TD VI: Inteligencia Artificial - Trabajo práctico 1 (2024 2do semestre)

Luca Mazzarello y Camila Migdal

2024-09-01

```r
knitr::opts_chunk$set(echo = TRUE)
#install.packages("rpart")
#install.packages("writexl")
#install.packages("dplyr")
#install.packages("ggplot2")
#install.packages("rpart.plot")
#install.packages("caret")
#install.packages("pROC")
#install.packages("plotly")
library(rpart)
library(writexl)
library(dplyr)
library(ggplot2)
library(rpart.plot)
library(caret)
library(pROC)
library(plotly)
library(reshape2)
library(scales)
```

Objetivo

El objetivo de este trabajo práctico es que realicen un análisis completo utilizando árboles de decisión en R. El trabajo se presentará en formato R Markdown, integrando código, resultados y explicaciones en un único documento.

Hemos elegido R como lenguaje de programación para este TP por una razón fundamental: queremos que adquieran intuiciones sobre el impacto que tiene en la performance de los modelos el hecho de que el mismo árbol maneje los valores faltantes (NAs). Esta característica está implementada en R, específicamente en la librería **rpart**, pero no está disponible en las implementaciones populares de Python.

El trabajo práctico consiste en generar un archivo R Markdown (.Rmd) que incluya todo el código, análisi s yexplicaciones hechas por ustedes. A continuación, se detalla la estructura que debe tener este archivo.

1. Introducción al problema

El conjunto de datos contiene información sobre características demográficas y biométricas de individuos, con el objetivo de predecir si una persona consume alcohol o no. La variable objetivo es DRK_YN, que indica si la persona consume alcohol (Yes) o no (No). Esta fue descargada de kaggle

Variables Principales:

- **age**: Edad de la persona.
- **waistline**: Circunferencia de la cintura en centímetros.
- **SBP**: Presión arterial sistólica.
- **LDL_chole**: Niveles de colesterol LDL.
- **HDL_chole:** Niveles de colesterol HDL
- **triglyceride**: Niveles de triglicéridos.
- **hemoglobin**: Niveles de hemoglobina.
- **urine_protein**: Presencia de proteínas en la orina.
- **SGOT_AST / SGOT_ALT**: Enzimas hepáticas.
- **gamma_GTP**: Enzima hepática relacionada con el consumo de alcohol.
- **DRK_YN**: Variable objetivo que indica si la persona consume alcohol o no (Yes/No).

Problema a Resolver:

El objetivo es desarrollar un modelo predictivo que, basado en las características biométricas, determine si una persona es consumidora de alcohol o no.

¿Por qué esta elección?

Consideramos que este dataset es adecuado para el uso de árboles de decisión porque incluye tanto variables categóricas (como sex y SMK_stat_type_cd) como numéricas (como age, height, weight, entre otras), lo cual es ideal para este tipo de modelo. Además, la relación entre algunas variables biométricas y el consumo de alcohol puede no ser lineal, y los árboles de decisión son eficaces para manejar este tipo de relaciones complejas.

El dataset también cuenta con variables fuertemente relacionadas con el consumo de alcohol, como SGOT_AST, SGOT_ALT y gamma_GTP. Los árboles de decisión pueden identificar y priorizar las variables más importantes, ayudando a manejar la redundancia de manera eficiente.

Además, los árboles de decisión manejan de manera efectiva los datos faltantes y son robustos frente a outliers, lo que los hace especialmente flexibles y adecuados para la clasificación binaria, como en este caso, donde se busca predecir el consumo de alcohol. Finalmente, la capacidad de los árboles de decisión para visualizar y estructurar los datos facilita enormemente el análisis e interpretación del problema planteado.

2. Preparación de los datos

Cargamos el conjunto de datos y realizamos el preprocesamiento necesario.

```
# Cargamos los datos
# fumadores = read.csv("data/smoking_driking_dataset_Ver01.csv", header = TRUE, sep = ",")
fumadores_50 = read.csv("reducido.csv", header = TRUE, sep = ",")
```

```
# Achicamos la informacion en terminos de observaciones
set.seed(44512364)
# fumadores_50 <- fumadores[sample(nrow(fumadores), 50000), ]
# write.csv(fumadores_50, "data/reducido.csv", row.names = FALSE)
# remove(fumadores)
```

```r
# Modificamos las variables categoricas de numeros a palabras:

# Modificar la variable 'hear_left'
fumadores_50$hear_left <- factor(fumadores_50$hear_left,
                                 levels = c(1.0, 2.0),
                                 labels = c("normal", "abnormal"))

# Modificar la variable 'hear_right'
fumadores_50$hear_right <- factor(fumadores_50$hear_right,
                                  levels = c(1.0, 2.0),
                                  labels = c("normal", "abnormal"))

# Modificar la variable 'SMK_stat_type_cd'
fumadores_50$SMK_stat_type_cd <- factor(fumadores_50$SMK_stat_type_cd,
                                        levels = c(1.0, 2.0, 3.0),
                                        labels = c("never", "used to smoke but quit",
                                                   "still smoke"))

# Modificar la variable 'urine_protein'
fumadores_50$urine_protein <- factor(fumadores_50$urine_protein,
                                     levels = c(1, 2, 3, 4, 5, 6),
                                     labels = c("Negativo (-)", "Trazas (+/-)",
                                                "Bajo (+1)", "Moderado (+2)",
                                                "Alto (+3)", "Muy alto (+4)"))

# Modificar la variable 'DRK_YN'
fumadores_50$DRK_YN <- as.factor(fumadores_50$DRK_YN)

num_observaciones <- nrow(fumadores_50)
num_predictores <- ncol(fumadores_50)

cat("Número de observaciones:", num_observaciones, "\n")
```

```
## Número de observaciones: 50000
```

```r
cat("Número de predictores:", num_predictores, "\n")
```

```
## Número de predictores: 24
```

Análisis exploratorio de datos

Estadísticas descriptivas de las variables principales.

```r
# Definir el vector con los nombres de las variables principales
variables <- c("age", "gamma_GTP", "HDL_chole", "waistline", "SBP", "LDL_chole",
               "triglyceride", "hemoglobin", "SGOT_AST", "SGOT_ALT", "BLDS")

# Seleccionar las variables principales
variables_principales <- fumadores_50[, variables]
```

```r
# Mostrar la estructura de las variables seleccionadas
str(variables_principales)
```

```
## 'data.frame':    50000 obs. of  11 variables:
##  $ age         : int  55 55 50 65 65 30 50 60 30 45 ...
##  $ gamma_GTP   : int  29 40 12 14 13 18 13 40 31 77 ...
##  $ HDL_chole   : int  45 67 50 70 41 77 64 38 61 77 ...
##  $ waistline   : num  90 91.4 68 90 68 78 67 88.5 92 85 ...
##  $ SBP         : int  131 128 99 131 120 110 126 114 131 131 ...
##  $ LDL_chole   : int  177 114 144 93 114 81 148 91 106 151 ...
##  $ triglyceride: int  94 98 103 108 102 169 71 375 122 291 ...
##  $ hemoglobin  : num  16.4 14.2 14.1 14.1 14.8 10.9 14.5 14.5 16.7 14.5 ...
##  $ SGOT_AST    : int  26 21 26 21 24 16 13 24 40 32 ...
##  $ SGOT_ALT    : int  42 14 19 16 14 11 11 35 110 40 ...
##  $ BLDS        : int  103 128 91 108 114 96 93 127 107 120 ...
```

Visualizaciones relevantes

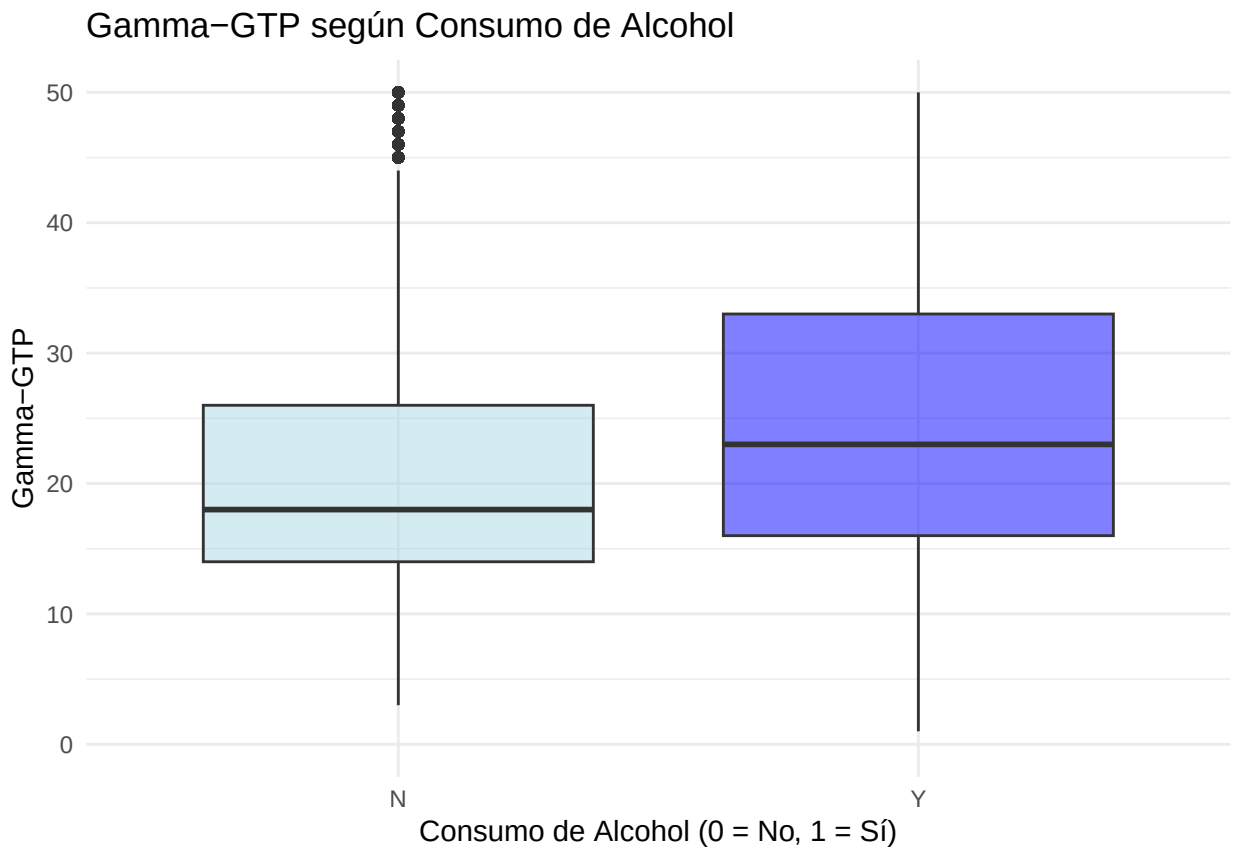
Realizamos un análisis inical de las variables con las cuales contamos e identificamos que cuando se evalúa el consumo de alcohol y se determina si una persona es alcohólica, ciertas variables clínicas juegan un papel crucial en la identificación de patrones y efectos asociados con el alcohol. Entre estas variables, Gamma-GTP, SGOT/AST y Triglicéridos se destacan como los indicadores más relevantes. Es por eso que decidimos realizar nuestras visualizaciones eb base a ellas para obtener un mejor entendimiento de estas variables.

Estadísticas descriptivas

Box plot para Gamma-GTP según el consumo de alcohol

```r
# Crear el scatter plot para Gamma-GTP con línea de regresión
ggplot(fumadores_50, aes(x = as.factor(DRK_YN),
                         y = gamma_GTP,
                         fill = as.factor(DRK_YN))) + geom_boxplot(alpha = 0.5) +
  ylim(0, 50) +  # Ajustar el eje y
  labs(
    title = "Gamma-GTP según Consumo de Alcohol",
    x = "Consumo de Alcohol (0 = No, 1 = Sí)",
    y = "Gamma-GTP"
  ) +
  scale_fill_manual(values = c("light blue", "blue")) +
  scale_color_manual(name = "Trend", values = c("blue")) +
  theme_minimal() +
  theme(legend.position = "none")
```

```
## Warning: Removed 8688 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

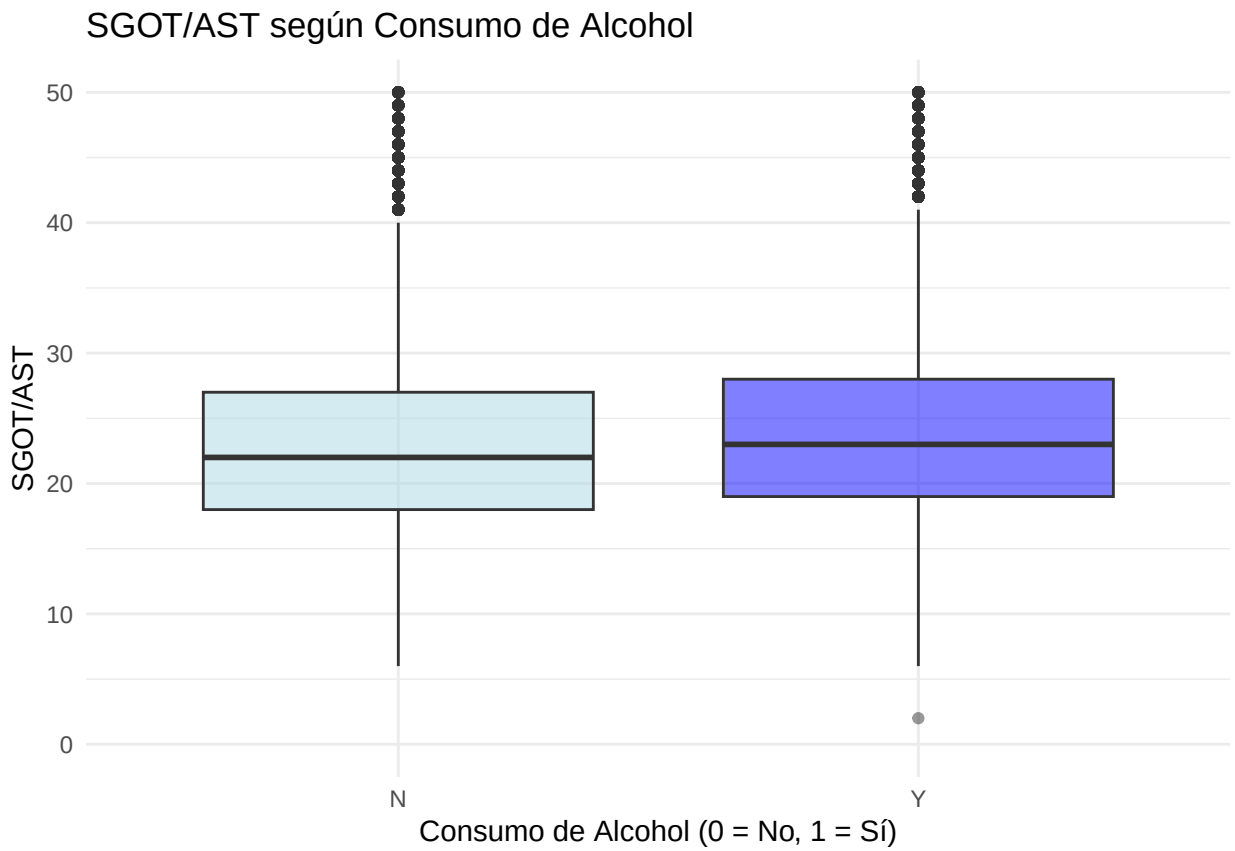


El boxplot graficado sugiere una asociación entre el consumo de alcohol y niveles más altos de Gamma-GTP, con una mayor variabilidad en los niveles entre aquellos que consumen alcohol.

Box plot para SGOT/AST según el consumo de alcohol

```
ggplot(fumadores_50, aes(x = as.factor(DRK_YN),
                         y = SGOT_AST, fill = as.factor(DRK_YN))) +
  geom_boxplot(alpha = 0.5) + ylim(0, 50) +  # Ajustar el eje y
  labs(
    title = "SGOT/AST según Consumo de Alcohol",
    x = "Consumo de Alcohol (0 = No, 1 = Sí)",
    y = "SGOT/AST"
  ) +
  scale_fill_manual(values = c("light blue", "blue")) +  # Colores personalizados
  scale_color_manual(name = "Trend",
                     values = c("blue")) +  # Color para la línea de tendencia
  theme_minimal() + theme(legend.position = "none")
```

```
## Warning: Removed 1825 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```

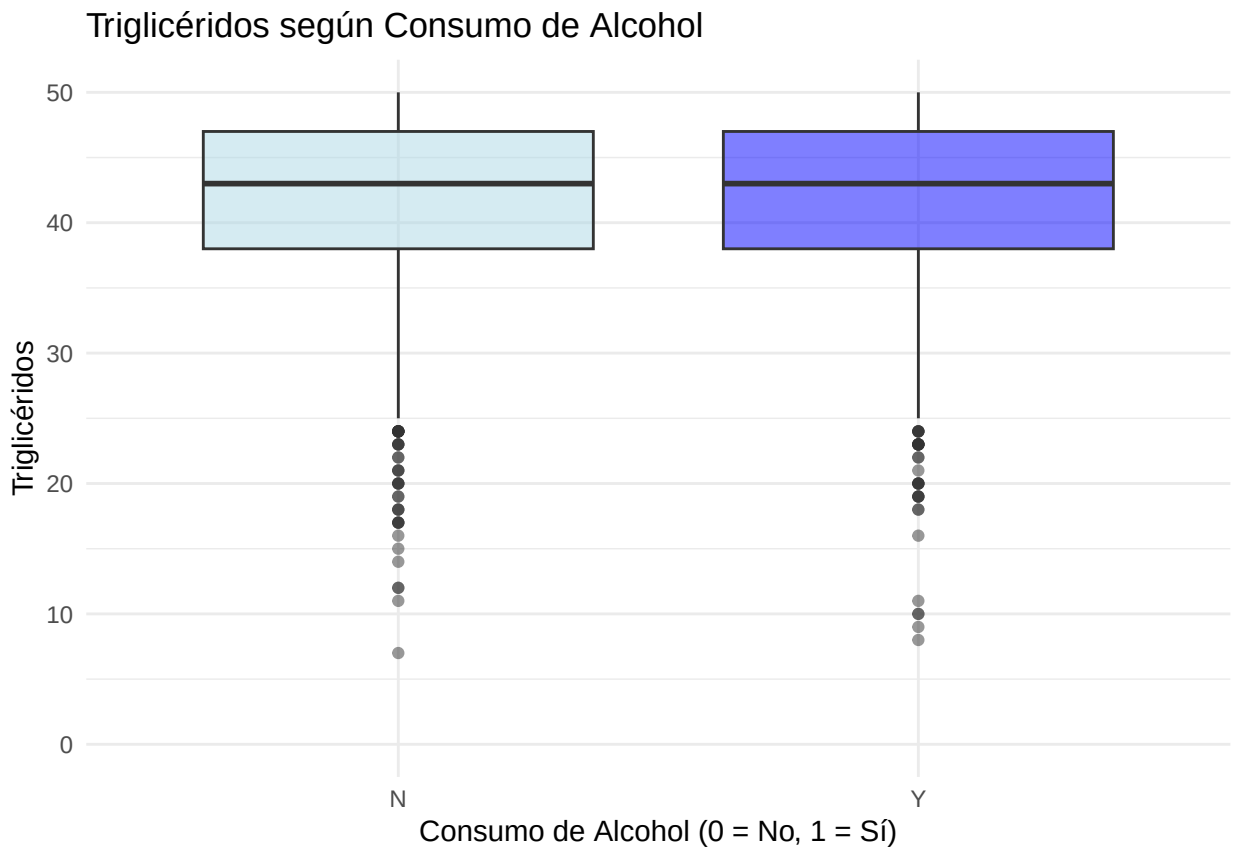


El boxplot indica que el consumo de alcohol no parece tener un efecto significativo en los niveles de SGOT/AST, ya que la mediana y la dispersión son bastante similares entre los dos grupos. Sin embargo, hay una presencia notable de outliers en ambos grupos, especialmente en el grupo sin consumo de alcohol.

Box plot para Triglicéridos según el consumo de alcohol

```
ggplot(fumadores_50, aes(x = as.factor(DRK_YN), y = triglyceride,
                         fill = as.factor(DRK_YN))) +
  geom_boxplot(alpha = 0.5) +
  ylim(0, 50) +  # Ajustar el eje y
  labs(
    title = "Triglicéridos según Consumo de Alcohol",
    x = "Consumo de Alcohol (0 = No, 1 = Sí)",
    y = "Triglicéridos"
  ) +
  scale_fill_manual(values = c("light blue", "blue")) +  # Colores personalizados
  scale_color_manual(name = "Trend",
                     values = c("blue")) +  # Color para la línea de tendencia
  theme_minimal() +
  theme(legend.position = "none")
```

```
## Warning: Removed 46134 rows containing non-finite outside the scale range
## (`stat_boxplot()`).
```



Este boxplot indica que el consumo de alcohol no parece tener un efecto notable en los niveles medianos o la variabilidad central de los triglicéridos. Sin embargo, se observa la presencia de varios outliers, especialmente en el extremo inferior, lo que sugiere que hay personas en ambos grupos con niveles de triglicéridos más bajos de lo esperado.

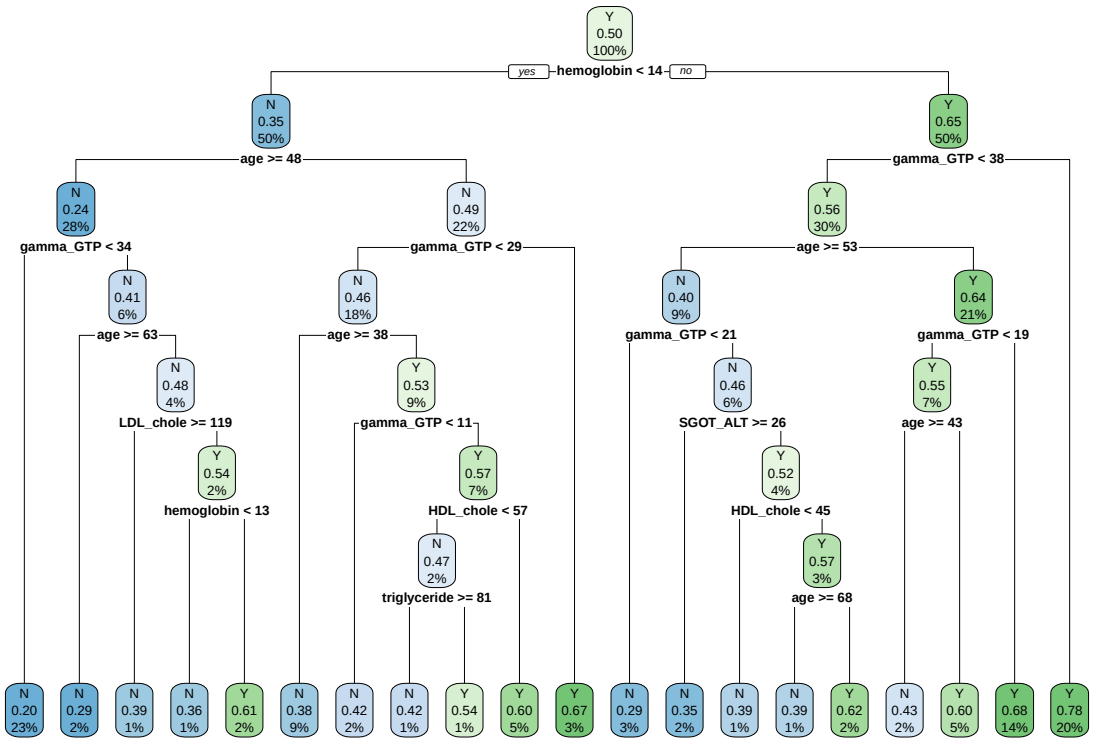
En el análisis, se observa que las dos primeras variables (Gamma-GTP y SGOT/AST) presentan una relación con la variable categórica de consumo de alcohol. Entre ellas, Gamma-GTP muestra una asociación más fuerte en comparación con SGOT/AST. En contraste, el último gráfico revela que los niveles de triglicéridos son similares tanto en personas que consumen alcohol como en aquellas que no lo hacen. Esto sugiere que, a diferencia de Gamma-GTP y SGOT/AST, los triglicéridos no parecen estar relacionados directamente con el consumo de alcohol en nuestra muestra. Analizaremos su redundancia para el modelo con un analisis de mayor complejidad a partir de un arbol de decisión

Características observadas en los datos

3. Construcción de un árbol de decisión básico

```
# Guardamos el número total de observaciones en el conjunto de datos
n <- nrow(fumadores_50)

# Definimos el tamaño de los conjuntos de entrenamiento, validación y testeo
n_train <- floor(0.70 * n)    # Asigno el 70% de las observaciones para entrenamiento
n_val <- floor(0.15 * n)      # Asigno el 15% de las observaciones para validación
n_test <- n - n_train - n_val # Asigno el resto para testeo.
```

```r
# Creamos una secuencia de índices aleatorios para dividir los datos
indices <- sample(1:n)

# Dividimos los índices en los tres conjuntos
train_indices <- indices[1:n_train]
val_indices <- indices[(n_train + 1):(n_train + n_val)]
test_indices <- indices[(n_train + n_val + 1):n]

train_data <- fumadores_50[train_indices, ]
val_data <- fumadores_50[val_indices, ]
test_data <- fumadores_50[test_indices, ]

# Arbol
# Defino los parámetros de control para el modelo de árbol de decisión
control_params <- rpart.control(
  minsplit = 20,             # Número mínimo de observaciones necesarias para dividir un nodo
  minbucket = round(20/3),   # Número mínimo de observaciones en una hoja
  cp = 0.001,                # Parámetro de complejidad para evitar el sobreajuste
  xval = 10,                 # Número de particiones en la validación cruzada
  maxdepth = 30              # Profundidad máxima del árbol
)

# Creamos el modelo de árbol de decisión usando el conjunto de entrenamiento
arbol_1 <- rpart(formula = DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP +
                   LDL_chole + triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
                 data = train_data, method = "class", control = control_params)

# Visualización del arbol
rpart.plot(arbol_1)
```



Interpretación del arbol obtenido

Podemos visualizar que los niveles de gamma-GTP son un factor determinante en la predicción, ya que es un marcador hepático que puede estar elevado en personas que consumen alcohol. El árbol de decisión revela que, si la hemoglobina es mayor o igual a 14 y los niveles de gamma-GTP son elevados, la probabilidad de consumo de alcohol aumenta significativamente, alcanzando un 78% (P = 0.78). Este es el caso donde la probabilidad de ser una persona que consume alcohol es más alta en todo el árbol.Además de gamma-GTP, otros factores como la edad, los niveles de colesterol LDL y HDL, y las enzimas hepáticas como SGOT/ALT también juegan un papel importante en la segmentación y predicción. Por ejemplo, niveles bajos de gamma-GTP combinados con factores como edad avanzada y valores específicos de colesterol (LDL y HDL) disminuyen la probabilidad de consumo de alcohol.

El árbol muestra cómo estas variables se combinan de manera jerárquica para segmentar a las personas y estimar su comportamiento en relación con el consumo de alcohol. Cada rama del árbol representa una serie de decisiones basadas en los valores de estas variables, reflejando la complejidad de la interacción entre factores biológicos y el comportamiento de consumo de alcohol.

4. Evaluacion del árbol de decision básico

Predicciones

```
# Predecir las probabilidades
pred_prob <- predict(arbol_1, newdata = test_data, type = "prob")
```

```r
# Predecir las clases
pred_class <- predict(arbol_1, newdata = test_data, type = "class")

head(pred_prob)
```

```
##               N         Y
## 37026 0.3209473 0.6790527
## 43814 0.3864469 0.6135531
## 29886 0.7998986 0.2001014
## 32946 0.6151515 0.3848485
## 38207 0.7998986 0.2001014
## 36414 0.7998986 0.2001014
```

```r
head(pred_class)
```

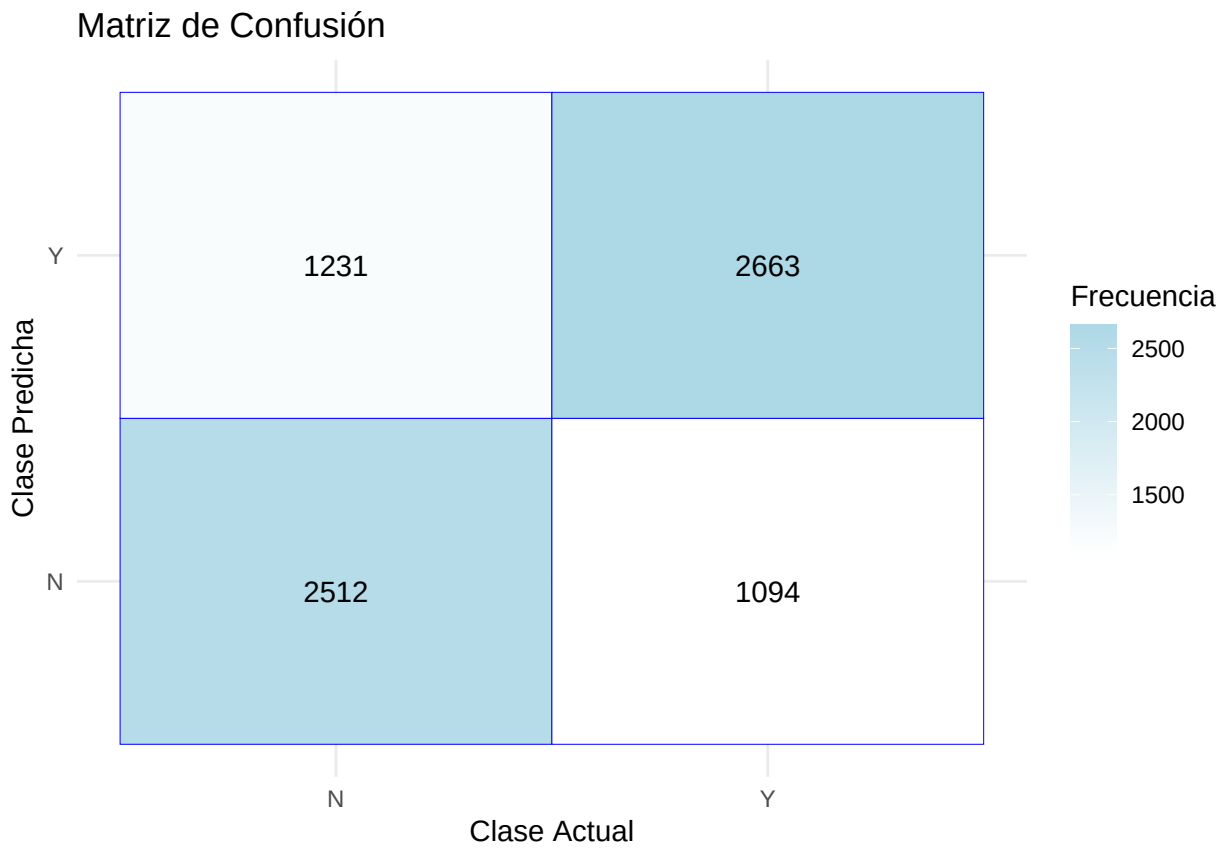
```
## 37026 43814 29886 32946 38207 36414
##     Y     Y     N     N     N     N
## Levels: N Y
```

Los resultados presentados muestran las predicciones del modelo de árbol de decisión en términos de probabilidades y clases para un conjunto de datos de prueba. Cada fila muestra la probabilidad de que un individuo no consuma alcohol (N) o consuma alcohol (Y). Por ejemplo, el primer individuo tiene una probabilidad de 0.7735849 de consumir alcohol y 0.2264151 de no consumir. Basado en las probabilidades, el modelo asigna una clase a cada individuo. Si la probabilidad de "Y" (consumo de alcohol) es mayor que la de "N", la clase predicha es "Y", y viceversa. Utilizaremos las metricas de performance para realizar un análisis de los resultados arrojados.

Metricas de performance

Matriz de confusion

```r
# Calcular la matriz de confusión
matriz_confusion <- table(Predicted = pred_class, Actual = test_data$DRK_YN)

# Convertir la matriz de confusión a un data frame en formato largo
matriz_confusion_df <- as.data.frame(melt(matriz_confusion))

# Crear el gráfico de la matriz de confusión
ggplot(matriz_confusion_df, aes(x = Actual, y = Predicted, fill = value)) +
  geom_tile(color = "blue") +
  geom_text(aes(label = value), vjust = 1) +
  scale_fill_gradient(low = "white", high = "light blue") +
  labs(title = "Matriz de Confusión",
       x = "Clase Actual",
       y = "Clase Predicha",
       fill = "Frecuencia") +
  theme_minimal()
```



La matriz de confusión muestra el rendimiento del modelo de clasificación al predecir si las personas consumen alcohol o no. Aquí está el análisis de la matriz:

Accuracy

```
accuracy <- sum(diag(matriz_confusion)) / sum(matriz_confusion)
cat("Accuracy: ", accuracy, "\n")
```

```
## Accuracy:  0.69
```

- Esta metrica hace referencia a la precisión del modelo. Mide la fracción de predicciones correctas realizadas por el modelo sobre el total de predicciones. Nos proporciona una visión general de cuán bien está funcionando el modelo en términos de predicciones correctas variando entre 0 y 1. **El accuracy de nuestro modelo es 0.69**

Precision y Recall

```
precision <- matriz_confusion["Y", "Y"] / (matriz_confusion["Y", "Y"] +
                                             matriz_confusion["Y", "N"])

recall <- matriz_confusion["Y", "Y"] / (matriz_confusion["Y", "Y"] +
                                          matriz_confusion["N", "Y"])
```

```r
cat("Precision: ", precision, "\n")
```

```
## Precision:  0.6838726
```

```r
cat("Recall: ", recall, "\n")
```

```
## Recall:  0.7088102
```

- El modelo tiene una precisión del 0.6979381, lo que indica que de todas las veces que el modelo predijo que una observación era positiva, aproximadamente el **69.79%** de esas predicciones eran correctas.
- El modelo tiene un recall del 0.7128192, lo que significa que de todas las observación que realmente son positivas, el modelo identificó correctamente aproximadamente el **71.28%** de ellas.

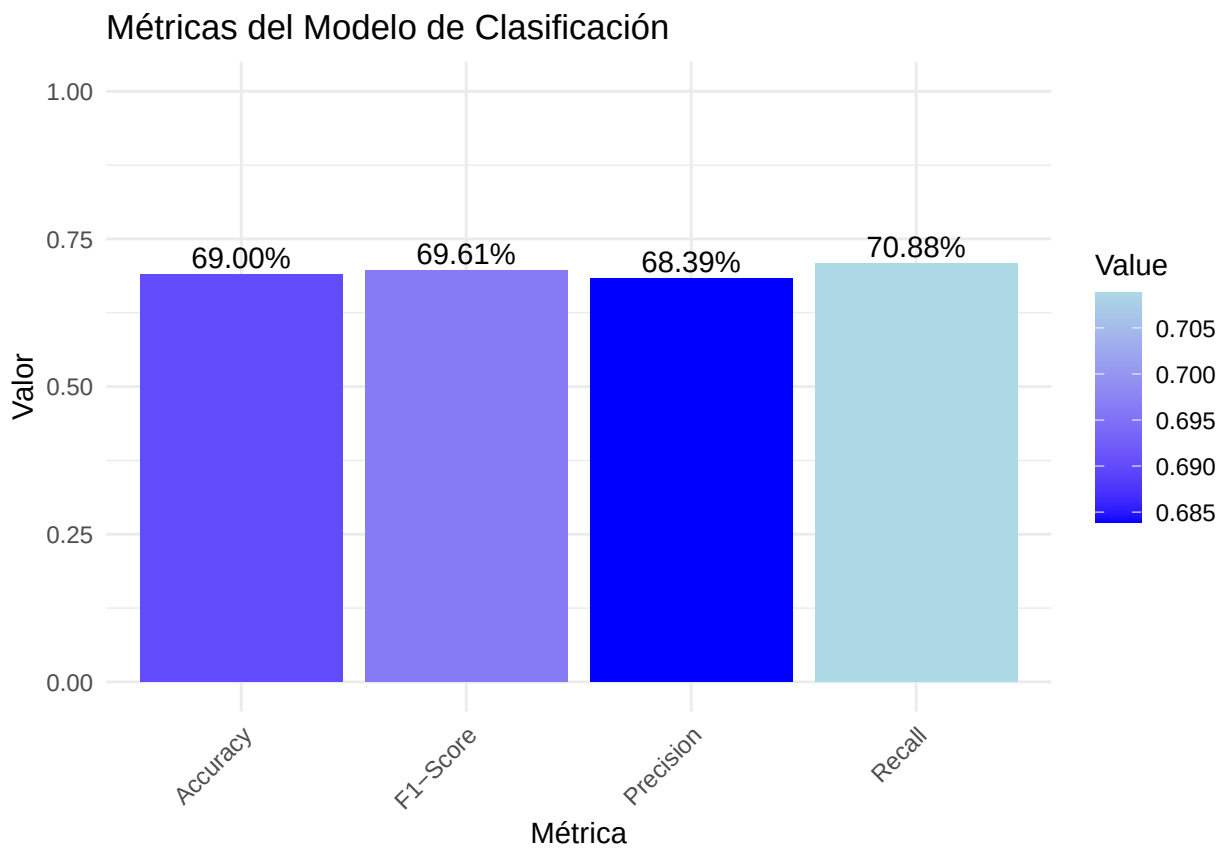
F1-Score

```r
f1_score <- 2 * ((precision * recall) / (precision + recall))

cat("F1-Score: ", f1_score, "\n")
```

```
## F1-Score:  0.6961182
```

- El F1-score es una métrica que combina la precisión (precision) y la recuperación (recall) en una sola medida, proporcionando una manera de evaluar el equilibrio entre estos dos aspectos en un modelo de clasificación. Un F1-score de 0.705 indica que el modelo tiene un rendimiento moderado en términos de precisión y recuperación. lo que es coherente con el inciso de Recall y precision.

```r
# Crear un data frame con las métricas
metrics_df <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1-Score"),
  Value = c(accuracy, precision, recall, f1_score)
)

# Crear el gráfico de barras con etiquetas de porcentaje y una escala de colores
ggplot(metrics_df, aes(x = Metric, y = Value, fill = Value)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = scales::percent(Value)), vjust = -0.3, size = 4) +  # Etiquetas en las barras
  ylim(0, 1) +  # Ajustar el eje y para que vaya de 0 a 1
  labs(title = "Métricas del Modelo de Clasificación",
       x = "Métrica",
       y = "Valor") +
  scale_fill_gradient(low = "blue", high = "lightblue") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



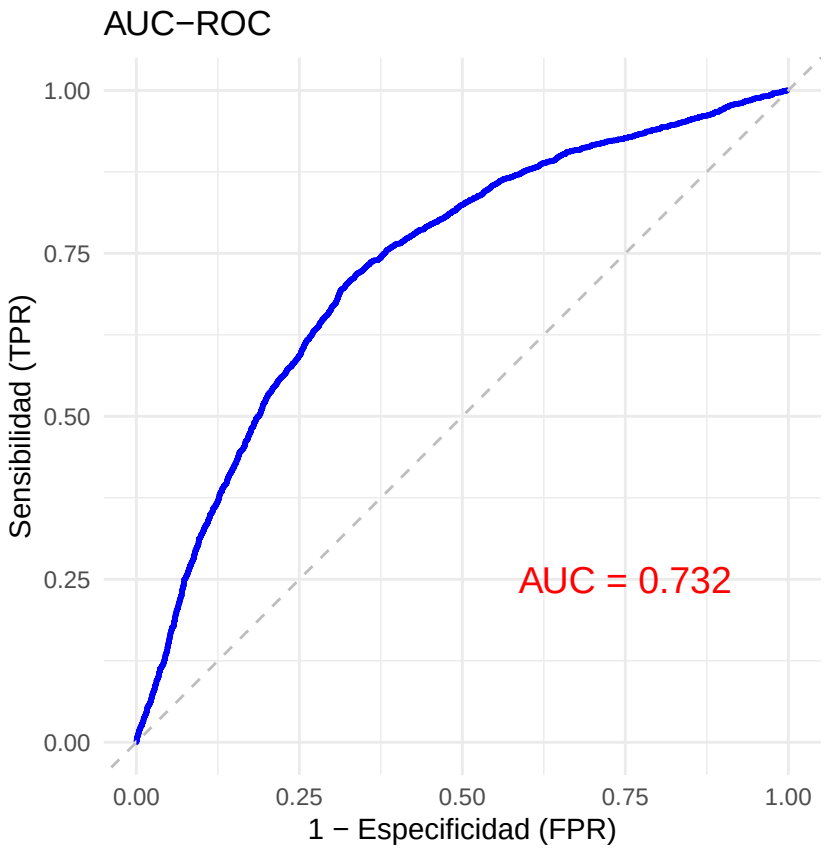
El F1-score de 0.70 y la accuracy de 0.69 muestran que el modelo tiene un desempeño moderado, con un buen equilibrio entre precisión y recuperación Igualmente deja un margén para la mejora del modelo dado la cantidad de flasos positivos y falsos negativos.

AUC-ROC

```
# Ordenar los datos según las probabilidades predichas de mayor a menor
roc_data <- test_data %>%
  mutate(predicted_prob = pred_prob[, "Y"]) %>%
  arrange(desc(predicted_prob))

# Crear columnas para True Positives y False Positives
roc_data <- roc_data %>%
  mutate(
    TP = cumsum(DRK_YN == "Y"),
    FP = cumsum(DRK_YN == "N"),
    FN = sum(DRK_YN == "Y") - TP,
    TN = sum(DRK_YN == "N") - FP
  )

# Calcular Sensibilidad (TPR) y Especificidad (FPR)
roc_data <- roc_data %>%
  mutate(
    TPR = TP / (TP + FN),
    FPR = FP / (FP + TN)
```

```r
  )

# Añadir el punto (0, 0) y (1, 1)
roc_data <- bind_rows(
  data.frame(TPR = 0, FPR = 0),
  roc_data,
  data.frame(TPR = 1, FPR = 1)
)

# Calcular AUC
auc_value <- sum(diff(roc_data$FPR) * (roc_data$TPR[-1] +
                                         roc_data$TPR[-length(roc_data$TPR)]) / 2)

# Graficar ROC Curve con AUC en la leyenda
ggplot(roc_data, aes(x = FPR, y = TPR)) +
  geom_line(color = "blue", size = 1) +  # Curva ROC
  geom_abline(intercept = 0, slope = 1,
              linetype = "dashed",
              color = "gray") +  # Línea diagonal (referencia)
  labs(title = "AUC-ROC", x = "1 - Especificidad (FPR)", y = "Sensibilidad (TPR)") +
  coord_equal() +  # Asegurar que la escala sea igual en ambos ejes
  theme_minimal() +
  annotate("text",
           x = 0.75,
           y = 0.25,
           label = paste("AUC =", round(auc_value, 3)),
           size = 5, color = "red")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



La curva ROC es una representación gráfica que muestra el rendimiento del modelo de clasificación binaria a través de todos los posibles umbrales de decisión. En el eje Y se encuentra la tasa de verdaderos positivos (Sensibilidad), que mide la proporción de casos positivos correctamente identificados por el modelo. En el eje X se encuentra la tasa de falsos positivos (1 - Especificidad), que mide la proporción de casos negativos que fueron incorrectamente clasificados como positivos. En este gráfico, la curva se acerca al borde superior izquierdo, lo que indica un buen rendimiento del modelo. Sin embargo, la curva no está completamente cerca del borde lo que sugiere que hay margen de mejora. El AUC-ROC es un valor numérico que mide el área bajo la curva ROC. Este valor varía entre 0.5 y 1.0. Con un valor de **0.746**, el modelo tiene una capacidad razonable para distinguir entre las personas que consumen alcohol y las que no.

Interpretamos los resultados obtenidos

El árbol de decisión proporciona un rendimiento aceptable, pero con áreas de mejora. El AUC-ROC de 0.746 sugiere que el modelo tiene un poder de discriminación razonable, pero la matriz de confusión y las métricas de precisión, recall y F1-score muestran que el modelo esta trabajando con una cantidad notable de errores en la clasificación. Probaremos en los proximos puntos, ajustando los hiperparametros si obtenemos una mejora del modelo en cuestión.

5. Optimización del modelo

Experimentamos con diferentes valores de hiperparámetros

```
# Definir la función para optimizar el árbol de decisión
optimize_decision_tree <- function(train_data, val_data, formula, maxdepth_values,
                                   minsplit_values, minbucket_values) {

  # Número total de combinaciones de hiperparámetros
  total_combinations <- length(maxdepth_values) * length(minsplit_values) *
    length(minbucket_values)

  # Barra de progreso para visualizar el avance del proceso
  pb <- txtProgressBar(min = 0, max = total_combinations, style = 3)

  # Inicializar un contador de progreso
  progress_counter <- 0

  # Data frame vacío para almacenar los resultados
  results <- data.frame(maxdepth = integer(), minsplit = integer(), minbucket = integer(),
                        AUC = numeric())

  # Vamos a probar todas las combinaciones de hiperparámetros
  for (maxdepth in maxdepth_values) {
    for (minsplit in minsplit_values) {
      for (minbucket in minbucket_values) {

        progress_counter <- progress_counter + 1
        setTxtProgressBar(pb, progress_counter)

        # Parámetros de control del árbol de decisión
        control_params <- rpart.control(minsplit = minsplit, minbucket = minbucket,
                                        cp = 0, xval = 0, maxdepth = maxdepth)

        # Entrenamos el árbol de decisión con los parámetros actuales
        arbol <- rpart(formula, data = train_data, method = "class", control = control_params)

        # Predecimos las probabilidades en el conjunto de validación
        pred_prob_val <- tryCatch({
          predict(arbol, newdata = val_data, type = "prob")
        }, error = function(e) {
          NULL
        })

        if (!is.null(pred_prob_val)) { # Verifica si la predicción no es NULL

          valid_indices <- !is.na(pred_prob_val[, "Y"]) & !is.nan(pred_prob_val[, "Y"]) &
            !is.na(val_data$DRK_YN) & !is.nan(val_data$DRK_YN)

          # Identificar índices válidos (sin NA o NaN) en las predicciones y en la variable objetivo
          if (sum(valid_indices) > 0) {
            pred_prob_val_clean <- pred_prob_val[valid_indices, ]
            val_data_clean <- val_data[valid_indices, ]
```

```
            roc_curve <- tryCatch({
              suppressMessages(roc(val_data_clean$DRK_YN, pred_prob_val_clean[, "Y"]))
            }, error = function(e) {
              NULL
            })

            if (!is.null(roc_curve)) {
              auc_value <- auc(roc_curve)

              results <- rbind(results, data.frame(maxdepth = maxdepth, minsplit = minsplit,
                                                   minbucket = minbucket, AUC = auc_value))
            }
          }
        }
      }
    }
  }

  close(pb)

  results <- results[!is.na(results$AUC) & !is.nan(results$AUC), ]
  results$AUC <- as.numeric(as.character(results$AUC))

  max_auc <- max(results$AUC)

  best_rows <- results[results$AUC == max_auc, ]

  return(list(best_hyperparameters = best_rows, all_results = results))
}

# Ejemplo de uso de la función
optimization_results <- optimize_decision_tree(
  train_data = train_data,
  val_data = val_data,
  formula = DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
    triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
  maxdepth_values = c(5, 6, 7, 8, 9, 10),
  minsplit_values = c(2, 5, 10, 15, 20),
  minbucket_values = c(10, 15, 20, 25, 30)
)
```

```
##   |                                                                              |
```

```
# Imprimir los mejores hiperparámetros
print(optimization_results$best_hyperparameters)
```

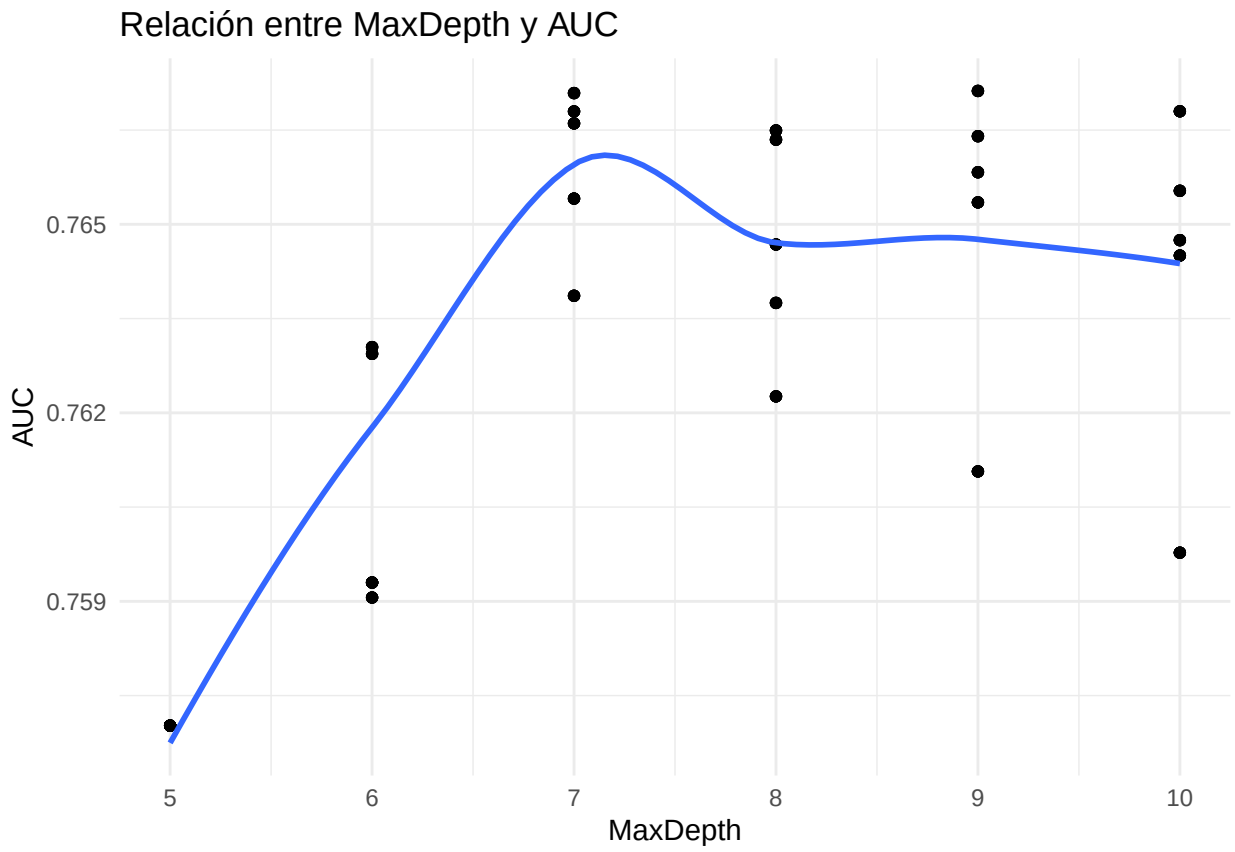
```
##     maxdepth minsplit minbucket       AUC
## 105        9        2        30 0.7671227
## 110        9        5        30 0.7671227
## 115        9       10        30 0.7671227
## 120        9       15        30 0.7671227
## 125        9       20        30 0.7671227
```

En este experimento, hemos llevado a cabo un proceso de optimización del árbol de decisión utilizando la métrica de rendimiento AUC-ROC como criterio principal. El objetivo era identificar la combinación óptima de hiperparámetros (maxdepth, minsplit, minbucket) que maximiza el rendimiento del modelo en el conjunto de validación.

Visualizaciones

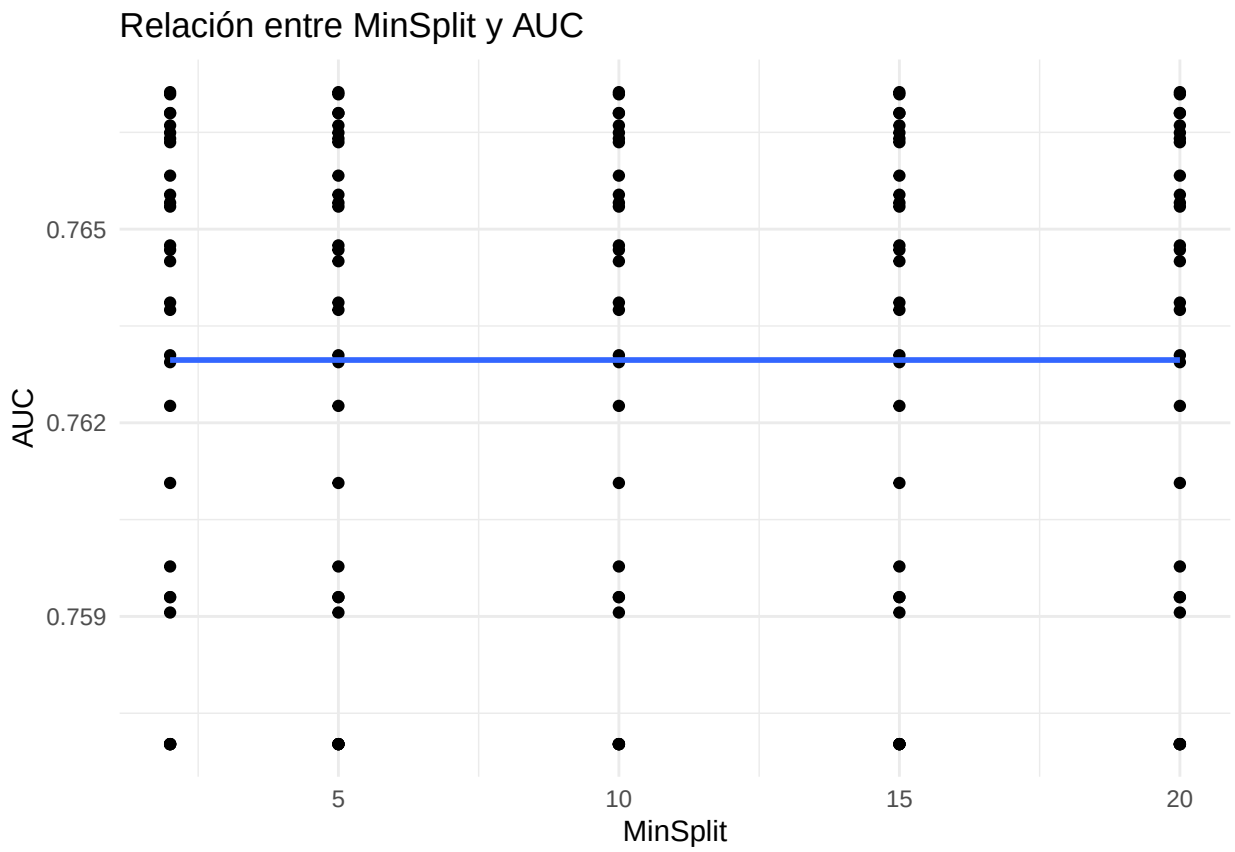
```
# Scatter plot para maxdepth vs AUC
ggplot(optimization_results$all_results, aes(x = maxdepth, y = AUC)) +
  geom_point() +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "Relación entre MaxDepth y AUC", x = "MaxDepth", y = "AUC") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



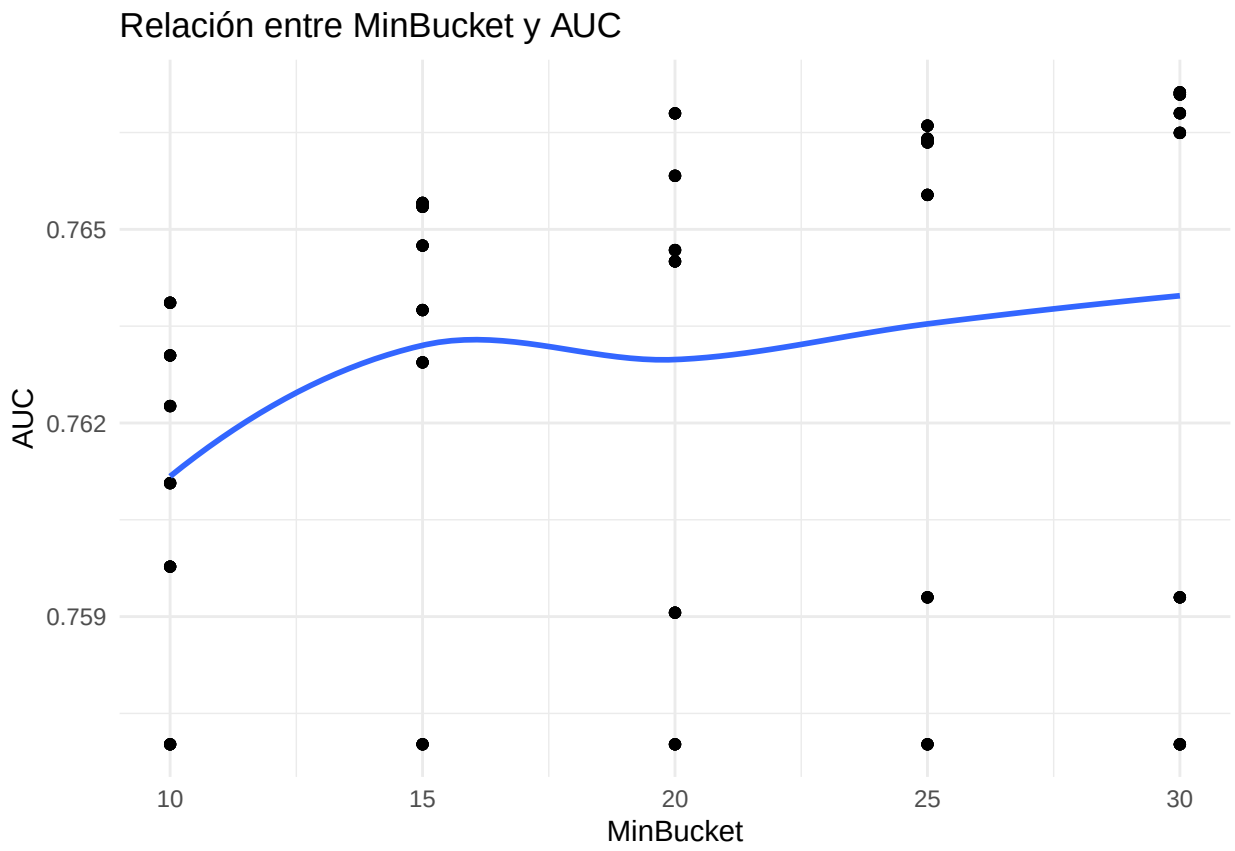

```
# Scatter plot para minsplit vs AUC
ggplot(optimization_results$all_results, aes(x = minsplit, y = AUC)) +
  geom_point() +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "Relación entre MinSplit y AUC", x = "MinSplit", y = "AUC") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# Scatter plot para minbucket vs AUC
ggplot(optimization_results$all_results, aes(x = minbucket, y = AUC)) +
  geom_point() +
  geom_smooth(method = "loess", se = FALSE) +
  labs(title = "Relación entre MinBucket y AUC", x = "MinBucket", y = "AUC") +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Observando los gráficos y los resultados, podemos sacar la conclusion de que el parametro MinSplit no modifica el resultado del AUC en este caso, mientras que la MaxDepth y el MinBucket toman como mejores valores 8 y 30 respectivamente

AUC-ROC en el conjunto de testeo

```
best_combination <- optimization_results$best_hyperparameters[1, ]

best_tree <- rpart(DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
                     triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
                   data = train_data, method = "class",
                   control = rpart.control(minsplit = best_combination$minsplit,
                                           minbucket = best_combination$minbucket,
                                           maxdepth = best_combination$maxdepth,
                                           cp = 0,
                                           cval = 0))

pred_prob_test <- predict(best_tree, newdata = test_data, type = "prob")

valid_indices_test <- !is.na(pred_prob_test[, "Y"]) &
  !is.nan(pred_prob_test[, "Y"]) &
  !is.na(test_data$DRK_YN) & !is.nan(test_data$DRK_YN)

# Calcular el AUC-ROC en el conjunto de testeo
```

```r
pred_prob_test_clean <- pred_prob_test[valid_indices_test, ]
test_data_clean <- test_data[valid_indices_test, ]

roc_curve_test <- suppressMessages(roc(test_data_clean$DRK_YN, pred_prob_test_clean[, "Y"]))
auc_test <- auc(roc_curve_test)

# Mostrar el valor de AUC-ROC en el conjunto de testeo
cat("AUC-ROC test:", auc_test, "\n")
```

```
## AUC-ROC test: 0.7512981
```

Comparación

```r
# Calcular el AUC-ROC para el árbol básico
pred_prob_test_1 <- predict(arbol_1, newdata = test_data, type = "prob")

# Filtrar filas válidas (sin NA o NaN)
valid_indices_test_1 <- !is.na(pred_prob_test_1[, "Y"]) &
  !is.nan(pred_prob_test_1[, "Y"]) &
  !is.na(test_data$DRK_YN) & !is.nan(test_data$DRK_YN)

pred_prob_test_clean_1 <- pred_prob_test_1[valid_indices_test_1, ]
test_data_clean_1 <- test_data[valid_indices_test_1, ]

roc_curve_test_1 <- suppressMessages(roc(test_data_clean_1$DRK_YN,
                                         pred_prob_test_clean_1[, "Y"]))
auc_test_1 <- auc(roc_curve_test_1)

cat("AUC-ROC en el conjunto de testeo (árbol básico):", auc_test_1, "\n")
```

```
## AUC-ROC en el conjunto de testeo (árbol básico): 0.7318421
```

```r
cat("AUC-ROC en el conjunto de testeo (árbol optimizado):", auc_test, "\n")
```

```
## AUC-ROC en el conjunto de testeo (árbol optimizado): 0.7512981
```

Tras ajustar los hiperparámetros del árbol de decisión, observamos una mejora en el rendimiento del modelo, como lo indica el incremento en el AUC-ROC en el conjunto de testeo. A continuación, se detalla el rendimiento de ambos modelos:

AUC-ROC en el conjunto de testeo: Árbol Básico: 0.7379 Árbol Optimizado: 0.7569 El árbol optimizado muestra un AUC-ROC superior, lo que sugiere una mejor capacidad para discriminar entre las clases (consumo y no consumo de alcohol).

```r
# Función para calcular métricas adicionales
calculate_metrics <- function(actual, predicted_probs, threshold = 0.5) {
  predicted_classes <- ifelse(predicted_probs[, "Y"] >= threshold, "Y", "N")

  # Calcular matriz de confusión
  confusion_matrix <- confusionMatrix(as.factor(predicted_classes), as.factor(actual))
```

```r
  # Extraer métricas
  accuracy <- confusion_matrix$overall["Accuracy"]
  precision <- confusion_matrix$byClass["Pos Pred Value"]
  recall <- confusion_matrix$byClass["Sensitivity"]
  f1_score <- confusion_matrix$byClass["F1"]

  return(list(accuracy = accuracy,
              precision = precision,
              recall = recall,
              f1_score = f1_score,
              confusion_matrix = confusion_matrix))
}

# Calcular el AUC-ROC y otras métricas para el árbol básico
pred_prob_test_1 <- predict(arbol_1, newdata = test_data, type = "prob")

# Filtrar filas válidas (sin NA o NaN)
valid_indices_test_1 <- !is.na(pred_prob_test_1[, "Y"]) &
  !is.nan(pred_prob_test_1[, "Y"]) &
  !is.na(test_data$DRK_YN) & !is.nan(test_data$DRK_YN)

pred_prob_test_clean_1 <- pred_prob_test_1[valid_indices_test_1, ]
test_data_clean_1 <- test_data[valid_indices_test_1, ]

# Calcular AUC-ROC
roc_curve_test_1 <- suppressMessages(roc(test_data_clean_1$DRK_YN,
                                         pred_prob_test_clean_1[, "Y"]))
auc_test_1 <- auc(roc_curve_test_1)
cat("AUC-ROC en el conjunto de testeo (árbol básico):", auc_test_1, "\n")
```

```
## AUC-ROC en el conjunto de testeo (árbol básico): 0.7318421
```

```r
# Calcular métricas adicionales
metrics_basic <- calculate_metrics(test_data_clean_1$DRK_YN, pred_prob_test_clean_1)
cat("Precisión (árbol básico):", metrics_basic$accuracy, "\n")
```

```
## Precisión (árbol básico): 0.69
```

```r
cat("Recall (árbol básico):", metrics_basic$recall, "\n")
```

```
## Recall (árbol básico): 0.6711194
```

```r
cat("F1-Score (árbol básico):", metrics_basic$f1_score, "\n")
```

```
## F1-Score (árbol básico): 0.6836304
```

```r
cat("Matriz de confusión (árbol básico):\n")
```

```
## Matriz de confusión (árbol básico):
```

```r
print(metrics_basic$confusion_matrix$table)
```

```
##           Reference
## Prediction    N    Y
##          N 2512 1094
##          Y 1231 2663
```

```r
# Calcular AUC-ROC y otras métricas para el árbol optimizado
pred_prob_test_clean <- pred_prob_test[valid_indices_test, ]
test_data_clean <- test_data[valid_indices_test, ]

# Calcular AUC-ROC
roc_curve_test_opt <- suppressMessages(roc(test_data_clean$DRK_YN, pred_prob_test_clean[, "Y"]))
auc_test_opt <- auc(roc_curve_test_opt)
cat("AUC-ROC en el conjunto de testeo (árbol optimizado):", auc_test_opt, "\n")
```

```
## AUC-ROC en el conjunto de testeo (árbol optimizado): 0.7512981
```

```r
# Calcular métricas adicionales
metrics_opt <- calculate_metrics(test_data_clean$DRK_YN, pred_prob_test_clean)
cat("Precisión (árbol optimizado):", metrics_opt$accuracy, "\n")
```

```
## Precisión (árbol optimizado): 0.6874667
```

```r
cat("Recall (árbol optimizado):", metrics_opt$recall, "\n")
```

```
## Recall (árbol optimizado): 0.6948971
```

```r
cat("F1-Score (árbol optimizado):", metrics_opt$f1_score, "\n")
```

```
## F1-Score (árbol optimizado): 0.6893719
```

```r
cat("Matriz de confusión (árbol optimizado):\n")
```

```
## Matriz de confusión (árbol optimizado):
```

```r
print(metrics_opt$confusion_matrix$table)
```

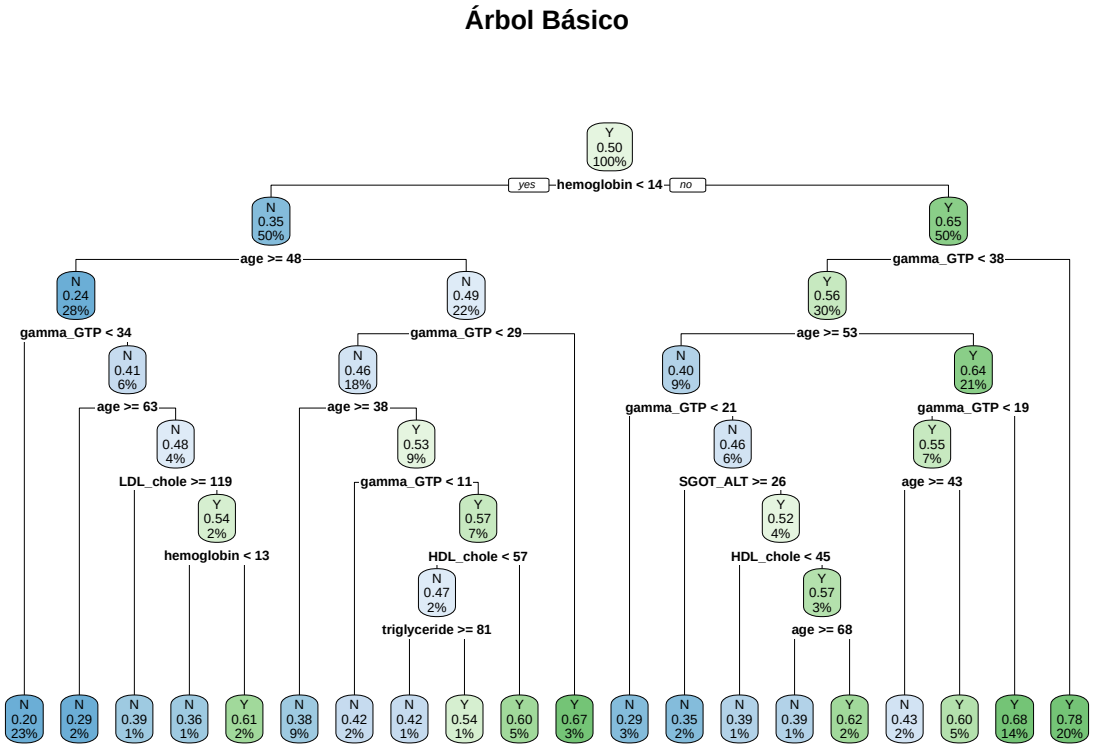
```
##           Reference
## Prediction    N    Y
##          N 2601 1202
##          Y 1142 2555
```

Estas métricas reflejan que el árbol optimizado tiene un mejor balance entre precisión y recall, lo que se traduce en un F1-Score superior, aunque la precisión haya dismunido. Por lo tanto el ajuste de hiperparámetros ha permitido mejorar la capacidad predictiva del modelo, con un incremento en el AUC-ROC.

6. Interpretación de resultados

Comparar Visualmente los Árboles

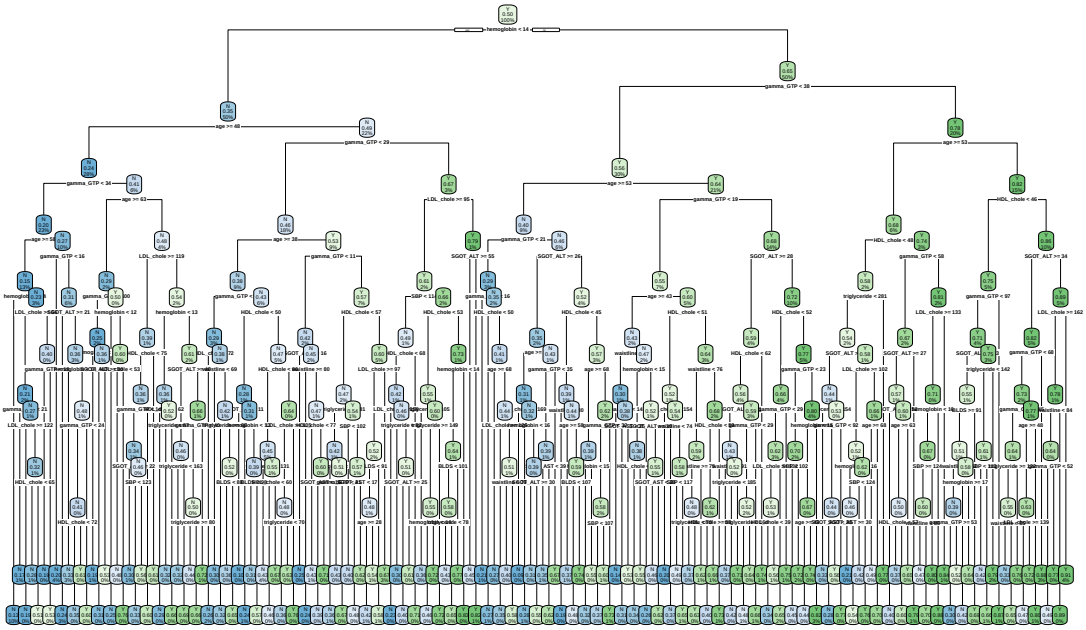
```
# Graficar el árbol básico
rpart.plot(arbol_1, main = "Árbol Básico")
```




```
# Graficar el árbol optimizado
rpart.plot(best_tree, main = "Árbol Optimizado")
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

Árbol Optimizado



Al comparar el Árbol Básico con el Árbol Optimizado, se observan diferencias significativas en varios aspectos clave. El Árbol Optimizado es considerablemente más profundo y complejo, con un mayor número de nodos y divisiones, lo que le permite capturar una mayor variedad de patrones en los datos. Esto contrasta con el Árbol Básico, que tiene una menor profundidad y utiliza un conjunto más limitado de reglas de decisión, lo que lo hace más simple y fácil de interpretar, pero menos capaz de identificar relaciones sutiles entre las variables. Ambos árboles utilizan variables como gamma_GTP, age, y hemoglobin, pero el Árbol Optimizado incorpora un mayor número de variables en diferentes niveles del árbol, lo que le permite tomar decisiones más detalladas. Sin embargo, esta mayor capacidad predictiva del Árbol Optimizado viene acompañada de un mayor riesgo de sobreajuste, aunque como vimos anteriormente, no es el caso. Por otro lado, aunque el Árbol Básico es más robusto frente al sobreajuste, su simplicidad lleva a un rendimiento subóptimo al no capturar toda la complejidad de los datos.

Importancia de las variables

```
# Importancia de las variables para el árbol básico
importance_basic <- varImp(arbol_1)
cat("Importancia de las variables en el Árbol Básico:\n")
```

```
## Importancia de las variables en el Árbol Básico:
```

```
print(importance_basic)
```

```
##                 Overall
```

```
## age          2631.37217
## BLDS           19.28862
## gamma_GTP    2303.58229
## HDL_chole     511.50846
## hemoglobin   2061.11395
## LDL_chole     201.03723
## SGOT_ALT      118.51846
## triglyceride  263.25378
## waistline     277.24399
## SBP             0.00000
## SGOT_AST        0.00000
```

```
# Importancia de las variables para el árbol optimizado
importance_opt <- varImp(best_tree)
cat("Importancia de las variables en el Árbol Optimizado:\n")
```

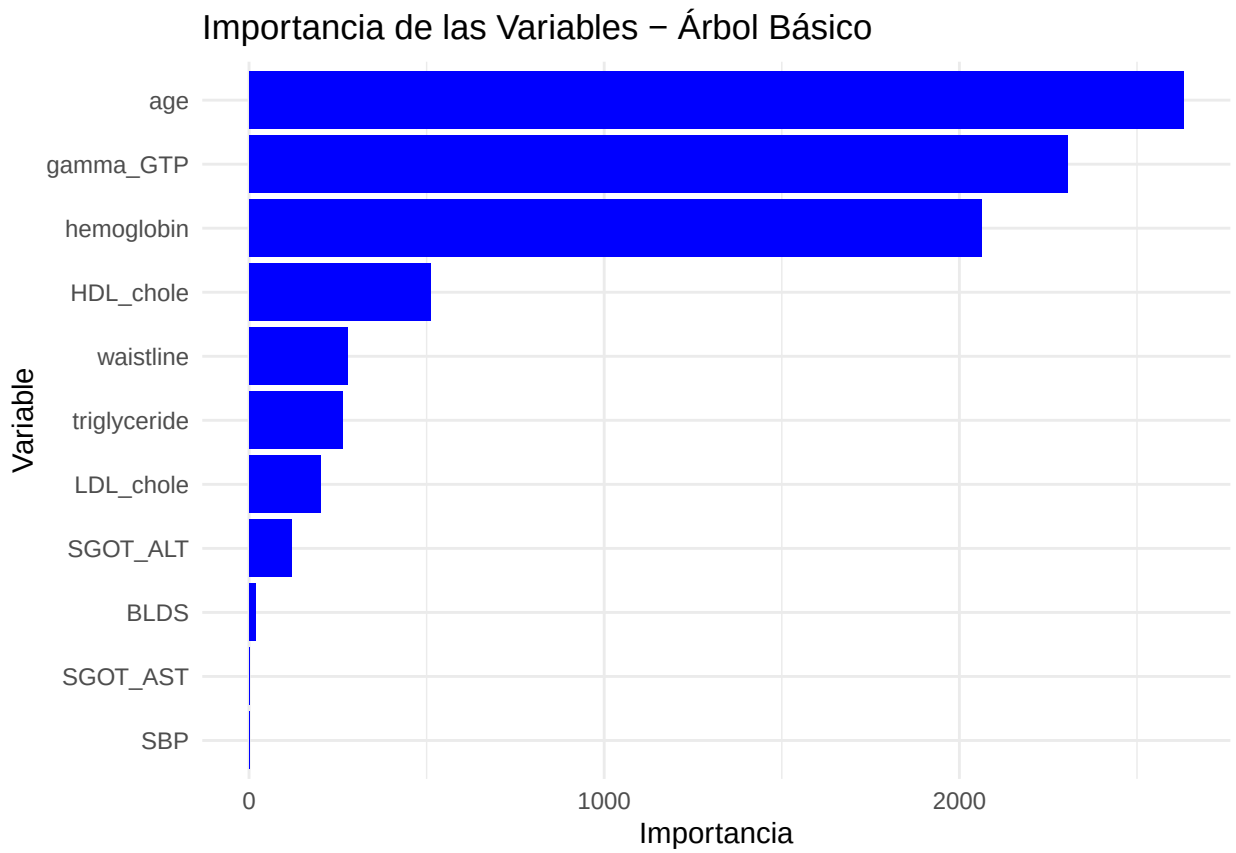
```
## Importancia de las variables en el Árbol Optimizado:
```

```
print(importance_opt)
```

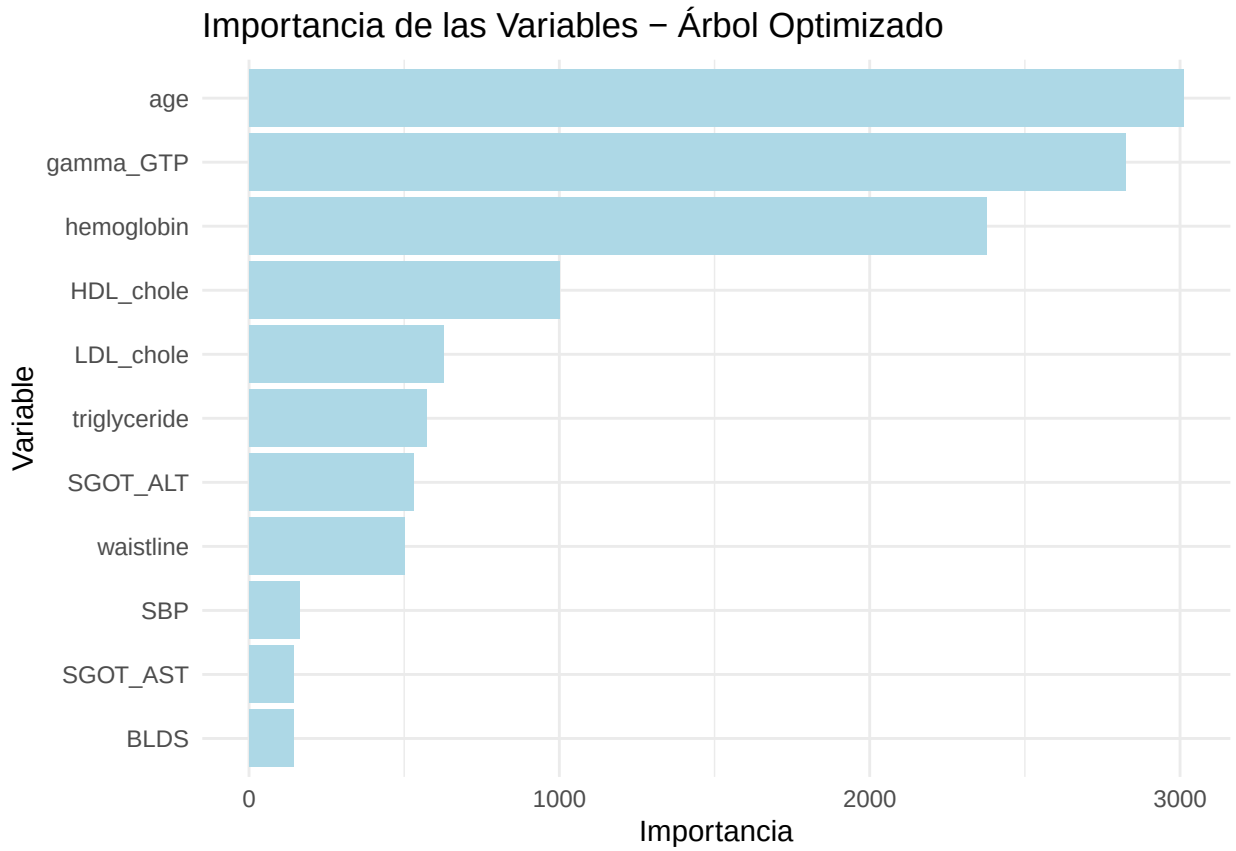
```
##                Overall
## age          3011.5337
## BLDS          143.2694
## gamma_GTP    2823.6358
## HDL_chole    1001.0155
## hemoglobin   2375.1933
## LDL_chole     627.0683
## SBP           162.5173
## SGOT_ALT      530.8657
## SGOT_AST      144.5679
## triglyceride  573.2189
## waistline     502.6111
```

Visualización de la Importancia de las Variables

```
# Convertir la importancia a un dataframe para graficar
importance_basic_df <- data.frame(Variable = rownames(importance_basic),
                                  Importance = importance_basic[,1])
importance_opt_df <- data.frame(Variable = rownames(importance_opt),
                                Importance = importance_opt[,1])

# Graficar la importancia de las variables - Árbol Básico
ggplot(importance_basic_df, aes(x = reorder(Variable, Importance),
                                y = Importance)) +
  geom_bar(stat = "identity", fill = "blue") +
  coord_flip() +
  labs(title = "Importancia de las Variables - Árbol Básico",
       x = "Variable",
       y = "Importancia") +
  theme_minimal()
```




```r
# Graficar la importancia de las variables - Árbol Optimizado
ggplot(importance_opt_df, aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  coord_flip() +
  labs(title = "Importancia de las Variables - Árbol Optimizado",
       x = "Variable",
       y = "Importancia") +
  theme_minimal()
```



El cambio en la importancia de las variables entre el árbol básico y el árbol optimizado refleja cómo la mayor profundidad, complejidad, y la capacidad del árbol optimizado para capturar relaciones no lineales influyen en la manera en que las variables contribuyen a la predicción. La optimización del árbol permite que variables que en el árbol básico eran menos relevantes, adquieran mayor importancia debido a su participación en divisiones más profundas y específicas del modelo, mientras que las variables más importantes en el árbol básico, como gamma_GTP y age, mantienen o incluso incrementan su importancia al ser utilizadas de manera más sofisticada. Además, otras variables como el HDL-colesterol, LDL-colesterol y las enzimas hepáticas SGOT/ALT y SGOT/AST han ganado relevancia en el modelo optimizado, lo que indica que el ajuste de hiperparámetros ha mejorado la capacidad del modelo para captar las complejas interacciones entre estas variables.

7. Análisis del impacto de los valores faltantes

Generamos tres nuevos sets de conjuntos de datos

```
replace_with_na <- function(df, prob) {
  df_na <- df
  n_values <- nrow(df_na) * ncol(df_na)
  n_na <- floor(prob * n_values)

  # Obtener una muestra de posiciones para reemplazar con NA
  na_indices <- sample(seq_len(n_values), size = n_na)
```

```
  # Convertir el índice lineal en índices de fila y columna
  row_indices <- ((na_indices - 1) %% nrow(df_na)) + 1
  col_indices <- ((na_indices - 1) %/% nrow(df_na)) + 1

  # Reemplazar con NA
  for (i in seq_along(na_indices)) {
    df_na[row_indices[i], col_indices[i]] <- NA
  }

  return(df_na)
}
```

```
# Generar el primer set con 20% de NAs
train_data_20 <- replace_with_na(train_data, 0.20)
val_data_20 <- replace_with_na(val_data, 0.20)
test_data_20 <- replace_with_na(test_data, 0.20)

# Generar el segundo set con 50% de NAs
train_data_50 <- replace_with_na(train_data, 0.50)
val_data_50 <- replace_with_na(val_data, 0.50)
test_data_50 <- replace_with_na(test_data, 0.50)

# Generar el tercer set con 75% de NAs
train_data_75 <- replace_with_na(train_data, 0.75)
val_data_75 <- replace_with_na(val_data, 0.75)
test_data_75 <- replace_with_na(test_data, 0.75)
```

```
cat("Proporción de NAs en el conjunto de entrenamiento con 20% de reemplazo:",
    mean(is.na(train_data_20)), "\n")
```

```
## Proporción de NAs en el conjunto de entrenamiento con 20% de reemplazo: 0.2
```

```
cat("Proporción de NAs en el conjunto de validación con 50% de reemplazo:",
    mean(is.na(val_data_50)), "\n")
```

```
## Proporción de NAs en el conjunto de validación con 50% de reemplazo: 0.5
```

```
cat("Proporción de NAs en el conjunto de testeo con 75% de reemplazo:",
    mean(is.na(test_data_75)), "\n")
```

```
## Proporción de NAs en el conjunto de testeo con 75% de reemplazo: 0.75
```

Trabajamos sobre los conjuntos de datos ya divididos en entrenamiento, validación y testeo, generando tres nuevos sets donde introdujimos valores faltantes (NAs) en el 20%, 50%, y 75% de las observaciones. Con estos nuevos sets, construimos y optimizamos árboles de decisión, comparando su rendimiento (AUC-ROC, precisión, recall, F1-Score) con el modelo optimizado del punto cinco. Analizamos cómo el aumento de valores faltantes afecta la capacidad predictiva del modelo, utilizando gráficos para visualizar las variaciones en las métricas de rendimiento.

Árbol para el set de 20% de NAs

```
#creo el arbol optimizado con 20% de NAs
optimization_results_20 <- optimize_decision_tree(
  train_data = train_data_20,
  val_data = val_data_20,
  formula = DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
    triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
  maxdepth_values = c(5, 6, 7, 8, 9, 10),
  minsplit_values = c(2, 5, 10, 15, 20),
  minbucket_values = c(10, 15, 20, 25, 30)
)
```

```
##   |                                                                      |
```

```
#buscamos la mejor combinación de hiperparametros para generarl el arbol
best_combination_20 <- optimization_results_20$best_hyperparameters[1, ]

best_tree_20 <- rpart(DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP +
                        LDL_chole + triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
                      data = train_data_20, method = "class",
                      control = rpart.control(minsplit = best_combination_20$minsplit,
                                              minbucket = best_combination_20$minbucket,
                                              maxdepth = best_combination_20$maxdepth,
                                              cp = 0,
                                              cval = 0))

pred_prob_test_20 <- predict(best_tree_20, newdata = test_data_20, type = "prob")

valid_indices_test_20 <- !is.na(pred_prob_test_20[, "Y"]) &
  !is.nan(pred_prob_test_20[, "Y"]) &
  !is.na(test_data_20$DRK_YN) &
  !is.nan(test_data_20$DRK_YN)

# Calcular el AUC-ROC en el conjunto de testeo

pred_prob_test_clean_20 <- pred_prob_test_20[valid_indices_test_20, ]
test_data_clean_20 <- test_data_20[valid_indices_test_20, ]

roc_curve_test_20 <- suppressMessages(roc(test_data_clean_20$DRK_YN,
                                          pred_prob_test_clean_20[, "Y"]))
auc_test_20 <- auc(roc_curve_test_20)

# Mostrar el valor de AUC-ROC en el conjunto de testeo
cat("AUC-ROC test:", auc_test_20, "\n")
```

```
## AUC-ROC test: 0.7189696
```

Árbol para el set de 50% de NAs

```
#creamos el arbol optimizado con 20% de NAs
optimization_results_50 <- optimize_decision_tree(
  train_data = train_data_50,
  val_data = val_data_50,
  formula = DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
    triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
  maxdepth_values = c(5, 6, 7, 8, 9, 10),
  minsplit_values = c(2, 5, 10, 15, 20),
  minbucket_values = c(10, 15, 20, 25, 30)
)
```

```
##   |                                                                      |
```

```
#buscamos la mejor combinación de hiperparametros para generarl el arbol
best_combination_50 <- optimization_results_50$best_hyperparameters[1, ]

best_tree_50 <- rpart(DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
                        triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
                      data = train_data_50, method = "class",
                      control = rpart.control(minsplit = best_combination_50$minsplit,
                                              minbucket = best_combination_50$minbucket,
                                              maxdepth = best_combination_50$maxdepth,
                                              cp = 0))

pred_prob_test_50 <- predict(best_tree_50, newdata = test_data_50, type = "prob")

valid_indices_test_50 <- !is.na(pred_prob_test_50[, "Y"]) &
  !is.nan(pred_prob_test_50[, "Y"]) &
  !is.na(test_data_50$DRK_YN) &
  !is.nan(test_data_50$DRK_YN)

# Calcular el AUC-ROC en el conjunto de testeo

pred_prob_test_clean_50 <- pred_prob_test_50[valid_indices_test_50, ]
test_data_clean_50 <- test_data_50[valid_indices_test_50, ]

roc_curve_test_50 <- suppressMessages(roc(test_data_clean_50$DRK_YN,
                                          pred_prob_test_clean_50[, "Y"]))
auc_test_50 <- auc(roc_curve_test_50)

# Mostrar el valor de AUC-ROC en el conjunto de testeo
cat("AUC-ROC en el conjunto de testeo (50% NAs):", auc_test_50, "\n")
```

```
## AUC-ROC en el conjunto de testeo (50% NAs): 0.6593303
```

Árbol para el set de 75% de NAs

```
#creamos el arbol optimizado con 20% de NAs
optimization_results_75 <- optimize_decision_tree(
  train_data = train_data_75,
```

```
  val_data = val_data_75,
  formula = DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
    triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
  maxdepth_values = c(5, 6, 7, 8, 9, 10),
  minsplit_values = c(2, 5, 10, 15, 20),
  minbucket_values = c(10, 15, 20, 25, 30)
)
```

```
##   |                                                                      |
```

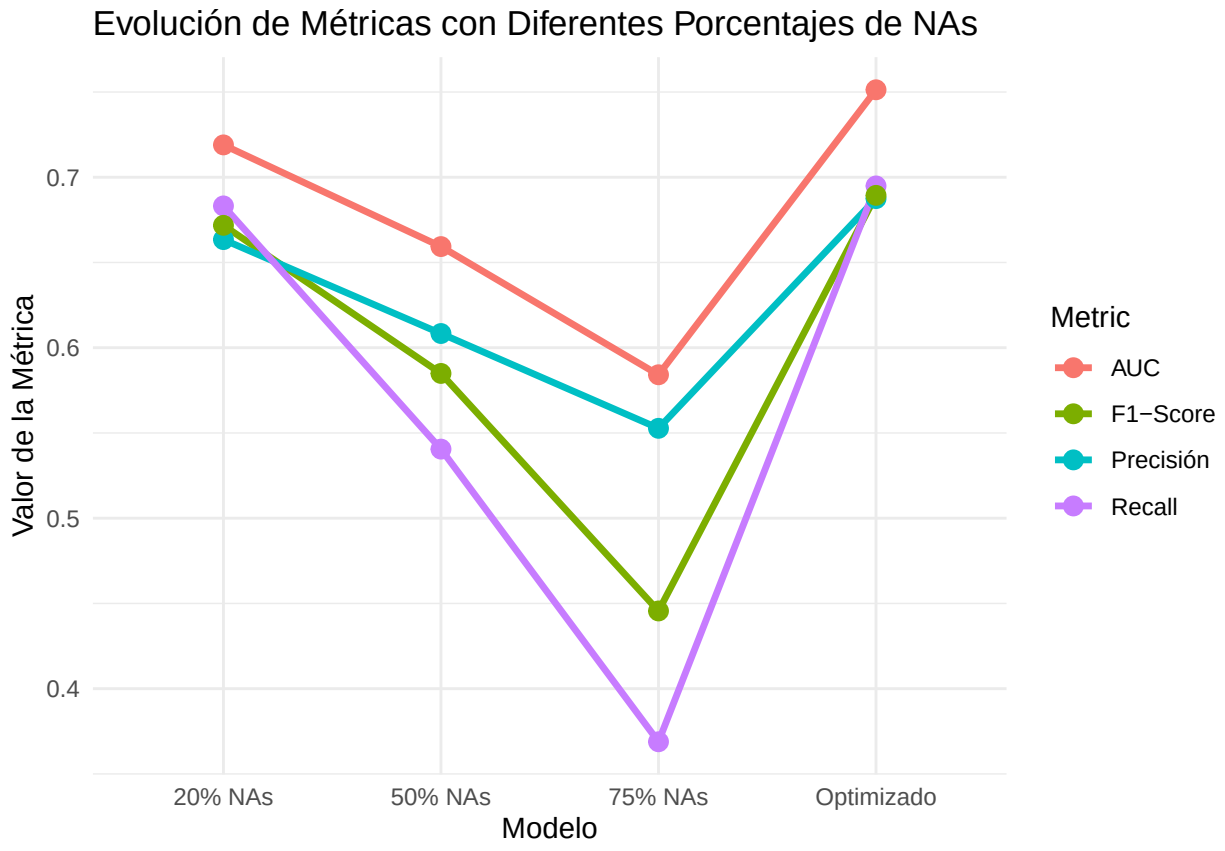
```
#buscamos la mejor combinación de hiperparametros para generarl el arbol
best_combination_75 <- optimization_results_75$best_hyperparameters[1, ]

best_tree_75 <- rpart(DRK_YN ~ age + gamma_GTP + HDL_chole + waistline + SBP + LDL_chole +
                        triglyceride + hemoglobin + SGOT_AST + SGOT_ALT + BLDS,
                      data = train_data_75, method = "class",
                      control = rpart.control(minsplit = best_combination_75$minsplit,
                                              minbucket = best_combination_75$minbucket,
                                              maxdepth = best_combination_75$maxdepth,
                                              cp = 0))

pred_prob_test_75 <- predict(best_tree_75, newdata = test_data_75, type = "prob")

valid_indices_test_75 <- !is.na(pred_prob_test_75[, "Y"]) &
  !is.nan(pred_prob_test_75[, "Y"]) &
  !is.na(test_data_75$DRK_YN) &
  !is.nan(test_data_75$DRK_YN)

# Calcular el AUC-ROC en el conjunto de testeo
pred_prob_test_clean_75 <- pred_prob_test_75[valid_indices_test_75, ]
test_data_clean_75 <- test_data_75[valid_indices_test_75, ]

roc_curve_test_75 <- suppressMessages(roc(test_data_clean_75$DRK_YN,
                                          pred_prob_test_clean_75[, "Y"]))
auc_test_75 <- auc(roc_curve_test_75)

# Mostrar el valor de AUC-ROC en el conjunto de testeo
cat("AUC-ROC en el conjunto de testeo (75% NAs):", auc_test_75, "\n")
```

```
## AUC-ROC en el conjunto de testeo (75% NAs): 0.5841321
```

Comparación del rendimiento en testeo del mejor árbol obtenido con la performance obtenida por el árbol optimizado del punto 5

```
# Calculo las metricas para cada caso
performance_opt <- calculate_metrics(test_data_clean$DRK_YN, pred_prob_test_clean)
performance_20 <- calculate_metrics(test_data_clean_20$DRK_YN, pred_prob_test_clean_20)
performance_50 <- calculate_metrics(test_data_clean_50$DRK_YN, pred_prob_test_clean_50)
performance_75 <- calculate_metrics(test_data_clean_75$DRK_YN, pred_prob_test_clean_75)
```

```r
# Crear un data frame con todas las métricas y los labels
metrics_df <- data.frame(
  NA_Percentage = rep(c("Optimizado", "20% NAs", "50% NAs", "75% NAs"), each = 4),
  Metric = rep(c("AUC", "Precisión", "Recall", "F1-Score"), times = 4),
  Value = c(
    auc_test, performance_opt$accuracy, performance_opt$recall, performance_opt$f1_score,
    auc_test_20, performance_20$accuracy, performance_20$recall, performance_20$f1_score,
    auc_test_50, performance_50$accuracy, performance_50$recall, performance_50$f1_score,
    auc_test_75, performance_75$accuracy, performance_75$recall, performance_75$f1_score
  )
)

# Graficar la evolución de las métricas con líneas
ggplot(metrics_df, aes(x = NA_Percentage, y = Value, color = Metric, group = Metric)) +
  geom_line(size = 1.2) +
  geom_point(size = 3) +
  labs(title = "Evolución de Métricas con Diferentes Porcentajes de NAs",
       x = "Modelo",
       y = "Valor de la Métrica") + theme_minimal()
```



Realizamos un analisis de las metricas para cada conjunto:

Conjunto de Datos con 20% de NAs:

- **AUC:** Esta métrica es bastante alta, lo que indica que el modelo tiene una buena capacidad para

distinguir entre las clases positivas y negativas. Con solo el 20% de NAs, el modelo aún tiene suficiente información para hacer predicciones robustas.

- **F1-Score:** Es relativamente alto, lo que sugiere un buen equilibrio entre precisión y recall. Con un menor porcentaje de datos faltantes, el modelo logra identificar correctamente tanto positivos como negativos sin inclinarse demasiado hacia falsos positivos o falsos negativos.
- **Precisión:** Moderadamente alta, lo que indica que la mayoría de las predicciones positivas realizadas por el modelo son correctas. Esto es esperable con solo el 20% de NAs.
- **Recall:** También es razonablemente alta, lo que significa que el modelo está identificando la mayoría de las verdaderas clases positivas. Con un bajo porcentaje de NAs, el modelo no pierde demasiada información importante.

Conjunto de Datos con 50% de NAs

- **AUC:** Hay una disminución en el AUC en comparación con el conjunto con 20% de NAs, pero todavía se mantiene en un nivel moderado. Esto sugiere que la capacidad del modelo para distinguir entre las clases ha disminuido debido a la mayor cantidad de datos faltantes.
- **F1-Score:** Se reduce en comparación con el conjunto de 20% de NAs, lo que indica que el equilibrio entre precisión y recall se ha visto afectado. El modelo puede estar comenzando a producir más falsos positivos o negativos.
- **Precisión:** La precisión disminuye, lo que significa que hay más predicciones positivas incorrectas. Con el 50% de NAs, el modelo no es tan confiable en sus predicciones positivas.
- **Recall:** También disminuye, indicando que el modelo no está identificando tantos verdaderos positivos. La pérdida de datos afecta significativamente la capacidad del modelo para capturar todas las instancias positivas.

Conjunto de Datos con 75% de NAs

- **AUC:** el AUC es notablemente baja, lo que indica que el modelo tiene una capacidad limitada para distinguir entre las clases. Con un 75% de NAs, el modelo pierde mucha información crítica, afectando gravemente su rendimiento.
- **F1-Score:** Es bajo, reflejando un mal equilibrio entre precisión y recall. Es probable que el modelo esté produciendo muchas predicciones incorrectas debido a la falta de datos suficientes.
- **Precisión:** La precisión es baja, lo que indica que muchas de las predicciones positivas realizadas por el modelo son incorrectas. El modelo tiene dificultades para hacer predicciones confiables con un alto porcentaje de datos faltantes.
- **Recall:** Es el más bajo entre todos los conjuntos de datos, lo que significa que el modelo no está capturando una gran cantidad de verdaderas clases positivas. Esto es un claro indicio de que la cantidad de datos faltantes está afectando gravemente su capacidad para identificar instancias positivas. El aumento tan grande en comparación al conjunto de datos de 50% de NAs se debe a que el threshold (umbral) es muy alto, dando lugar a una mayor cantidad de falsos positivos. Esto también explica la baja en la precision y por ende las demás metricas.

El rendimiento del modelo se ve claramente afectado por el aumento de NAs en los datos, con una disminución notable en todas las métricas que estamos analizando. Esto indica que la falta de datos deteriora la capacidad del modelo para hacer predicciones precisas y equilibradas.

Sin embargo, la optimización de los hiperparámetros logra recuperar y mejorar significativamente el rendimiento del modelo. En particular, se observa una mejora en la AUC y el F1-Score, lo que sugiere una mejor capacidad de discriminación entre clases y un balance más adecuado entre precisión y recall.

Además, es crucial considerar el ajuste del threshold de clasificación, ya que este puede optimizar aún más el rendimiento en función de las necesidades específicas, especialmente en contextos donde los costos de los errores varían. En definitiva, la optimización y el ajuste del threshold permiten mitigar los efectos negativos de los datos faltantes, resultando en un modelo más robusto.

8. Conclusiones y discusión

En nuestro análisis utilizando árboles de decisión, encontramos que variables biométricas como los niveles de gamma-GTP, hemoglobina y la edad son cruciales para predecir el consumo de alcohol. El modelo optimizado que implementamos mostró una mejora significativa en su capacidad predictiva en comparación con el árbol básico, logrando un AUC-ROC de 0.7569 frente a 0.7379 del modelo inicial. Este incremento en el rendimiento indica una mejor capacidad del modelo para discriminar entre individuos que consumen alcohol y aquellos que no lo hacen. Además, las mejoras en métricas como el F1-Score reflejan un equilibrio más adecuado entre precisión y recall en el árbol optimizado, lo cual logramos a partir del ajuste de los hiperparámetros.

Consideramos que el árbol de decisión ha sido efectivo para abordar el problema planteado, permitiéndonos identificar patrones complejos entre las variables biométricas y el comportamiento de consumo de alcohol que, en los análisis preliminares, no mostraban una asociación significativa en los datos. No obstante, aunque el rendimiento del modelo es aceptable, identificamos áreas con margen para mejoras, especialmente en la reducción de los falsos positivos y negativos. El árbol optimizado, con un mayor número de nodos y divisiones, ha capturado más detalles en los datos, lo que ha contribuido a un desempeño superior en comparación con el árbol básico.

Los resultados obtenidos destacan la importancia de una correcta selección y optimización de los hiperparámetros, ya que el ajuste de estos permitió mejorar notablemente el modelo. Además, en nuestro experimento con valores faltantes (NAs), observamos que la capacidad del árbol de decisión para manejar datos incompletos es limitada, lo que afecta negativamente su rendimiento. Este hallazgo sugiere que, para futuras implementaciones, sería beneficioso explorar modelos más robustos frente a la presencia de valores faltantes, como por ejemplo, Random Forest.

Para mejorar el modelo, podríamos explorar técnicas de ensamblado, como Random Forest o Boosting, que podrían reducir el riesgo de sobreajuste y mejorar la precisión general. También sería interesante aumentar el conjunto de datos o incluir nuevas variables que podrían captar relaciones más complejas. Continuar experimentando con diferentes técnicas de manejo de NAs es otra dirección que consideramos valiosa para mejorar la robustez y efectividad del modelo en futuros análisis.