartil).

Creación de Variables Binarias:

Cada intervalo se convierte en una variable binaria:

- `int_rate_low`: Toma el valor 1 si int_rate es menor a 9.62, y 0 en caso contrario.
- `int_rate_mid_low`: Toma el valor 1 si int_rate está entre 9.62 y 11.99, y 0 en caso contrario.
- `int_rate_mid_high`: Toma el valor 1 si int_rate está entre 11.99 y 14.72, y 0 en caso contrario.
- `int_rate_high`: Toma el valor 1 si int_rate es mayor o igual a 14.72, y 0 en caso contrario.

Impacto en el Proceso de Naive Bayes

Adaptación del Algoritmo:

Al convertir int_rate en variables categóricas, Naive Bayes puede manejar estas características de manera eficiente, calculando las probabilidades condicionales de cada categoría respecto a la variable objetivo repay_fail.

Mantenimiento del Supuesto de Independencia:

El supuesto de independencia condicional en Naive Bayes se mantiene más fácilmente con variables binarias, simplificando el cálculo de las probabilidades conjuntas.

Para las siguientes variables numericas se realizó el mismo proceso, pero con sus respectivos cuartiles.

```
# variable numerica annual_inc

#Min.   :   1896
#1st Qu.:  40000
#Median :  58880
#Mean   :  69058
#3rd Qu.:  82100
#Max.   :6000000

loan_data2$annual_inc_high <- ifelse(loan_data2$annual_inc > 82100, 1, 0)
loan_data2$annual_inc_low <- ifelse(loan_data2$annual_inc < 40000, 1, 0)
loan_data2$annual_inc_mid_low <- ifelse(loan_data2$annual_inc >= 40000 &
                                        loan_data2$annual_inc < 58880, 1, 0)
loan_data2$annual_inc_mid_high <- ifelse(loan_data2$annual_inc >= 58880 &
                                         loan_data2$annual_inc <= 82100, 1, 0)
```

Para la variable numérica “annual_inc”, considerando los valores, hice los binning de la siguiente manera:

- Menor que el primer cuartil (40,000)

- Entre el primer cuartil y la mediana (40,000 - 58,880)
- Entre la mediana y el tercer cuartil (58,880 - 82,100)
- Mayor que el tercer cuartil (82,100)

```
# variable numerica dti

#Min.   : 0.00
#1st Qu.: 8.21
#Median :13.49
#Mean   :13.38
#3rd Qu.:18.69
#Max.   :29.99

loan_data2$dti_high <- ifelse(loan_data2$dti > 18.69, 1, 0)
loan_data2$dti_low <- ifelse(loan_data2$dti < 8.21, 1, 0)
loan_data2$dti_mid_low <- ifelse(loan_data2$dti >= 8.21 &
                                 loan_data2$dti < 13.49, 1, 0)
loan_data2$dti_mid_high <- ifelse(loan_data2$dti >= 13.49 &
                                  loan_data2$dti <= 18.69, 1, 0)
```

Para la variable numérica “dti”, considerando los valores, hice los binning de la siguiente manera:

- Menor que el primer cuartil (8.21)
- Entre el primer cuartil y la mediana (8.21 - 13.49)
- Entre la mediana y el tercer cuartil (13.49 - 18.69)
- Mayor que el tercer cuartil (18.69)

```
# variable numerica total_pymnt

#Min.   :   35.71
#1st Qu.: 5487.53
#Median : 9708.77
#Mean   :12012.95
#3rd Qu.:16427.20
#Max.   :58563.68

loan_data2$total_pymnt_high <- ifelse(loan_data2$total_pymnt > 16427.20, 1, 0)
loan_data2$total_pymnt_low <- ifelse(loan_data2$total_pymnt < 5487.53, 1, 0)
loan_data2$total_pymnt_mid_low <- ifelse(loan_data2$total_pymnt >= 5487.53 &
                                         loan_data2$total_pymnt < 9708.77, 1, 0)
loan_data2$total_pymnt_mid_high <- ifelse(loan_data2$total_pymnt >= 9708.77 &
                                          loan_data2$total_pymnt
                                          <= 16427.20, 1, 0)
```

Para la variable numérica “total_pymnt”, considerando los valores, hice los binning de la siguiente manera:

- Menor que el primer cuartil (5487.53)
- Entre el primer cuartil y la mediana (5487.53 - 9708.77)
- Entre la mediana y el tercer cuartil (9708.77 - 16427.20)
- Mayor que el tercer cuartil (16427.20)

```
# variable numerica total_pymnt_inv

#Min.   :    0
#1st Qu.: 4851
#Median : 8981
#Mean   :11307
```

```
#3rd Qu.:15517
#Max.   :58564

loan_data2$total_pymnt_inv_high <- ifelse(loan_data2$total_pymnt_inv >
                                          15517, 1, 0)
loan_data2$total_pymnt_inv_low <- ifelse(loan_data2$total_pymnt_inv < 4851, 1, 0)
loan_data2$total_pymnt_inv_mid_low <- ifelse(loan_data2$total_pymnt_inv >= 4851 &
                                             loan_data2$total_pymnt_inv <
                                               8981, 1, 0)
loan_data2$total_pymnt_inv_mid_high <- ifelse(loan_data2$total_pymnt_inv >= 8981 &
                                              loan_data2$total_pymnt_inv <=
                                                15517, 1, 0)
```

Para la variable numérica "total_pymnt_inv", considerando los valores, hice binning de la siguiente manera:

- Menor que el primer cuartil (4851)
- Entre el primer cuartil y la mediana (4851 - 8981)
- Entre la mediana y el tercer cuartil (8981 - 15517)
- Mayor que el tercer cuartil (15517)

```
# variable numerica total_rec_prncp

#Min.   :    0
#1st Qu.: 4485
#Median : 8000
#Mean   : 9673
#3rd Qu.:13450
#Max.   :35000

loan_data2$total_rec_prncp_high <- ifelse(loan_data2$total_rec_prncp >
                                          13450, 1, 0)
loan_data2$total_rec_prncp_low <- ifelse(loan_data2$total_rec_prncp < 4485, 1, 0)
loan_data2$total_rec_prncp_mid_low <- ifelse(loan_data2$total_rec_prncp >= 4485 &
                                             loan_data2$total_rec_prncp <
                                               8000, 1, 0)
loan_data2$total_rec_prncp_mid_high <- ifelse(loan_data2$total_rec_prncp >= 8000 &
                                              loan_data2$total_rec_prncp <=
                                                13450, 1, 0)
```

Para la variable numérica "total_rec_prncp", considerando los valores, hice binning de la siguiente manera:

- Menor que el primer cuartil (4485)
- Entre el primer cuartil y la mediana (4485 - 8000)
- Entre la mediana y el tercer cuartil (8000 - 13450)
- Mayor que el tercer cuartil (13450)

```
# variable numerica last_pymnt_amnt

#Min.   :    0.0
#1st Qu.:  213.8
#Median :  528.5
#Mean   : 2622.5
#3rd Qu.: 3184.3
#Max.   :36115.2

loan_data2$last_pymnt_amnt_high <- ifelse(loan_data2$last_pymnt_amnt >
```

```
                                              3184.3, 1, 0)
loan_data2$last_pymnt_amnt_low <- ifelse(loan_data2$last_pymnt_amnt < 213.8, 1, 0)
loan_data2$last_pymnt_amnt_mid_low <- ifelse(loan_data2$last_pymnt_amnt >=
                                               213.8 &
                                               loan_data2$last_pymnt_amnt <
                                                 528.5, 1, 0)
loan_data2$last_pymnt_amnt_mid_high <- ifelse(loan_data2$last_pymnt_amnt >=
                                                528.5 &
                                                loan_data2$last_pymnt_amnt <=
                                                  3184.3, 1, 0)
```

Para la variable numérica "last_pymnt_amnt", considerando los valores, hice binning de la siguiente manera:

- Menor que el primer cuartil (213.8)
- Entre el primer cuartil y la mediana (213.8 - 528.5)
- Entre la mediana y el tercer cuartil (528.5 - 3184.3)
- Mayor que el tercer cuartil (3184.3)

La siguiente celda elimina las columnas que ya fueron procesadas, de acuerdo a si son numericas y categoricas, eliminando su columna original no transformada.

```
loan_data2 <- loan_data2 %>%
  dplyr::select(-term, -last_pymnt_d, -last_credit_pull_d,
                -int_rate, -annual_inc, -dti, -total_pymnt,
                -total_pymnt_inv, -total_rec_prncp, -last_pymnt_amnt)
dim(loan_data2)
```

```
## [1] 38345    34
```

```
#nombre de las columnas de loan_data2
colnames(loan_data2)
```

```
##  [1] "repay_fail"                   "term_60_months"
##  [3] "last_pymnt_d_2011_2013"       "last_pymnt_d_2014_2016"
##  [5] "last_credit_pull_d_2011_2013" "last_credit_pull_d_2014_2016"
##  [7] "int_rate_low"                 "int_rate_mid_low"
##  [9] "int_rate_mid_high"            "int_rate_high"
## [11] "annual_inc_high"              "annual_inc_low"
## [13] "annual_inc_mid_low"           "annual_inc_mid_high"
## [15] "dti_high"                     "dti_low"
## [17] "dti_mid_low"                  "dti_mid_high"
## [19] "total_pymnt_high"             "total_pymnt_low"
## [21] "total_pymnt_mid_low"          "total_pymnt_mid_high"
## [23] "total_pymnt_inv_high"         "total_pymnt_inv_low"
## [25] "total_pymnt_inv_mid_low"      "total_pymnt_inv_mid_high"
## [27] "total_rec_prncp_high"         "total_rec_prncp_low"
## [29] "total_rec_prncp_mid_low"      "total_rec_prncp_mid_high"
## [31] "last_pymnt_amnt_high"         "last_pymnt_amnt_low"
## [33] "last_pymnt_amnt_mid_low"      "last_pymnt_amnt_mid_high"
```

La siguiente celda asegura que la variable objetivo repay_fail esté en el formato adecuado para ser utilizada en el modelo Naive Bayes, que requiere que la variable de clase sea categórica (en este caso, un factor en R).

```
# repay_fail lo dejo como formato factor
table(loan_data2$repay_fail)
```

```
##
##     0     1
```

```
## 32606  5739
```

```
loan_data2$repay_fail <- as.factor(loan_data2$repay_fail)
prop.table(table(loan_data2$repay_fail))
```

```
##
##         0         1
## 0.8503325 0.1496675
```

En la siguiente celda se realiza la división del conjunto de datos en conjuntos de entrenamiento y prueba, asegurando que la distribución de la variable objetivo repay_fail se mantenga consistente en ambos conjuntos mediante estratificación. Se utiliza una semilla aleatoria para garantizar la reproducibilidad de los resultados.

```
### Splitting initial de los datos en conjunto de entrenamiento y prueba
set.seed(123) # Semilla
split <- initial_split(loan_data2, prop = 0.7, strata = "repay_fail")
loan_data_train <- training(split)
loan_data_test <- testing(split)
```

Por qué del 70%-30%

Suficiente Cantidad de Datos para Entrenamiento:

Usar el 70% de los datos para el entrenamiento asegura que el modelo tenga acceso a una cantidad suficiente de datos para aprender las características y patrones del conjunto de datos. Esto es esencial para construir un modelo robusto y reducir el riesgo de subajuste.

Evaluación Representativa:

Al reservar el 30% de los datos para el conjunto de prueba, se asegura una evaluación representativa del rendimiento del modelo. Este conjunto debe ser lo suficientemente grande para proporcionar una evaluación precisa y confiable del modelo en datos no vistos.

Prácticas Comunes:

La proporción de 70%-30% es una práctica común en machine learning, proporcionando un buen balance entre tener suficientes datos para el entrenamiento y una cantidad adecuada para la evaluación del modelo.

Subajuste y Sobreajuste

Subajuste

Definición: Ocurre cuando un modelo es demasiado simple para capturar la estructura subyacente de los datos. Un modelo con subajuste tendrá un rendimiento pobre tanto en el conjunto de entrenamiento como en el de prueba.

Prevención: Utilizar una proporción adecuada de datos de entrenamiento ayuda a mitigar el subajuste. Con el 70% de los datos para el entrenamiento, el modelo tiene suficiente información para aprender patrones complejos sin ser demasiado simple.

Sobreajuste

Definición: Ocurre cuando un modelo es demasiado complejo y se ajusta demasiado a los datos de entrenamiento, capturando también el ruido y las peculiaridades específicas de estos datos. Un modelo con sobreajuste tendrá un rendimiento excelente en el conjunto de entrenamiento pero pobre en el conjunto de prueba.

Prevención: Reservar el 30% de los datos para el conjunto de prueba permite evaluar cómo generaliza el modelo a datos nuevos. Si el modelo tiene un rendimiento significativamente peor en el conjunto de prueba comparado con el conjunto de entrenamiento, esto indica un posible sobreajuste.

Estratificación por repay_fail

Motivación: La estratificación por la variable objetivo repay_fail asegura que la distribución de las clases (yes y no) sea representativa en ambos conjuntos (entrenamiento y prueba). Esto es crucial para evitar sesgos y asegurar que el modelo sea evaluado de manera justa en una muestra que refleja la población original.

La proporción 70%-30% para la división del conjunto de datos se elige para equilibrar adecuadamente la cantidad de datos disponibles para el entrenamiento y la evaluación, ayudando a prevenir tanto el subajuste como el sobreajuste. La estratificación por repay_fail asegura que ambos conjuntos tengan distribuciones representativas, mejorando la robustez y la precisión de la evaluación del modelo.

El siguiente código verifica que la proporción de la variable objetivo repay_fail en los conjuntos de entrenamiento y prueba sea consistente, y luego genera vectores para la variable objetivo en ambos conjuntos.

```
# Compruebo proporcion de "repay_fail"
round(prop.table(table(loan_data_train$repay_fail)), 2)
```

```
##
##    0    1
## 0.85 0.15
```

```
round(prop.table(table(loan_data_test$repay_fail)), 2)
```

```
##
##    0    1
## 0.85 0.15
```

```
# genero vectores de la variable objetivo
repay_fail_train <- loan_data_train$repay_fail
repay_fail_test <- loan_data_test$repay_fail
```

Verificación de la Proporción de repay_fail:

El uso de round(prop.table(table(...)), 2) permite comprobar que la distribución de la variable repay_fail es similar en los conjuntos de entrenamiento y prueba. Esto es importante para asegurar que ambos conjuntos sean representativos de la población original.

Los resultados 0.85 y 0.15 para ambas clases (0 y 1) indican que el 85% de las observaciones no tienen fallos de repago (0) y el 15% sí tienen fallos de repago (1), tanto en el conjunto de entrenamiento como en el de prueba.

1era forma: Evaluación del Modelo Naive Bayes sin busqueda de Parámetros

1era forma: Paquete e1071, metodo naiveBayes que no permite configuracion de parametros (excepto sólo laplace) y colocando hiperparametro suboptimo de forma manual, sin saber hiperparametro y parametros optimos que se obtendrian mediante remuestreo y grilla.

1.1 Evaluacion del modelo, metrica de accuracy, con output un vector de niveles

```
set.seed(12345)
nb_level <- naiveBayes(repay_fail ~ ., data = loan_data_train, laplace = 0)
nb_level <- predict(nb_level,
                    loan_data_test[colnames(loan_data_test) != "repay_fail"],
                    type = "class")
```

```
# Resultados
results <- confusionMatrix(nb_level, repay_fail_test)
results$table
```

```
##            Reference
## Prediction    0    1
##          0 7536  410
##          1 2246 1312
```

```
results$overall[names(results$overall) == "Accuracy"]
```

```
##  Accuracy
## 0.7691238
```

Entrenamiento del Modelo Naive Bayes:

naiveBayes(repay_fail ~ ., data = loan_data_train, laplace = 0) entrena un modelo Naive Bayes utilizando el conjunto de datos de entrenamiento. El parámetro laplace = 0 indica que no se está aplicando suavizado de Laplace.

Predicción:

predict(nb_level, loan_data_test[colnames(loan_data_test) != repay_fail], type = "class") genera predicciones para el conjunto de prueba. El tipo de predicción class indica que se deben devolver las clases predichas.

Evaluación del Modelo:

confusionMatrix(nb_level, repay_fail_test) calcula la matriz de confusión y varias métricas de evaluación, incluyendo la precisión (accuracy). results$overall[names(results$overall) == "Accuracy"] extrae el valor de precisión del resultado de la matriz de confusión.

Interpretación de los Resultados

Matriz de Confusión:

La matriz de confusión muestra la cantidad de verdaderos negativos, falsos negativos, verdaderos positivos y falsos positivos:

- 7536 verdaderos negativos (TN)
- 410 falsos negativos (FN)
- 2246 falsos positivos (FP)
- 1312 verdaderos positivos (TP)

Precisión (Accuracy):

La precisión es una métrica que mide la proporción de predicciones correctas sobre el total de predicciones realizadas. La precisión de 0.7691 indica que el modelo Naive Bayes predijo correctamente aproximadamente

el 77% de las veces en el conjunto de prueba. Sin embargo, esta evaluación inicial usa un valor subóptimo para el parámetro de Laplace y no ha optimizado otros hiperparámetros posibles. Para mejorar el rendimiento del modelo, será útil realizar una búsqueda de hiperparámetros mediante técnicas de remuestreo y grillas, lo que se abordará en pasos posteriores.

Definiciones en el Contexto de Predicción del No Pago de Préstamos

En el contexto de intentar predecir el no pago de préstamos por parte de los clientes, las definiciones de verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN) son las siguientes:

1. **Verdadero Positivo (TP)**:
   - Un verdadero positivo ocurre cuando el modelo predice correctamente que un cliente no pagará su préstamo.
   - **Ejemplo**: El modelo predice que un cliente no pagará su préstamo, y efectivamente el cliente no lo paga.
2. **Verdadero Negativo (TN)**:
   - Un verdadero negativo ocurre cuando el modelo predice correctamente que un cliente pagará su préstamo.
   - **Ejemplo**: El modelo predice que un cliente pagará su préstamo, y efectivamente el cliente lo paga.
3. **Falso Positivo (FP)**:
   - Un falso positivo ocurre cuando el modelo predice incorrectamente que un cliente no pagará su préstamo, pero en realidad el cliente sí lo paga.
   - **Ejemplo**: El modelo predice que un cliente no pagará su préstamo, pero el cliente efectivamente lo paga.
4. **Falso Negativo (FN)**:
   - Un falso negativo ocurre cuando el modelo predice incorrectamente que un cliente pagará su préstamo, pero en realidad el cliente no lo paga.
   - **Ejemplo**: El modelo predice que un cliente pagará su préstamo, pero el cliente efectivamente no lo paga.

Importancia de Estas Definiciones en el Contexto del Problema

- **Verdaderos Positivos (TP)**: Identificar correctamente a los clientes que no pagarán sus préstamos es crucial para la gestión de riesgos en las instituciones financieras. Permite tomar medidas preventivas y reducir el impacto de los préstamos incobrables.
- **Verdaderos Negativos (TN)**: Reconocer correctamente a los clientes que pagarán sus préstamos asegura que se concedan préstamos a clientes de bajo riesgo, optimizando el rendimiento de la cartera de préstamos.
- **Falsos Positivos (FP)**: Predicciones incorrectas de no pago pueden llevar a la denegación de préstamos a clientes que son de bajo riesgo, lo cual puede resultar en la pérdida de oportunidades de negocio y afectar negativamente la satisfacción del cliente.
- **Falsos Negativos (FN)**: Predicciones incorrectas de pago pueden resultar en la aprobación de préstamos a clientes que son de alto riesgo, aumentando la probabilidad de incurrir en pérdidas por préstamos incobrables.

Comprender y minimizar los falsos negativos y falsos positivos es esencial para mejorar la precisión del modelo y la toma de decisiones en la concesión de préstamos, contribuyendo a una gestión de riesgos más efectiva y a la optimización de los recursos financieros.

1.2 Evaluacion de modelo, mediante ROC, con output un vector de probabilidades

```
set.seed(12345)
nb_prob <- naiveBayes(repay_fail ~ ., data = loan_data_train, laplace = 0)
nb_prob <- predict(nb_prob,
```

```
                  loan_data_test[colnames(loan_data_test) != "repay_fail"],
                  type = "raw")
nb_prob <- as.numeric(nb_prob[, colnames(nb_prob) == "1"])
```

```
# Umbral nivel/probabilidad
level_prob <- data.frame(nb_level, nb_prob)
level_prob <- min(level_prob[level_prob$nb_level == "1", "nb_prob"])
level_prob
```

```
## [1] 0.5035332
```

Explicación del Propósito Entrenamiento del Modelo Naive Bayes:

naiveBayes(repay_fail ~ ., data = loan_data_train, laplace = 0) entrena un modelo Naive Bayes utilizando el conjunto de datos de entrenamiento. El parámetro laplace = 0 indica que no se está aplicando suavizado de Laplace.

Predicción:

predict(nb_prob, loan_data_test[colnames(loan_data_test) != "repay_fail"], type = "raw") genera predicciones para el conjunto de prueba en forma de probabilidades. type = "raw" indica que se deben devolver las probabilidades de cada clase. nb_prob <- as.numeric(nb_prob[, colnames(nb_prob) == "1"]) extrae las probabilidades asociadas con la clase 1.

Cálculo del Umbral de Probabilidad:

level_prob <- data.frame(nb_level, nb_prob) crea un data frame con las predicciones de clase y sus probabilidades correspondientes. level_prob <- min(level_prob[level_prob$nb_level == "1", "nb_prob"]) encuentra el umbral mínimo de probabilidad para el cual se predice la clase 1.

Interpretación del Resultado

Este umbral indica que cualquier instancia con una probabilidad predicha mayor o igual a 0.503533225275366 será clasificada como clase 1 (repay_fail = 1). Este valor es crucial para trazar la curva ROC y evaluar el rendimiento del modelo en términos de sensibilidad y especificidad.

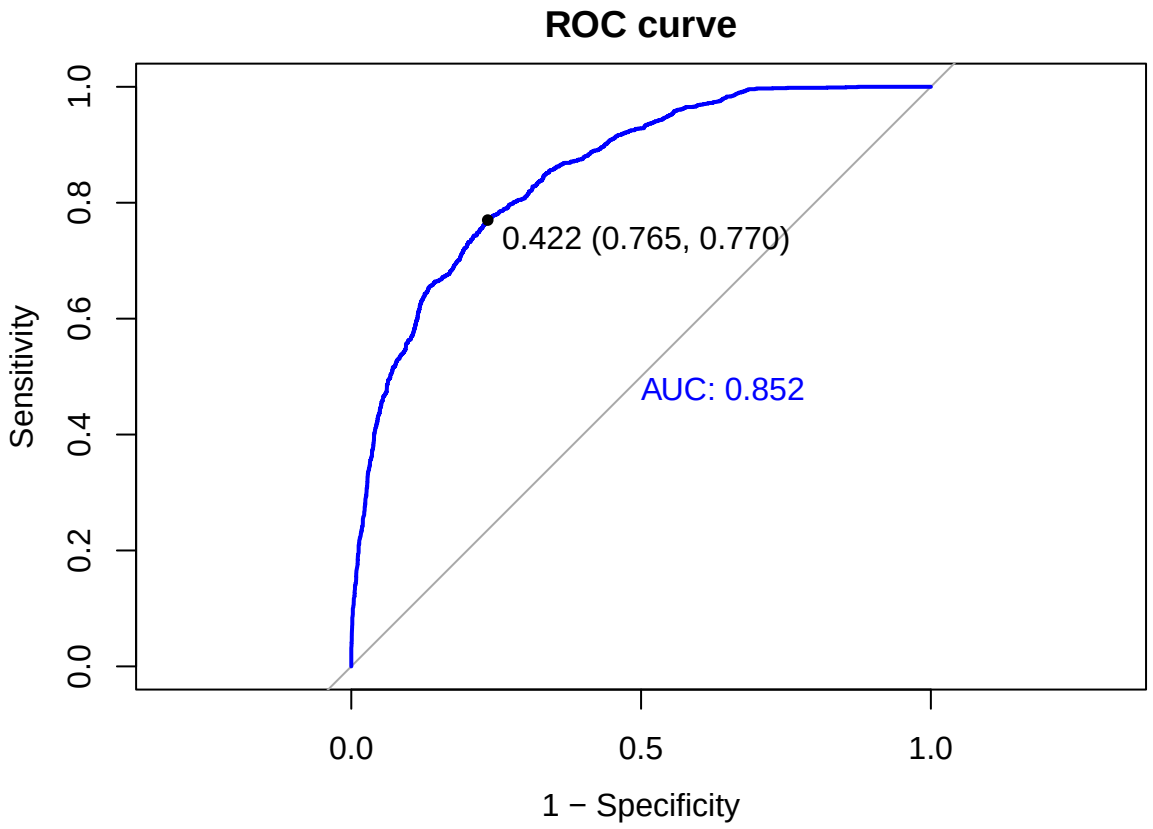
Este valor indica que cualquier cliente con una probabilidad de no pago mayor o igual a 0.5035 será clasificado como de alto riesgo de no pago. Este umbral es crucial para determinar la clasificación final de los clientes y ayuda a tomar decisiones informadas sobre la aprobación de préstamos.

El umbral de probabilidad calculado proporciona un punto de corte para clasificar las instancias en la clase 1. Utilizar este umbral permite evaluar el modelo Naive Bayes mediante la curva ROC, proporcionando una visión más detallada del rendimiento del modelo en comparación con solo utilizar la precisión.

Creación de la Curva ROC

El siguiente código crea la curva ROC y encuentra el mejor umbral de corte para ajustar las probabilidades.

```
# Creo el ROC y la curva ROC
nb_roc <- roc(repay_fail_test, nb_prob, levels = c("0", "1"), direction = "<")
plot.roc(nb_roc, main = "ROC curve", col = "blue", lwd = 2, legacy.axes = TRUE,
         print.thres = "best", print.auc = TRUE)
```




```r
# Encuentro el mejor umbral de corte
aux <- nb_roc$sensitivities + nb_roc$specificities
corte <- nb_roc$thresholds[which(aux == max(aux))]

# Ajusto las probabilidades al umbral de corte
nb_prob <- ifelse(nb_prob < corte, 0, 1)
nb_prob <- factor(nb_prob, levels = c(0, 1), labels = c("0", "1"))
```

El umbral de corte encontrado es 0.422.

Interpretación del Resultado

Curva ROC: La curva ROC muestra la relación entre la sensibilidad (True Positive Rate) y 1 - especificidad (False Positive Rate) para diferentes umbrales de probabilidad.

AUC (Área Bajo la Curva): Un AUC de 0.852 indica un buen rendimiento del modelo, ya que un valor cercano a 1 representa un modelo con alta capacidad de discriminación.

Mejor Umbral de Corte: El mejor umbral de corte (0.422) se utiliza para ajustar las probabilidades, clasificando cualquier instancia con una probabilidad mayor o igual a este valor como clase 1.

El uso de la curva ROC y el ajuste del umbral de corte proporciona una evaluación más detallada del rendimiento del modelo Naive Bayes. La AUC de 0.852 y el umbral de corte identificado aseguran que el modelo tenga un buen balance entre sensibilidad y especificidad.

Explicación de la Imagen

- **Curva ROC**: La curva ROC muestra cómo varía la tasa de verdaderos positivos (sensibilidad) contra la tasa de falsos positivos (1 - especificidad) para diferentes umbrales.
- **AUC**: El área bajo la curva (AUC) es 0.852, lo que indica que el modelo tiene un buen rendimiento de clasificación.

- **Mejor Umbral de Corte**: El mejor umbral de corte es 0.422, lo que proporciona un buen equilibrio entre sensibilidad y especificidad.

Este análisis confirma que el modelo Naive Bayes, evaluado mediante la curva ROC, tiene un buen desempeño, con un AUC de 0.852. Ajustar las probabilidades utilizando el mejor umbral de corte mejora la precisión del modelo.

En el contexto de predecir el no pago de préstamos:

- **AUC**: Indica la capacidad general del modelo para discriminar entre clientes de alto y bajo riesgo.
- **Umbral de Corte**: Define la probabilidad a partir de la cual un cliente es clasificado como de alto riesgo.
- **Sensibilidad**: Mide la capacidad del modelo para identificar correctamente a los clientes de alto riesgo.
- **Especificidad**: Mide la capacidad del modelo para identificar correctamente a los clientes de bajo riesgo.
- **Suma de Sensibilidad y Especificidad**: Proporciona una visión combinada del rendimiento del modelo, indicando un buen equilibrio entre las tasas de verdaderos positivos y verdaderos negativos.

El siguiente código muestra cómo evaluar los resultados del modelo Naive Bayes ajustado con el mejor umbral de corte utilizando la matriz de confusión, y calcula la suma de la sensibilidad y la especificidad.

```
# Resultados
results <- confusionMatrix(nb_prob, repay_fail_test)
results$table
```

```
##            Reference
## Prediction    0    1
##          0 7479  396
##          1 2303 1326
```

```
sensitivity(nb_prob, repay_fail_test) + specificity(nb_prob, repay_fail_test)
```

```
## [1] 1.534602
```

Interpretación de los Resultados

Matriz de Confusión:

La matriz de confusión muestra la cantidad de verdaderos negativos, falsos negativos, verdaderos positivos y falsos positivos:

- 7479 verdaderos negativos (TN)
- 396 falsos negativos (FN)
- 2303 falsos positivos (FP)
- 1326 verdaderos positivos (TP)

Sensibilidad y Especificidad:

- La sensibilidad (True Positive Rate) es la proporción de verdaderos positivos correctamente identificados por el modelo.
- La especificidad (True Negative Rate) es la proporción de verdaderos negativos correctamente identificados por el modelo.
- La suma de la sensibilidad y la especificidad es 1.53460241629901, lo cual indica un buen equilibrio entre ambas métricas.

2da forma: Evaluación del Modelo Naive Bayes con busqueda de Parámetros

2da forma: Mediante Paquete naivebayes que nos permite configuracion de parametros laplace, usekernel y adjust. Para esta forma se buscarán los mejores 3 parametros (laplace, usekernel y adjust), para estos efectos se utilizará el remuestreo k-fold CV y grilla.

2.1 Evaluacion del modelo, mediante accuracy, con output un vector de niveles

El siguiente código configura la estrategia de remuestreo k-fold cross-validation y define una grilla de hiperparámetros para entrenar un modelo Naive Bayes con el paquete `naivebayes`.

```
# Estrategia de remuestreo
cv <- trainControl(method = "cv", number = 5)

# Parámetros
hyper_grid <- expand.grid(
  laplace = c(0, 1),
  usekernel = c(TRUE, FALSE),
  adjust = seq(0.5, 3, by = 0.5)
)
```

Estrategia de Remuestreo

trainControl(method = “cv”, number = 5) establece la estrategia de remuestreo k-fold cross-validation con 5 particiones. Esta estrategia divide los datos en 5 subconjuntos, entrena el modelo en 4 de ellos y lo valida en el subconjunto restante, repitiendo este proceso 5 veces. Este método ayuda a evaluar la estabilidad y generalización del modelo.

Parámetros:

expand.grid(laplace = c(0, 1), usekernel = c(TRUE, FALSE), adjust = seq(0.5, 3, by = 0.5)) crea una grilla de hiperparámetros que especifica las combinaciones de valores posibles para laplace, usekernel y adjust. Esta grilla será utilizada para buscar los mejores valores de estos parámetros durante el entrenamiento del modelo:

- laplace: Suavizado de Laplace, con valores 0 y 1.
- usekernel: Uso de kernel para estimar las densidades de probabilidad, con valores TRUE y FALSE.
- adjust: Parámetro de ajuste para la estimación del kernel, con valores en el rango de 0.5 a 3 en incrementos de 0.5.

Importancia en el Contexto de Naive Bayes

Remuestreo k-fold CV:

Utilizar k-fold cross-validation ayuda a obtener una estimación más robusta del rendimiento del modelo, mitigando el riesgo de sobreajuste y subajuste al evaluar el modelo en diferentes particiones del conjunto de datos.

Grilla de Hiperparámetros:

La búsqueda en grilla permite explorar sistemáticamente las combinaciones de los hiperparámetros laplace, usekernel y adjust para encontrar los valores que optimizan el rendimiento del modelo. Esta optimización es crucial para mejorar la precisión y la capacidad de generalización del modelo Naive Bayes.

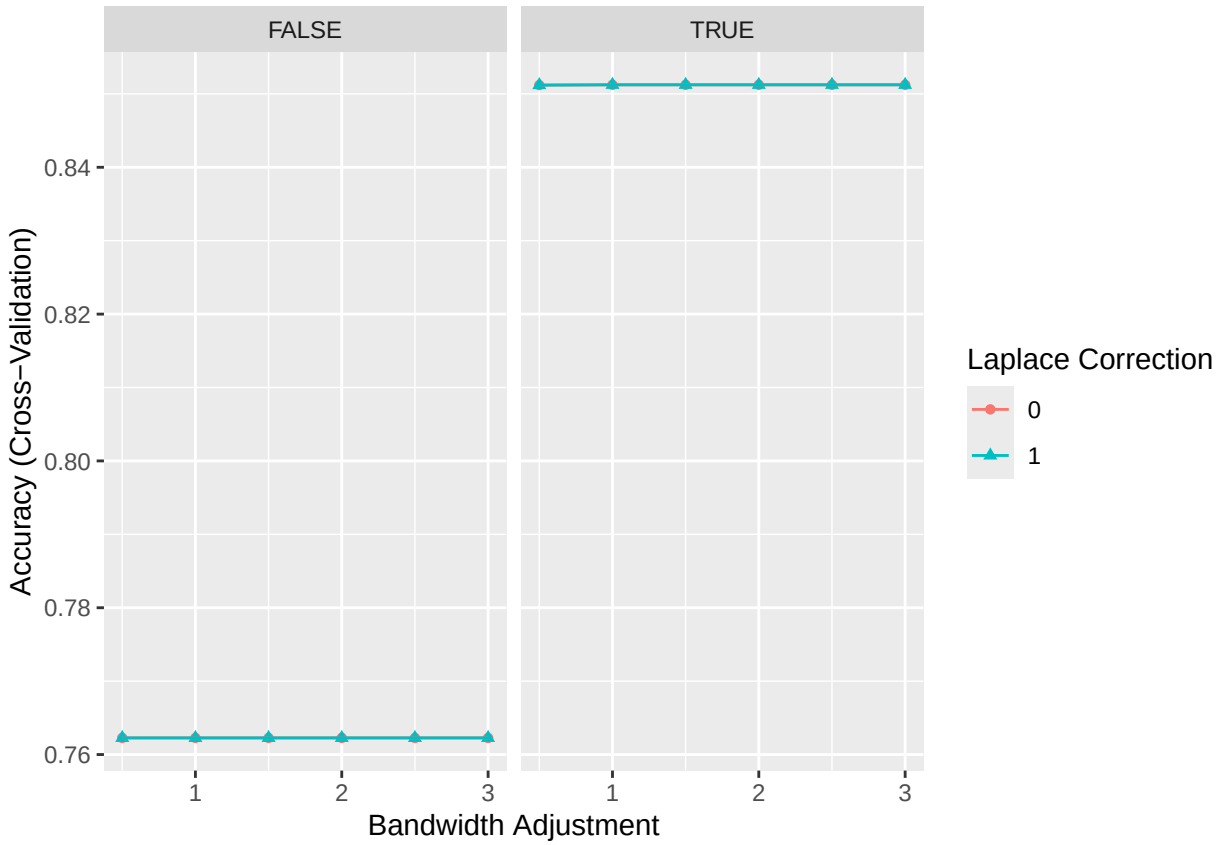
El siguiente código muestra cómo entrenar un modelo Naive Bayes utilizando el paquete `naivebayes` con remuestreo k-fold cross-validation y una grilla de hiperparámetros.

```
# Entrenamiento del modelo Naive Bayes con el paquete naivebayes
# toma 25 segundos
set.seed(12345)
naive_bayes_fit <- train(
  repay_fail ~ .,
```

```
  data = loan_data_train,
  method = "naive_bayes",
  metric = "Accuracy",
  trControl = cv,
  tuneGrid = hyper_grid
)

# Resultados
naive_bayes_fit
```

```
## Naive Bayes
##
## 26841 samples
##    33 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 21473, 21472, 21473, 21473, 21473
## Resampling results across tuning parameters:
##
##   laplace  usekernel  adjust  Accuracy   Kappa
##   0        FALSE      0.5     0.7622666  0.360051055
##   0        FALSE      1.0     0.7622666  0.360051055
##   0        FALSE      1.5     0.7622666  0.360051055
##   0        FALSE      2.0     0.7622666  0.360051055
##   0        FALSE      2.5     0.7622666  0.360051055
##   0        FALSE      3.0     0.7622666  0.360051055
##   0         TRUE      0.5     0.8511978  0.009679496
##   0         TRUE      1.0     0.8512350  0.010096310
##   0         TRUE      1.5     0.8512350  0.010096310
##   0         TRUE      2.0     0.8512350  0.010096310
##   0         TRUE      2.5     0.8512350  0.010096310
##   0         TRUE      3.0     0.8512350  0.010096310
##   1        FALSE      0.5     0.7622666  0.360051055
##   1        FALSE      1.0     0.7622666  0.360051055
##   1        FALSE      1.5     0.7622666  0.360051055
##   1        FALSE      2.0     0.7622666  0.360051055
##   1        FALSE      2.5     0.7622666  0.360051055
##   1        FALSE      3.0     0.7622666  0.360051055
##   1         TRUE      0.5     0.8511978  0.009679496
##   1         TRUE      1.0     0.8512350  0.010096310
##   1         TRUE      1.5     0.8512350  0.010096310
##   1         TRUE      2.0     0.8512350  0.010096310
##   1         TRUE      2.5     0.8512350  0.010096310
##   1         TRUE      3.0     0.8512350  0.010096310
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = TRUE
##  and adjust = 1.
```

```
ggplot(naive_bayes_fit)
```



Visualización de los Resultados La imagen adjunta muestra la precisión (Accuracy) para las diferentes combinaciones de los parametros laplace, usekernel y adjust.

El mejor modelo Naive Bayes fue seleccionado utilizando remuestreo k-fold cross-validation y una grilla de hiperparámetros. Los parámetros óptimos fueron laplace = 0, usekernel = TRUE y adjust = 1. La visualización de los resultados ayuda a entender cómo cada combinación de parámetros afecta la precisión del modelo.

El siguiente código muestra cómo evaluar el mejor modelo Naive Bayes utilizando los parámetros óptimos encontrados (`laplace = 0`, `usekernel = TRUE`, `adjust = 1.0`). La evaluación se realiza utilizando la métrica de precisión (accuracy).

```
# Best Model - Accuracy
set.seed(12345)
nb_acc <- naive_bayes(
  repay_fail ~ .,
  data = loan_data_train,
  laplace = 0,
  usekernel = TRUE,
  adjust = 1.0
)
```

```
# Predicciones
nb_acc_predictions <- predict(
  nb_acc,
  loan_data_test[, colnames(loan_data_test) != "repay_fail"],
  type = "class"
)
```

```r
# Resultados
results <- confusionMatrix(nb_acc_predictions, loan_data_test$repay_fail)
results$table
```

```
##            Reference
## Prediction    0    1
##          0 9782 1706
##          1    0   16
```

```r
results$overall[names(results$overall) == "Accuracy"]
```

```
##  Accuracy
## 0.8517038
```

Matriz de Confusión:

La matriz de confusión muestra la cantidad de verdaderos negativos, falsos negativos, verdaderos positivos y falsos positivos:

- 9782 verdaderos negativos (TN)
- 1706 falsos negativos (FN)
- 0 falsos positivos (FP)
- 16 verdaderos positivos (TP)

Precisión (Accuracy):

La precisión es una métrica que mide la proporción de predicciones correctas sobre el total de predicciones realizadas.

El mejor modelo Naive Bayes, configurado con los parámetros óptimos laplace = 0, usekernel = TRUE y adjust = 1.0, logra una precisión de 0.8517 en el conjunto de prueba.

2.2 Evaluacion del modelo, mediante ROC, con output un vector de probabilidades

```r
# Me aseguro de que los niveles de repay_fail sean nombres de variables válidos
loan_data_train$repay_fail <- make.names(as.factor(loan_data_train$repay_fail))
loan_data_test$repay_fail <- make.names(as.factor(loan_data_test$repay_fail))
```

El siguiente código configura la estrategia de remuestreo k-fold cross-validation con probabilidades de clase y una grilla de hiperparámetros para entrenar un modelo Naive Bayes con el paquete `naivebayes`.

```r
# Estrategia
cv <- trainControl(method = "cv", number = 5, classProbs = TRUE,
                   summaryFunction = twoClassSummary)

# Parámetros
hyper_grid <- expand.grid(
  laplace = c(0, 1),
  usekernel = c(TRUE, FALSE),
  adjust = seq(0.5, 3, by = 0.5)
)
```

Estrategia de Remuestreo:

- trainControl(method = “cv”, number = 5, classProbs = TRUE, summaryFunction = twoClassSummary) establece la estrategia de remuestreo k-fold cross-validation con 5 particiones.
- classProbs = TRUE indica que se calcularán las probabilidades de clase.

- summaryFunction = twoClassSummary especifica que se utilizará la función de resumen twoClassSummary, que calcula métricas como el área bajo la curva ROC (AUC), sensibilidad y especificidad. Parámetros:
- expand.grid(laplace = c(0, 1), usekernel = c(TRUE, FALSE), adjust = seq(0.5, 3, by = 0.5)) crea una grilla de hiperparámetros que especifica las combinaciones de valores posibles para laplace, usekernel y adjust. Esta grilla será utilizada para buscar los mejores valores de estos parámetros durante el entrenamiento del modelo:
- laplace: Suavizado de Laplace, con valores 0 y 1.
- usekernel: Uso de kernel para estimar las densidades de probabilidad, con valores TRUE y FALSE.
- adjust: Parámetro de ajuste para la estimación del kernel, con valores en el rango de 0.5 a 3 en incrementos de 0.5.

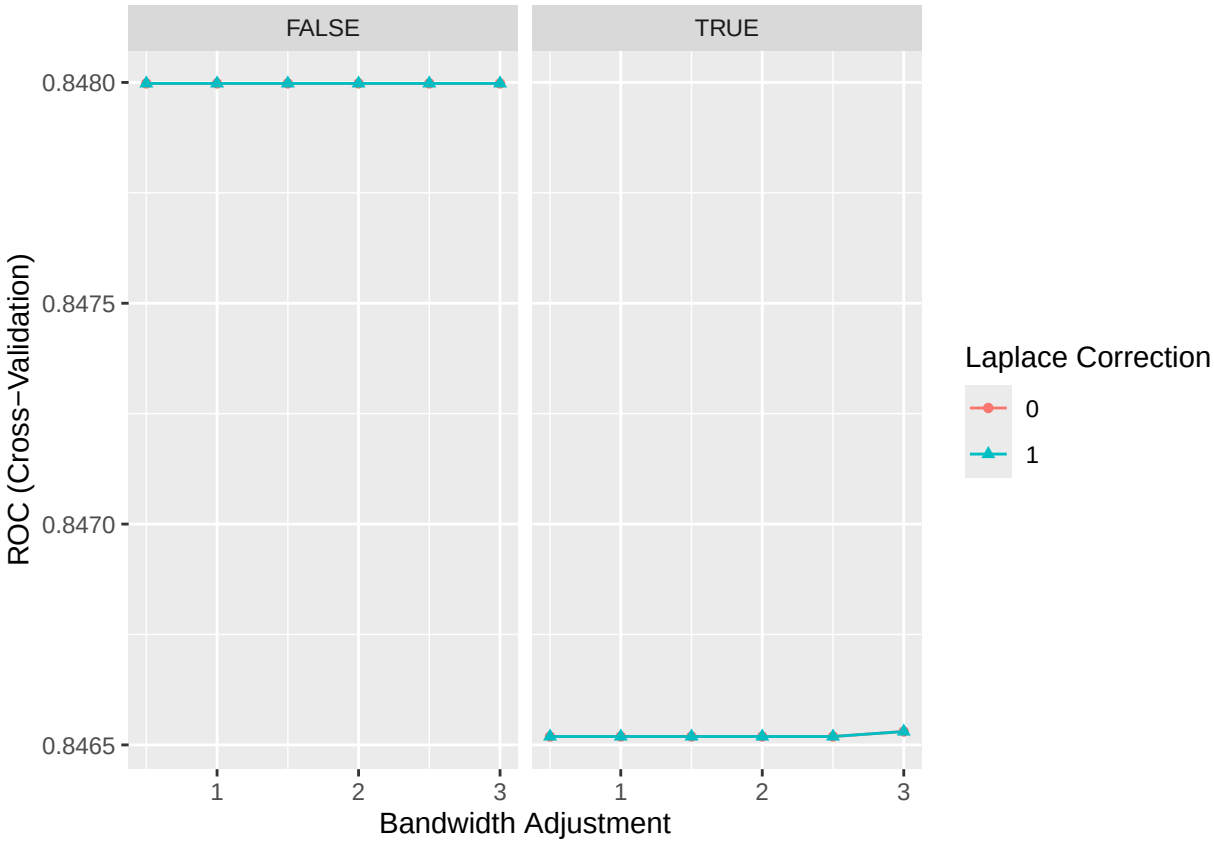
El siguiente código muestra cómo entrenar un modelo Naive Bayes utilizando el paquete `naivebayes` con remuestreo k-fold cross-validation y una grilla de hiperparámetros. La métrica de evaluación utilizada es el área bajo la curva ROC (AUC).

```
# Entrenamiento
set.seed(12345)
naive_bayes_fit <- train(
  repay_fail ~ .,
  data = loan_data_train,
  method = "naive_bayes",
  metric = "ROC",
  trControl = cv,
  tuneGrid = hyper_grid
)

naive_bayes_fit
```

```
## Naive Bayes
##
## 26841 samples
##    33 predictor
##     2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 21473, 21472, 21473, 21473, 21473
## Resampling results across tuning parameters:
##
##   laplace  usekernel  adjust  ROC        Sens       Spec
##   0        FALSE      0.5     0.8479977  0.7620485  0.763504086
##   0        FALSE      1.0     0.8479977  0.7620485  0.763504086
##   0        FALSE      1.5     0.8479977  0.7620485  0.763504086
##   0        FALSE      2.0     0.8479977  0.7620485  0.763504086
##   0        FALSE      2.5     0.8479977  0.7620485  0.763504086
##   0        FALSE      3.0     0.8479977  0.7620485  0.763504086
##   0         TRUE      0.5     0.8465190  1.0000000  0.005725420
##   0         TRUE      1.0     0.8465190  1.0000000  0.005974176
##   0         TRUE      1.5     0.8465190  1.0000000  0.005974176
##   0         TRUE      2.0     0.8465190  1.0000000  0.005974176
##   0         TRUE      2.5     0.8465190  1.0000000  0.005974176
##   0         TRUE      3.0     0.8465303  1.0000000  0.005974176
```

```
##   1        FALSE      0.5     0.8479977  0.7620485  0.763504086
##   1        FALSE      1.0     0.8479977  0.7620485  0.763504086
##   1        FALSE      1.5     0.8479977  0.7620485  0.763504086
##   1        FALSE      2.0     0.8479977  0.7620485  0.763504086
##   1        FALSE      2.5     0.8479977  0.7620485  0.763504086
##   1        FALSE      3.0     0.8479977  0.7620485  0.763504086
##   1         TRUE      0.5     0.8465190  1.0000000  0.005725420
##   1         TRUE      1.0     0.8465190  1.0000000  0.005974176
##   1         TRUE      1.5     0.8465190  1.0000000  0.005974176
##   1         TRUE      2.0     0.8465190  1.0000000  0.005974176
##   1         TRUE      2.5     0.8465190  1.0000000  0.005974176
##   1         TRUE      3.0     0.8465303  1.0000000  0.005974176
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were laplace = 0, usekernel = FALSE
##  and adjust = 0.5.
```

```
ggplot(naive_bayes_fit)
```



Visualización de los Resultados

La imagen adjunta muestra la AUC (ROC) para las diferentes combinaciones de los hiperparámetros laplace, usekernel y adjust.

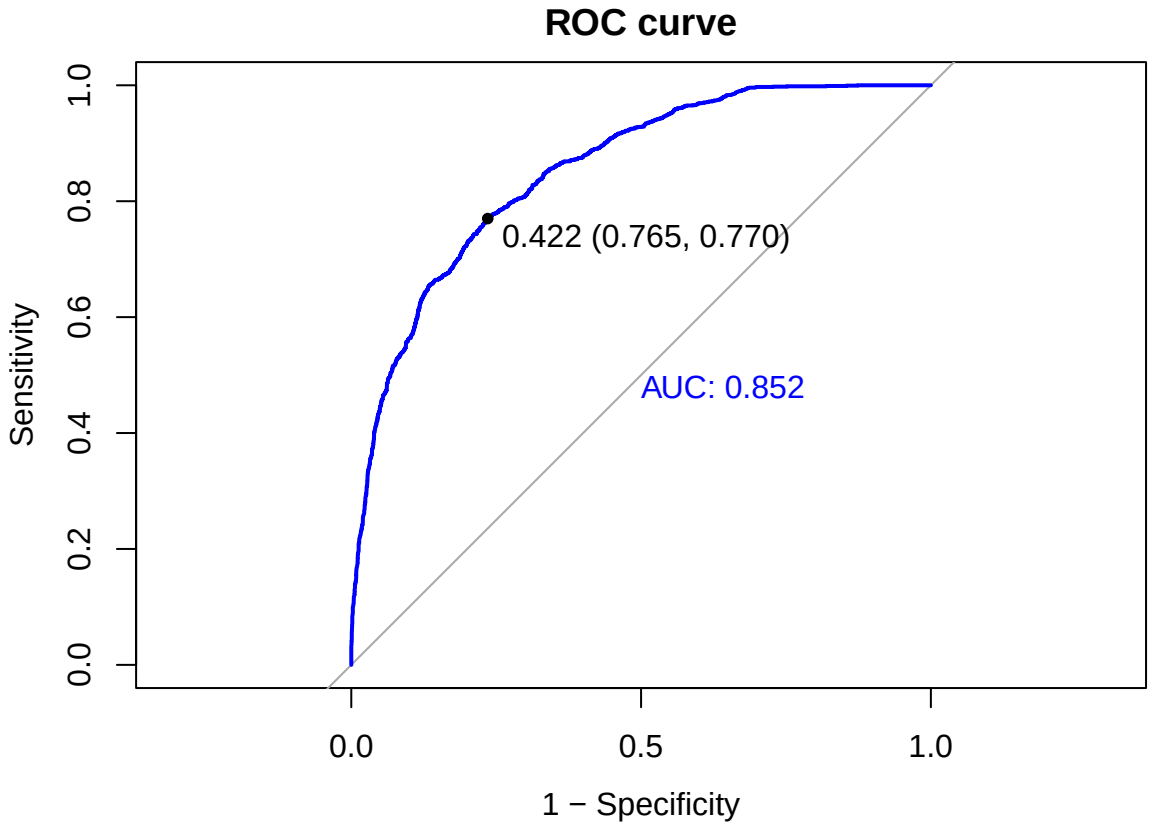
El mejor modelo Naive Bayes fue seleccionado utilizando remuestreo k-fold cross-validation y una grilla de hiperparámetros. Los parámetros óptimos fueron laplace = 0, usekernel = FALSE y adjust = 0.5.

El siguiente código muestra cómo evaluar el mejor modelo Naive Bayes utilizando los parámetros óptimos (`laplace = 0`, `usekernel = FALSE`, `adjust = 0.5`) y la métrica ROC. La evaluación se realiza utilizando

las probabilidades de clase.

```
# Best Model - ROC
set.seed(12345)
nb_roc <- naive_bayes(
  repay_fail ~ .,
  data = loan_data_train,
  laplace = 0,
  usekernel = FALSE,
  adjust = 0.5
)
```

```
# Predicciones de probabilidad
nb_roc_prob <- predict(
  nb_roc,
  loan_data_test[, colnames(loan_data_test) != "repay_fail"],
  type = "prob"
)
nb_roc_prob <- as.numeric(nb_roc_prob[, "X1"])

# ROC
nb_roc_curve <- roc(loan_data_test$repay_fail, nb_roc_prob,
                    levels = c("X0", "X1"), direction = "<")
plot.roc(nb_roc_curve, main = "ROC curve", col = "blue", lwd = 2,
         legacy.axes = TRUE, print.thres = "best", print.auc = TRUE)
```



Interpretación de los Resultados

Curva ROC:

La curva ROC muestra la relación entre la sensibilidad (tasa de verdaderos positivos) y 1 - especificidad (tasa de falsos positivos) para diferentes umbrales de probabilidad. La AUC (Área Bajo la Curva) es 0.852, lo que indica un buen rendimiento del modelo en la discriminación entre las clases repay_fail = 0 y repay_fail = 1.

Umbral de Corte:

El mejor umbral de corte es 0.422, que proporciona un buen equilibrio entre sensibilidad y especificidad. Este valor es crucial para determinar a partir de qué probabilidad se clasificará un cliente como de alto riesgo de no pago. Conclusión El mejor modelo Naive Bayes, configurado con los parámetros óptimos laplace = 0, usekernel = FALSE y adjust = 0.5, logra una AUC de 0.852 en el conjunto de prueba.

Implicaciones para Predecir el No pago de Préstamos

AUC de 0.852: Un AUC de 0.852 sugiere que el modelo tiene una alta capacidad para distinguir entre los clientes que fallarán en el pago del préstamo y aquellos que no lo harán. Este valor indica un buen rendimiento del modelo y una capacidad efectiva para identificar el riesgo de no pago.

Umbral de Corte de 0.422: El umbral de corte de 0.422 significa que cualquier cliente con una probabilidad de no pago mayor o igual a 0.422 será clasificado como de alto riesgo. Esto ayuda a la institución financiera a tomar decisiones informadas sobre la aprobación de préstamos y la gestión del riesgo, enfocándose en medidas preventivas para clientes con mayor probabilidad de incumplimiento.

El siguiente código muestra cómo calcular el punto de corte óptimo basado en la suma de sensibilidades y especificidades, convertir las probabilidades a clases binarias usando este punto de corte, y evaluar el modelo Naive Bayes utilizando la matriz de confusión.

```
# Calculo el punto de corte óptimo basado en la suma
# de sensibilidades y especificidades
aux <- nb_roc_curve$sensitivities + nb_roc_curve$specificities
corte <- nb_roc_curve$thresholds[which(aux == max(aux))]

# Convierto las probabilidades a clases binarias usando el punto de corte óptimo
nb_roc_class <- ifelse(nb_roc_prob < corte, 0, 1)
nb_roc_class <- factor(nb_roc_class, levels = c(0, 1), labels = c("X0", "X1"))
```

```
# Me aseguro que `loan_data_test$repay_fail` tenga los mismos niveles
loan_data_test$repay_fail <- factor(loan_data_test$repay_fail,
                                    levels = c("X0", "X1"))

# Resultados
results <- confusionMatrix(nb_roc_class, loan_data_test$repay_fail)
results$table
```

```
##           Reference
## Prediction   X0   X1
##         X0 7479  396
##         X1 2303 1326
```

```
sensitivity(nb_roc_class, loan_data_test$repay_fail) +
  specificity(nb_roc_class, loan_data_test$repay_fail)
```

```
## [1] 1.534602
```

Matriz de Confusión:

La matriz de confusión muestra la cantidad de verdaderos negativos, falsos negativos, verdaderos positivos y falsos positivos:

- 7479 verdaderos negativos (TN)
- 396 falsos negativos (FN)

- 2303 falsos positivos (FP)
- 1326 verdaderos positivos (TP)
- Sensibilidad y Especificidad:

La sensibilidad (True Positive Rate) es la proporción de verdaderos positivos correctamente identificados por el modelo.

La especificidad (True Negative Rate) es la proporción de verdaderos negativos correctamente identificados por el modelo.

La suma de la sensibilidad y la especificidad es 1.53460241629901, lo cual indica un buen equilibrio entre ambas métricas.

El mejor modelo Naive Bayes, evaluado mediante la curva ROC y ajustado con el punto de corte óptimo, logra un buen rendimiento en términos de sensibilidad y especificidad.

Implicaciones para Predecir el No Pago de Préstamos

Suma de Sensibilidad y Especificidad: Una suma de 1.5346 indica un buen equilibrio entre la capacidad del modelo para identificar correctamente tanto los clientes que no pagarán (alta sensibilidad) como aquellos que sí pagarán (alta especificidad).

Conclusiones y Comparación de Resultados Modelo Naive Bayes

Interpretación:

- **AUC de 0.852** indica un rendimiento igual al del primer modelo (paquete e1071) en términos de capacidad discriminativa.
- **Umbral de Corte de 0.422** proporciona el mismo umbral óptimo que en la primera forma, asegurando consistencia en la clasificación.
- **Specificity + Sensitivity:** 1.5346 indica un equilibrio adecuado igual al primer modelo (paquete e1071).

Comparación de Resultados:

1. **Paquete `e1071`, Método `naiveBayes` sin Configuración de Parámetros:**

   - **Precisión (Accuracy):** 0.7691
   - **AUC:** 0.852
   - **Specificity + Sensitivity:** 1.5346

   Este método muestra un rendimiento moderado con una precisión aceptable, pero con una cantidad significativa de falsos positivos y falsos negativos. Aunque la AUC es alta, indicando buena capacidad discriminativa, el modelo sufre de imprecisión en la clasificación de clientes.

2. **Paquete `naivebayes` con Configuración de Parámetros:**

   - **Precisión (Accuracy):** 0.8517
   - **AUC:** 0.852
   - **Specificity + Sensitivity:** 1.5346

   Este método muestra una mejora significativa en la precisión, alcanzando el 85% de exactitud. Sin embargo, aunque elimina los falsos positivos, aumenta considerablemente los falsos negativos, lo cual es problemático para identificar clientes de alto riesgo.

- Ambos modelos muestran una alta capacidad discriminativa (AUC de 0.852), pero el modelo con configuración de parámetros (`naivebayes`) proporciona una mejor precisión general.
- La elección del modelo dependerá de la prioridad de la institución financiera. Si se busca minimizar los falsos positivos para no rechazar clientes fiables, el segundo modelo es más adecuado. Sin embargo, si es crucial identificar la mayor cantidad posible de clientes de alto riesgo, se debe considerar mejorar el balance entre falsos negativos y verdaderos positivos.

Analisis Comparativo y Conclusiones k-NN v/s Naive Bayes

k-NN de Clasificacion VS Naive Bayes

En el siguiente análisis compararemos los modelos Naive Bayes y k-NN en el contexto del problema de predecir qué clientes fallarán en el pago de sus préstamos. Evaluaremos ambos modelos usando dos configuraciones: una con parámetros subóptimos definidos manualmente y otra con parámetros optimizados mediante k-fold cross-validation y búsqueda en grilla.

1. Precisión (Accuracy):

- Naive Bayes (Primera Configuración): 0.769
- Naive Bayes (Segunda Configuración): 0.852
- k-NN (Primera Configuración): 0.908
- k-NN (Segunda Configuración): 0.920

Conclusión: El modelo k-NN tiene una mayor precisión que Naive Bayes en ambas configuraciones, mostrando que k-NN es más eficaz para este conjunto de datos.

2. Evaluación mediante ROC y AUC:

Naive Bayes (Primera Configuración):

- AUC: 0.852
- Suma de Sensibilidad + Especificidad: 1.534

Naive Bayes (Segunda Configuración):

- AUC: 0.852
- Suma de Sensibilidad + Especificidad: 1.534

k-NN (Primera Configuración):

- AUC: 0.892
- Suma de Sensibilidad + Especificidad: 1.653

k-NN (Segunda Configuración):

- AUC: 0.917
- Suma de Sensibilidad + Especificidad: 1.695

Conclusión: El modelo k-NN no solo muestra una mayor precisión sino también un mejor rendimiento en términos de AUC y la suma de sensibilidad y especificidad, lo que indica una mejor capacidad para distinguir entre clientes que fallarán y no fallarán en el pago de sus préstamos.

3. Comparación de Matrices de Confusión:

Naive Bayes (Primera Configuración):

- True Negatives (TN): 7536
- False Positives (FP): 2246
- False Negatives (FN): 410
- True Positives (TP): 1312

Naive Bayes (Segunda Configuración):

- True Negatives (TN): 9782
- False Positives (FP): 0
- False Negatives (FN): 1706
- True Positives (TP): 16

k-NN (Primera Configuración):

- True Negatives (TN): 9395

- False Positives (FP): 387
- False Negatives (FN): 674
- True Positives (TP): 1048

k-NN (Segunda Configuración):

- True Negatives (TN): 9571
- False Positives (FP): 211
- False Negatives (FN): 704
- True Positives (TP): 1018

Conclusión: La segunda configuración del modelo k-NN reduce significativamente los falsos positivos en comparación con Naive Bayes, lo cual es crucial en el contexto de predecir fallos en pagos de préstamos.

Conclusiones Finales

- En el contexto del problema de predecir si los clientes fallarán en el pago de sus préstamos, el modelo k-NN ha demostrado ser superior al modelo Naive Bayes en términos de precisión, AUC, ROC, Sensibilidad + Especificidad y la capacidad de distinguir entre clases. La optimización de los hiperparámetros mediante remuestreo k-fold cross-validation y búsqueda en grilla ha mejorado significativamente el rendimiento de ambos modelos, pero k-NN sigue superando a Naive Bayes.