

Minería de datos: PRA2 - Proyecto de minería de datos

Autor: Tim Thorp

Enero 2024

Contents

| | |
|--|-----------|
| Ejercicio 1 | 2 |
| Se genera un modelo no supervisado. | 2 |
| Se analizan, muestran y comentan las medidas de calidad del modelo generado. | 6 |
| Se comentan las conclusiones. | 7 |
| Ejercicio 2 | 7 |
| Se genera de nuevo el modelo no supervisado anterior, pero usando una métrica de distancia distinta. | 7 |
| Se muestran y comentan las medidas de calidad del modelo generado. | 10 |
| Se comparan los dos modelos no supervisados con métricas de distancia distintas. | 11 |
| Se comentan las conclusiones. | 12 |
| Ejercicio 3 | 13 |
| Se aplican los algoritmos DBSCAN y OPTICS de forma correcta. | 13 |
| Se prueban, describen e interpretan los resultados con diferentes valores de <code>eps</code> y <code>minPts</code> | 14 |
| Se obtiene una medida de lo bueno que es el agrupamiento. | 19 |
| Se comparan los resultados obtenidos de los modelos anteriores y DBSCAN. | 20 |
| Se comentan las conclusiones. | 21 |
| Ejercicio 4 | 21 |
| Se seleccionan las muestras de entrenamiento y test. | 21 |
| Se justifican las proporciones seleccionadas. | 23 |
| Ejercicio 5 | 24 |
| Se generan reglas y se seleccionan e interpretan las más significativas. | 24 |
| Se extraen las reglas del modelo en formato texto y gráfico. | 27 |
| Se genera la matriz de confusión para medir la capacidad predictiva del algoritmo, teniendo en cuenta las distintas métricas asociadas a dicha matriz (precisión, sensibilidad, especificidad...). | 29 |
| Se comparan e interpretan los resultados (sin y con opciones de poda), explicando las ventajas e inconvenientes del modelo generado respecto a otro método de construcción. | 35 |

| | |
|---|-----------|
| Se evalúa la tasa de error en cada nivel de árbol, la eficiencia en la clasificación (en las muestras de entrenamiento y test) y la comprensibilidad del resultado. | 49 |
| Se comentan las conclusiones. | 53 |
| Ejercicio 6 | 54 |
| Se prueba con una variación u otro enfoque algorítmico. | 54 |
| Se detalla, comenta y evalúa la calidad de clasificación. | 58 |
| Se comparan los resultados de manera exhaustiva con el método supervisado del ejercicio 5. | 60 |
| Ejercicio 7 | 63 |
| Se identifica qué posibles limitaciones tienen los datos que has seleccionado para obtener conclusiones con los modelos (supervisado y no supervisado) | 63 |
| Se identifican posibles riesgos del uso del modelo. | 63 |

Ejercicio 1

Se genera un modelo no supervisado.

En este análisis, utilizaremos *k-means* para identificar clústeres potenciales basándonos en la hora programada de salida (CRS_DEP_TIME_CONTINUOUS) y la duración programada del vuelo (CRS_ELAPSED_TIME).

Para comenzar, cargamos nuestro dataset de la PRA1.

```
flightData <- read.csv('../data/flightData_clean.csv', row.names=NULL, stringsAsFactors=TRUE)
str(flightData)

## 'data.frame': 560887 obs. of 19 variables:
## $ DAY_OF_WEEK : Factor w/ 7 levels "domingo","jueves",...: 3 3 3 3 3 3 3 3 3 3 ...
## $ FL_DATE : Factor w/ 30 levels "2023-09-01","2023-09-02",...: 4 4 4 4 4 4 4 4 4 4 ...
## $ OP_UNIQUE_CARRIER : Factor w/ 15 levels "Alaska Airlines",...: 5 5 5 5 5 5 5 5 5 5 ...
## $ ORIGIN : Factor w/ 340 levels "ABE","ABI","ABQ",...: 1 1 1 5 5 12 12 13 15 15 ...
## $ DEST : Factor w/ 340 levels "ABE","ABI","ABQ",...: 21 21 21 21 21 21 21 21 21 21 ...
## $ CRS_DEP_TIME : int 654 1242 1727 735 1405 610 1620 1015 1245 706 ...
## $ DEP_TIME : int 649 1237 1717 725 1400 609 1636 1010 1237 701 ...
## $ DEP_DELAY_NEW : int 0 0 0 0 0 16 0 0 0 ...
## $ CRS_ARR_TIME : int 900 1450 1938 848 1512 859 1906 1123 1428 803 ...
## $ ARR_TIME : int 842 1431 1905 820 1454 904 1913 1100 1411 759 ...
## $ ARR_DELAY_NEW : int 0 0 0 0 0 5 7 0 0 0 ...
## $ ARR_DEL15 : int 0 0 0 0 0 0 0 0 0 ...
## $ CRS_ELAPSED_TIME : int 126 128 131 73 67 109 106 68 103 57 ...
## $ ACTUAL_ELAPSED_TIME : int 113 114 108 55 54 115 97 50 94 58 ...
## $ CRS_DEP_TIME_DISCRETE : Factor w/ 24 levels "00:00-00:59",...: 7 13 18 8 15 7 17 11 13 8 ...
## $ CRS_ARR_TIME_DISCRETE : Factor w/ 24 levels "00:00-00:59",...: 9 15 20 9 16 9 20 12 15 9 ...
## $ CRS_ELAPSED_TIME_DISCRETE: Factor w/ 5 levels "Corto","Intermedio",...: 1 1 1 4 4 1 1 4 1 4 ...
## $ CRS_DEP_TIME_CONTINUOUS : int 414 762 1047 455 845 370 980 615 765 426 ...
## $ CRS_ARR_TIME_CONTINUOUS : int 540 890 1178 528 912 539 1146 683 868 483 ...
```

A continuación, seleccionamos las variables relevantes y tomamos una muestra de 10,000 observaciones para evitar un tiempo de procesamiento excesivo.

```

set.seed(123)
sample_indices <- sample(1:nrow(flightData), size = 10000)
sampled_data <- flightData[sample_indices, c("CRS_DEP_TIME_CONTINUOUS", "CRS_ELAPSED_TIME")]

```

Antes de aplicar el algoritmo, hacemos una inspección inicial de los datos. CRS_DEP_TIME_CONTINUOUS y CRS_ELAPSED_TIME se expresan en minutos, así que para mejorar la interpretación, ajustamos las etiquetas de los ejes a formato de horas.

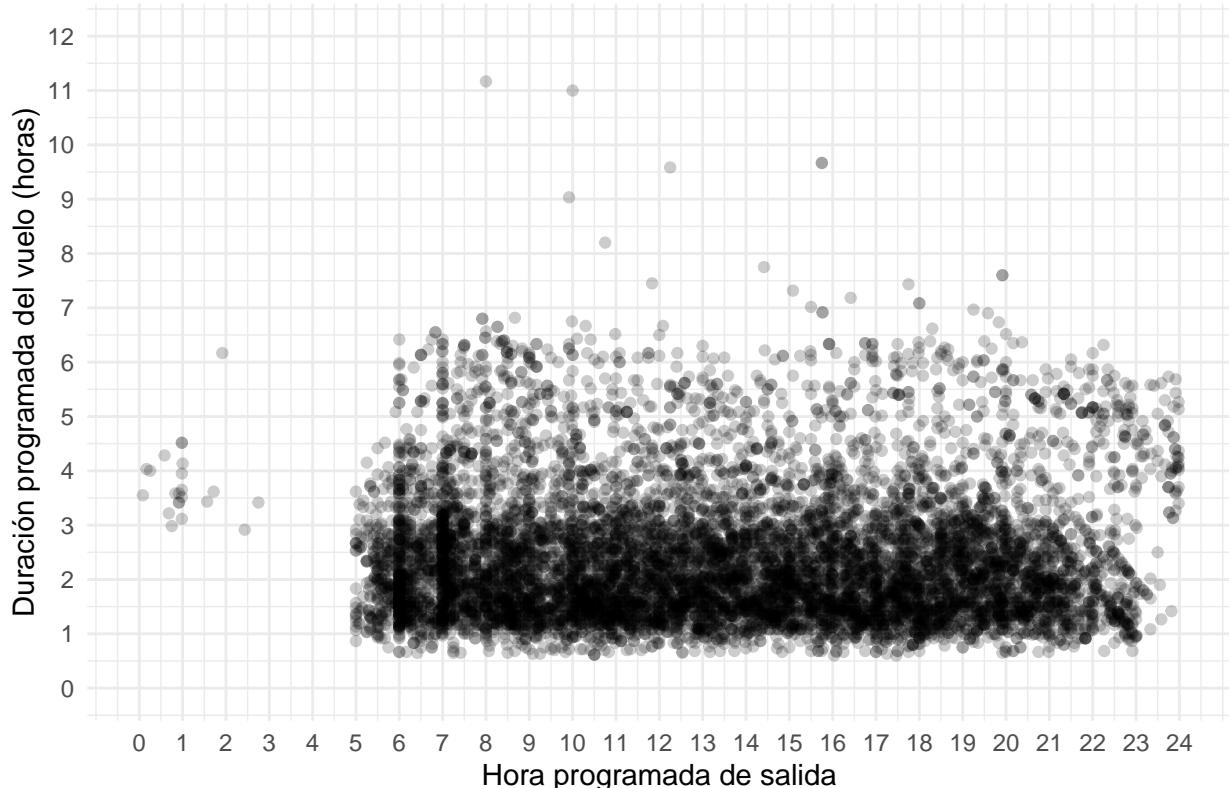
```

library(ggplot2)

ggplot(sampled_data, aes(x = CRS_DEP_TIME_CONTINUOUS, y = CRS_ELAPSED_TIME)) +
  geom_point(alpha = 0.2) +
  labs(title = "Duración del vuelo según hora de salida",
       x = "Hora programada de salida",
       y = "Duración programada del vuelo (horas)") +
  scale_x_continuous(breaks = seq(0, 1440, by = 60),
                     labels = 0:24,
                     limits = c(0, 1440)) +
  scale_y_continuous(breaks = seq(0, 720, by = 60),
                     labels = 0:12,
                     limits = c(0, 720)) +
  theme_minimal()

```

Duración del vuelo según hora de salida



Visualmente, podemos apreciar dos grupos. El primero, más pequeño y en el lado izquierdo del gráfico, representa los vuelos de madrugada con duraciones de 3–4.5 horas aproximadamente.

El segundo grupo engloba la mayoría de los vuelos, extendiéndose desde aproximadamente las 05:00 hasta la medianoche. Dentro de este grupo, se observan variaciones en la densidad, siendo más densos en la parte inferior, correspondientes a vuelos más cortos. Los puntos más aislados y superiores son vuelos más largos, como los que van del territorio continental de EE.UU. a Hawái.

Posteriormente, escalamos los datos para que ambas variables tengan igual importancia en el análisis, con una media de 0 y una desviación estándar de 1.

```
scaled_data <- as.data.frame(scale(sampled_data))
sapply(scaled_data, mean)
```

```
## CRS_DEP_TIME_CONTINUOUS      CRS_ELAPSED_TIME
##          2.863438e-17          1.683472e-16
```

```
sapply(scaled_data, sd)
```

```
## CRS_DEP_TIME_CONTINUOUS      CRS_ELAPSED_TIME
##          1                      1
```

Aplicamos *k-means*, que por defecto usa la distancia euclíadiana, y experimentamos con distintos valores de *k* (2-9).

```
library(gridExtra)

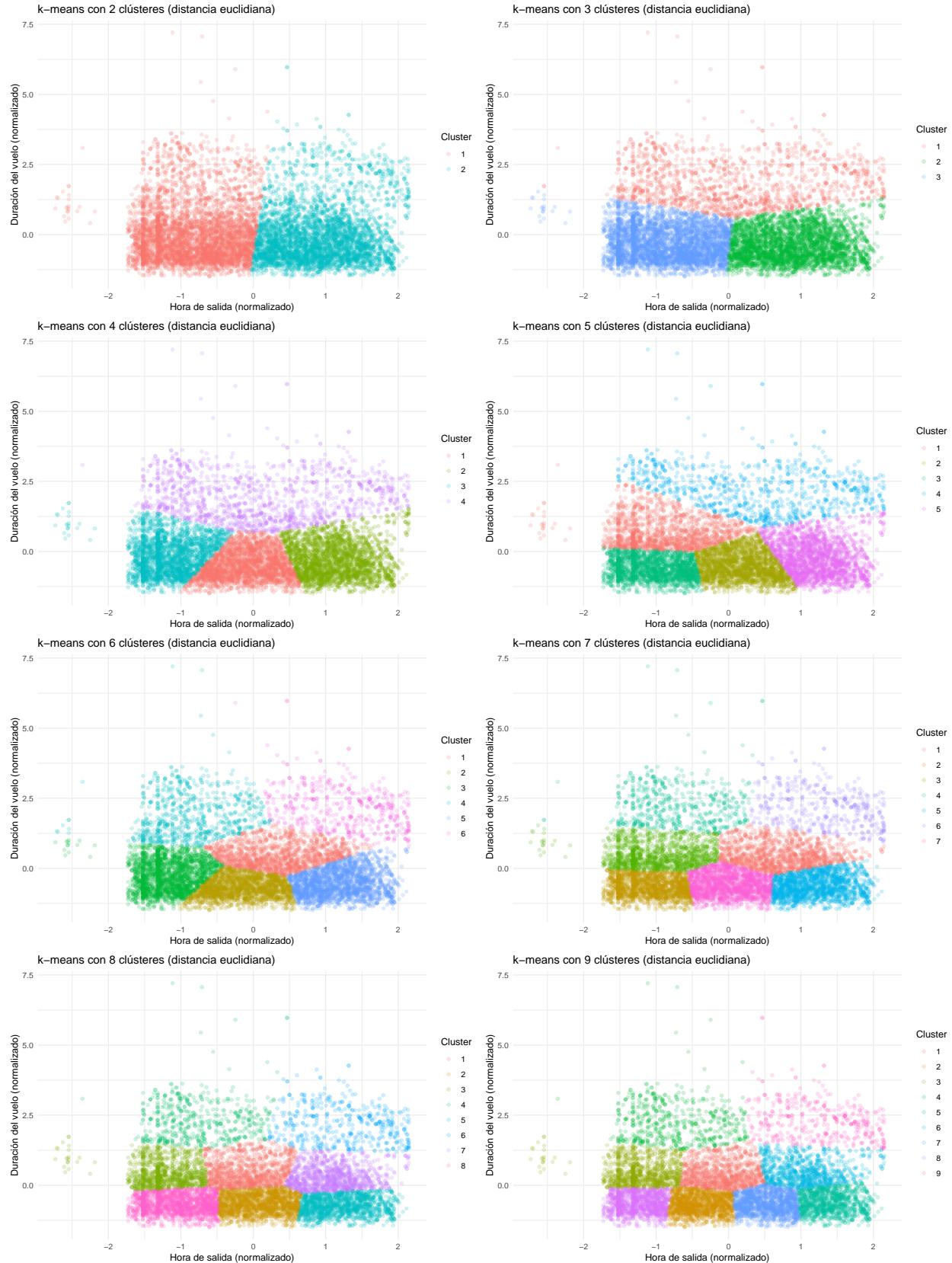
# Función para crear gráficos de k-means para un número dado de clústeres
create_kmeans_plot <- function(k) {
  # Establecemos una semilla para reproducibilidad
  set.seed(123)

  # Aplicamos el algoritmo k-means a los datos
  fit <- kmeans(scaled_data, centers = k)

  # Creamos un gráfico de dispersión donde cada clúster tiene un color diferente
  ggplot(scaled_data, aes(x = CRS_DEP_TIME_CONTINUOUS,
                          y = CRS_ELAPSED_TIME,
                          color = factor(fit$cluster))) +
    geom_point(alpha = 0.2) +
    labs(title = paste("k-means con", k, "clústeres (distancia euclíadiana)"),
         x = "Hora de salida (normalizado)",
         y = "Duración del vuelo (normalizado)",
         color = "Cluster") +
    theme_minimal()
}

# Generamos gráficos para diferentes números de clústeres
kmeans_plots <- lapply(2:9, create_kmeans_plot)

# Mostramos los gráficos por pantalla con 2 columnas
grid.arrange(grobs = kmeans_plots, ncol = 2)
```



Con $k = 2$, el algoritmo separa los vuelos del grupo más grande basándose principalmente en la hora de

salida. Con $k = 3$, emerge un tercer grupo que se enfoca en vuelos más largos.

A medida que incrementamos el valor de k (4-9), el algoritmo sigue fragmentando el grupo principal en segmentos menores, pero no logra aislar el grupo de vuelos nocturnos como un clúster independiente.

Fuente: <https://cran.r-project.org/web/packages/gridExtra/vignettes/arrangeGrob.html>

Se analizan, muestran y comentan las medidas de calidad del modelo generado.

Para evaluar la calidad del modelo, utilizamos la silueta media así como la SSW (*Sum of Squares Within*).

```
library(cluster)

# Inicializamos vectores vacíos para almacenar los resultados de silueta y SSW
kmeans_sil_results <- rep(0, 10)
kmeans_ssw_results <- rep(0, 10)

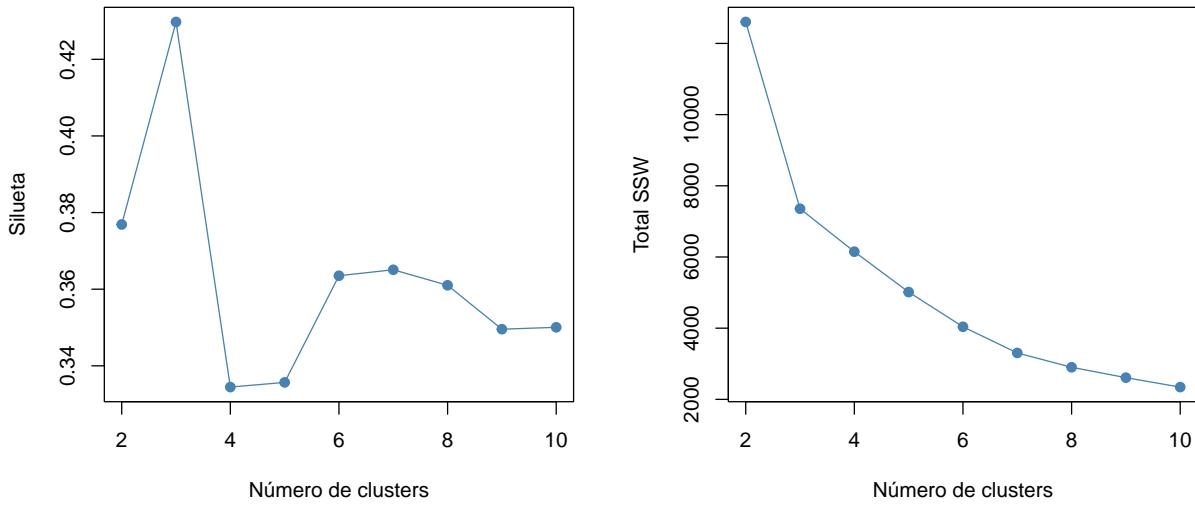
# Creamos una matriz de distancias para calcular la silueta
d <- daisy(scaled_data)

# Bucle para calcular silueta y SSW para diferentes números de clústeres (2-10)
for (k in 2:10) {
  set.seed(123) # Establecemos una semilla para reproducibilidad
  fit <- kmeans(scaled_data, k) # Aplicamos k-means
  y_cluster <- fit$cluster # Extraemos las asignaciones de clúster
  sk <- silhouette(y_cluster, d) # Calculamos los valores de silueta
  kmeans_sil_results[k] <- mean(sk[,3]) # Almacenamos la media de la silueta
  kmeans_ssw_results[k] <- fit$tot.withinss # Almacenamos el total de SSW
}

# Mostramos los dos gráficos un lado al otro
par(mfrow = c(1, 2))

# Creamos y mostramos el gráfico con los resultados de silueta
plot(2:10, kmeans_sil_results[2:10], type = "o", col = "steelblue", pch = 19,
      xlab = "Número de clusters", ylab = "Silueta")

# Creamos y mostramos el gráfico con los resultados de SSW
plot(2:10, kmeans_ssw_results[2:10], type = "o", col = "steelblue", pch = 19,
      xlab = "Número de clusters", ylab = "Total SSW")
```



El pico más alto de silueta se observa con $k = 3$ (0.43). En el gráfico de SSW (*Sum of Squares Within*), aunque no hay un codo claramente definido, la curva comienza a estabilizarse alrededor de $k = 3$.

Se comentan las conclusiones.

Si tuviéramos que decidir el mejor valor de k , sería 3. Esta elección se alinea con los resultados de la silueta y SSW, así como con lo que percibimos visualmente en los gráficos.

Aunque k -means ha sido eficaz en segmentar el bloque principal de vuelos según la hora de salida y la duración, no ha podido identificar con éxito el grupo más pequeño de vuelos nocturnos. Esto destaca una de las limitaciones de k -means: tiende a ser más efectivo con clústeres de tamaño similar.

Ejercicio 2

Se genera de nuevo el modelo no supervisado anterior, pero usando una métrica de distancia distinta.

A continuación, aplicamos el algoritmo k -medioms de la librería **flexclust**, que emplea la distancia Manhattan.

```
library(flexclust)

# Función para crear gráficos de k-medians para un número dado de clústeres
create_kmedioms_plot <- function(k) {
  # Establecemos una semilla para reproducibilidad
  set.seed(123)

  # Aplicamos el algoritmo k-medians a los datos
  fit <- flexclust::kcca(scaled_data, k = k, family = kccaFamily(which = "kmedioms"))
}
```

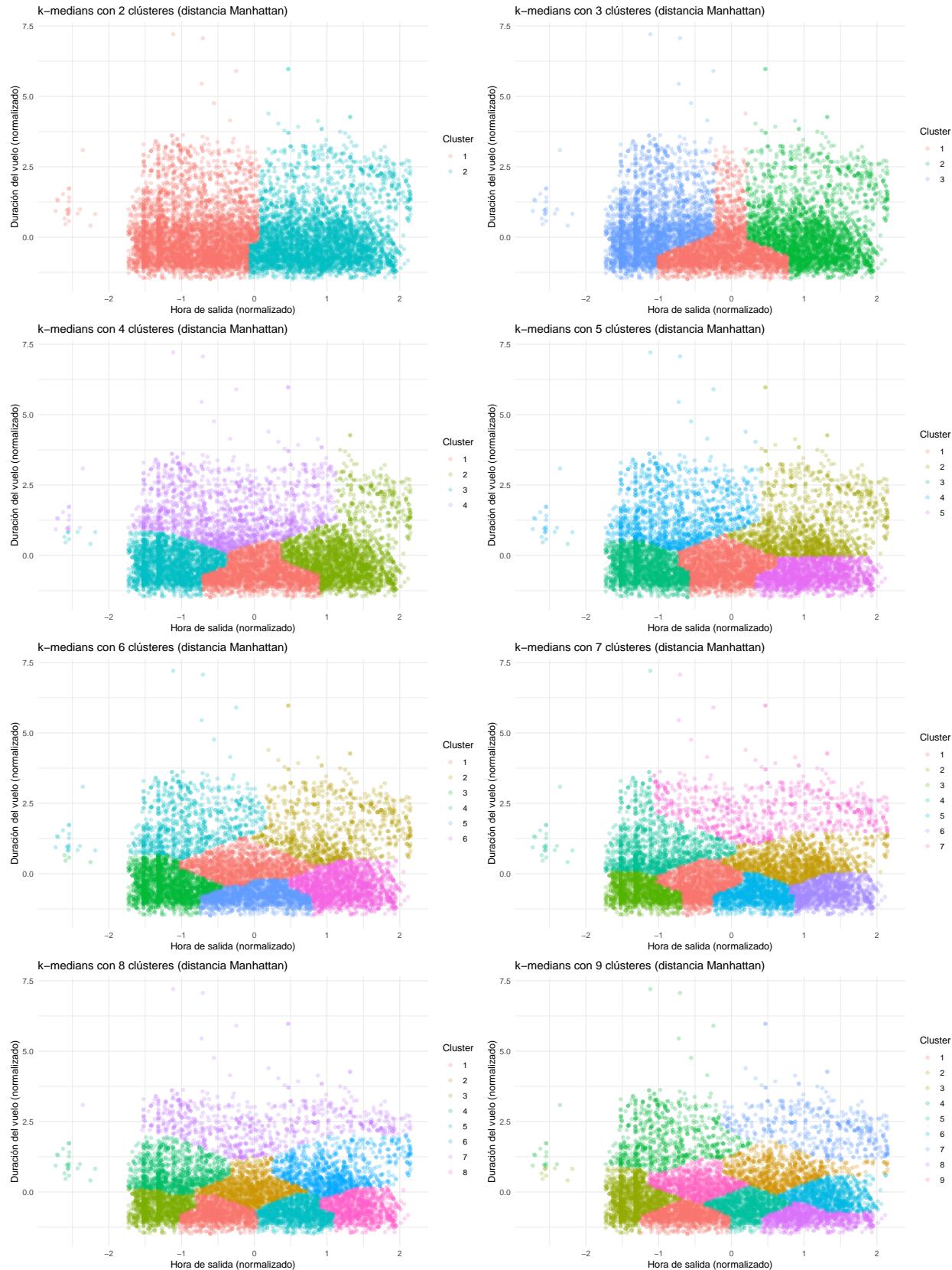
```

# Creamos un gráfico de dispersión donde cada clúster tiene un color diferente
ggplot(scaled_data, aes(x = CRS_DEP_TIME_CONTINUOUS,
                        y = CRS_ELAPSED_TIME,
                        color = factor(fit@cluster))) +
  geom_point(alpha = 0.25) +
  labs(title = paste("k-medians con", k, "clústeres (distancia Manhattan)"),
       x = "Hora de salida (normalizado)",
       y = "Duración del vuelo (normalizado)",
       color = "Cluster") +
  theme_minimal()
}

# Generamos gráficos para diferentes números de clústeres
kmedians_plots <- lapply(2:9, create_kmedians_plot)

# Mostramos los gráficos por pantalla con 2 columnas
grid.arrange(grobs = kmedians_plots, ncol = 2)

```



Al utilizar el algoritmo *k-medians* con la distancia Manhattan, observamos que los grupos se forman con

ángulos más marcados, en contraste con los clústeres más esféricos que se obtienen con *k-means*.

De manera similar a lo anterior, con $k = 2$, el algoritmo distingue los vuelos del grupo más grande según la hora de salida. Sin embargo, con $k = 3$, el clúster 1 adquiere una forma peculiar, asemejándose a un embudo invertido. Cuando incrementamos el valor de k hasta 9, el algoritmo continúa subdividiendo el grupo principal en segmentos más pequeños, pero aún así no consigue identificar el grupo de vuelos nocturnos como un clúster independiente.

Fuente: <https://stackoverflow.com/questions/7524042/how-to-specify-distance-metric-while-for-kmeans-in-r>

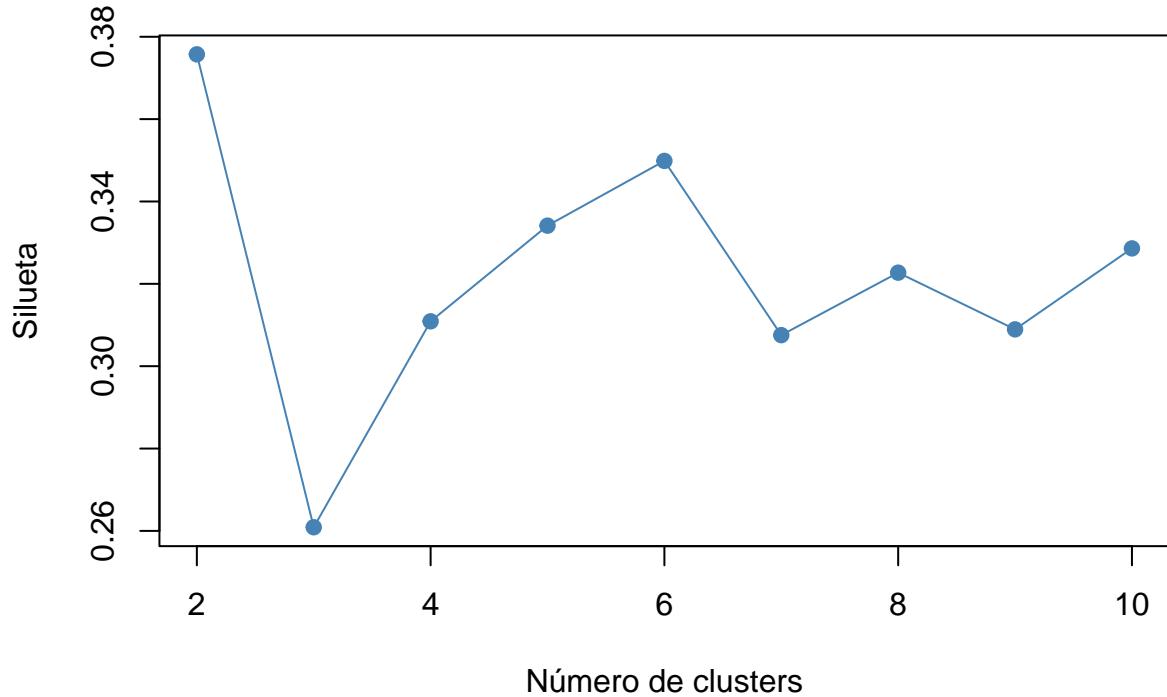
Se muestran y comentan las medidas de calidad del modelo generado.

La librería **flexclust** no cuenta con una función específica para calcular los SSW (*Sum of Squares Within*). Sin embargo, sí es posible calcular la silueta media.

```
kmedians_sil_results <- rep(0, 10)

for (k in 2:10) {
  set.seed(123)
  fit <- flexclust::kcca(scaled_data, k = k, family = kccaFamily(which = "kmedians"))
  sk <- silhouette(fit@cluster, d)
  kmedians_sil_results[k] <- mean(sk[,3])
}

plot(2:10, kmedians_sil_results[2:10], type="o", col="steelblue", pch=19,
      xlab="Número de clusters", ylab="Silueta")
```



En contraste con *k-means*, en este caso la silueta media más alta (0.38) se obtiene con $k = 2$. No obstante, estos dos grupos se dividen principalmente en la hora de salida. Si resulta relevante segmentar los vuelos en función de su duración, una buena alternativa podría ser $k = 6$, que tiene una silueta media de 0.35.

Se comparan los dos modelos no supervisados con métricas de distancia distintas.

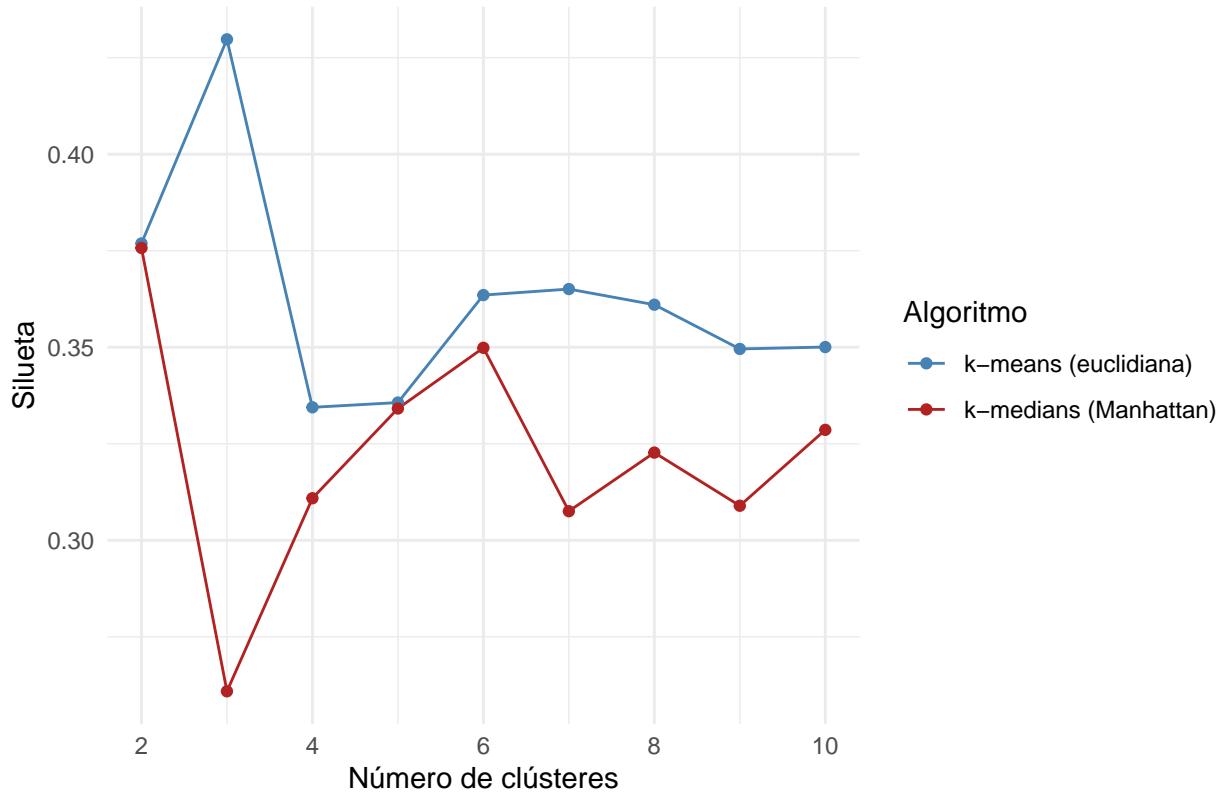
A continuación, creamos un gráfico para comparar las siluetas obtenidas con cada algoritmo.

```
library(ggplot2)

data <- data.frame(
  Cluster = rep(2:10, 2),
  Silhouette = c(kmeans_sil_results[2:10], kmedians_sil_results[2:10]),
  Algoritmo = rep(c("k-means (euclíadiana)", "k-medians (Manhattan)"), each = 9)
)

ggplot(data, aes(x = Cluster, y = Silhouette, color = Algoritmo, group = Algoritmo)) +
  geom_line() +
  geom_point() +
  labs(x = "Número de clústeres", y = "Silueta", title = "Comparación de silueta media") +
  theme_minimal() +
  scale_color_manual(values = c("steelblue", "firebrick"))
```

Comparación de silueta media



Podemos apreciar que los algoritmos *k-means* y *k-medians* muestran un desempeño similar cuando $k = 2$ y $k = 5$. No obstante, el algoritmo *k-means* presenta un mejor rendimiento en general, destacando especialmente con $k = 3$.

En términos globales, el valor más alto alcanzado fue 0.43, utilizando *k-means* con $k = 3$. De manera sorprendente, el algoritmo *k-medians* experimentó su peor rendimiento con este mismo valor de k , alcanzando solo 0.26.

Se comentan las conclusiones.

En los gráficos, es evidente que los dos algoritmos generan formas de clústeres bastante distintas. *k-means*, al utilizar la distancia euclídea, tiende a formar clústeres más esféricos. Por otro lado, *k-medians*, que se basa en la distancia de Manhattan, da lugar a clústeres con formas más angulares.

A pesar de que *k-means* ha logrado una silueta media más elevada en comparación con *k-medians*, ninguno de los dos algoritmos ha podido identificar el grupo más pequeño como un clúster independiente. Aunque *k-medians* gestiona mejor los valores atípicos, ambos algoritmos comparten la misma limitación: enfrentan dificultades cuando los clústeres varían significativamente en tamaño.

Ejercicio 3

Se aplican los algoritmos DBSCAN y OPTICS de forma correcta.

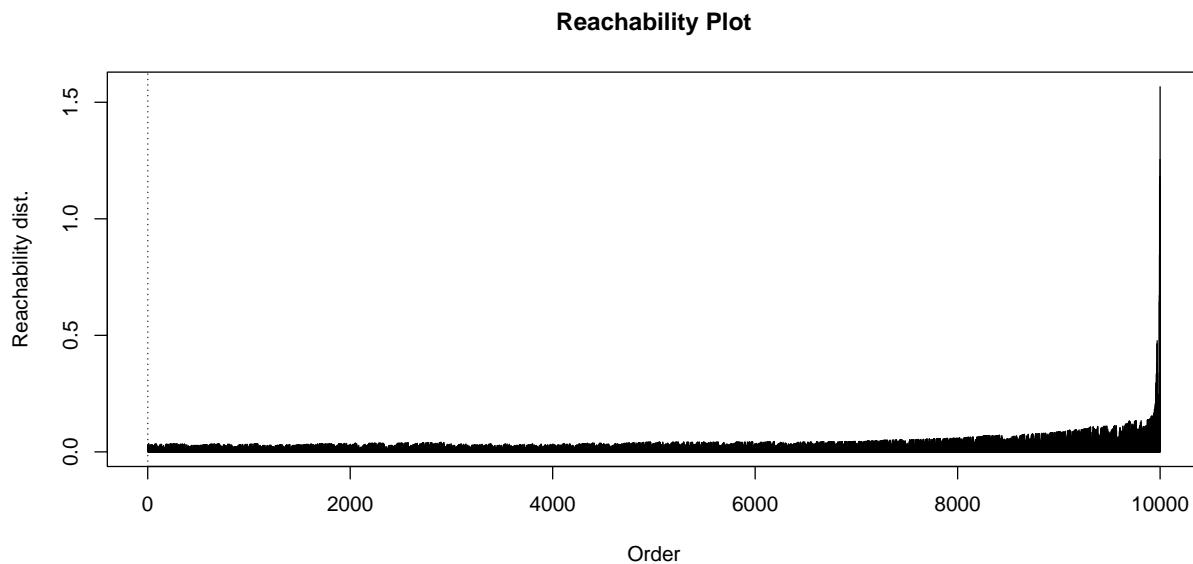
Para comenzar, cargamos la librería `dbscan` y aplicamos `optics()` con sus parámetros predeterminados (`eps = NULL, minPts = 5`).

```
library(dbscan)
optics_minPts5 <- optics(scaled_data)
print(optics_minPts5)
```

```
## OPTICS ordering/clustering for 10000 objects.
## Parameters: minPts = 5, eps = 2.00761620370805, eps_cl = NA, xi = NA
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                     eps_cl, xi
```

A continuación, visualizamos el diagrama de alcanzabilidad (*reachability plot*).

```
plot(optics_minPts5)
```



Al observar el extremo derecho del gráfico (cerca del punto 10000), notamos un pequeño espacio entre dos líneas verticales. La línea de menor altura representa una *reachability distance* aproximada de 0.5. Por tanto, optamos por establecer un umbral ligeramente inferior, de 0.4.

A continuación, utilizamos `extractDBSCAN()` con un umbral de `eps_cl = 0.4` para identificar los clústeres.

```
minPts5_eps0_4 <- extractDBSCAN(optics_minPts5, eps_cl = 0.4)
minPts5_eps0_4
```

```
## OPTICS ordering/clustering for 10000 objects.
## Parameters: minPts = 5, eps = 2.00761620370805, eps_cl = 0.4, xi = NA
## The clustering contains 2 cluster(s) and 11 noise points.
```

```

## 
##   0   1   2
## 11 9968  21
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, cluster

```

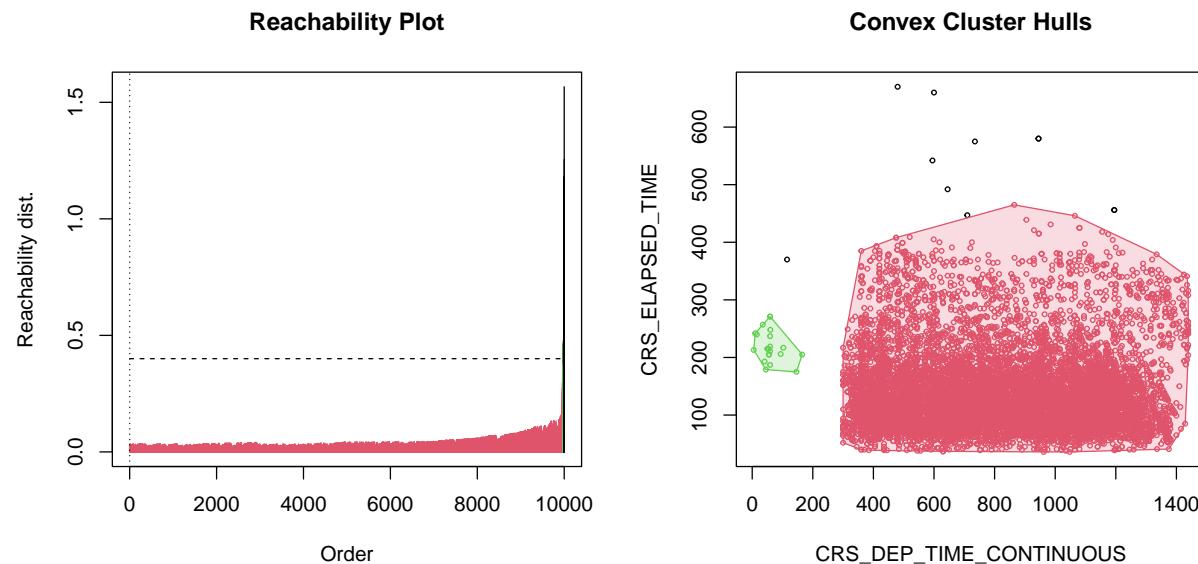
Vemos que se han formado dos clústeres: uno muy grande (9968 puntos) y otro muy pequeño (21 puntos). Los 11 puntos restantes se clasifican como ruido.

Podemos examinar estos clústeres mediante el *reachability plot*, así como con el *hullplot*.

```

par(mfrow=c(1, 2))
plot(minPts5_eps0_4)
hullplot(sampled_data, minPts5_eps0_4)

```



Observamos que DBSCAN ha logrado diferenciar con bastante precisión los dos clústeres. Sin embargo, los vuelos más largos, aquellos que van desde el territorio continental de EE.UU. a Hawái, se han clasificado como ruido.

Se prueban, describen e interpretan los resultados con diferentes valores de `eps` y `minPts`.

Primero, incrementamos `eps_cl` ligeramente, de 0.4 a 0.47.

```

minPts5_eps0_47 <- extractDBSCAN(optics_minPts5, eps_cl = 0.47)
minPts5_eps0_47

```

```

## OPTICS ordering/clustering for 10000 objects.
## Parameters: minPts = 5, eps = 2.00761620370805, eps_cl = 0.47, xi = NA
## The clustering contains 2 cluster(s) and 9 noise points.
##

```

```

##      0     1     2
##    9 9970   21
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, cluster

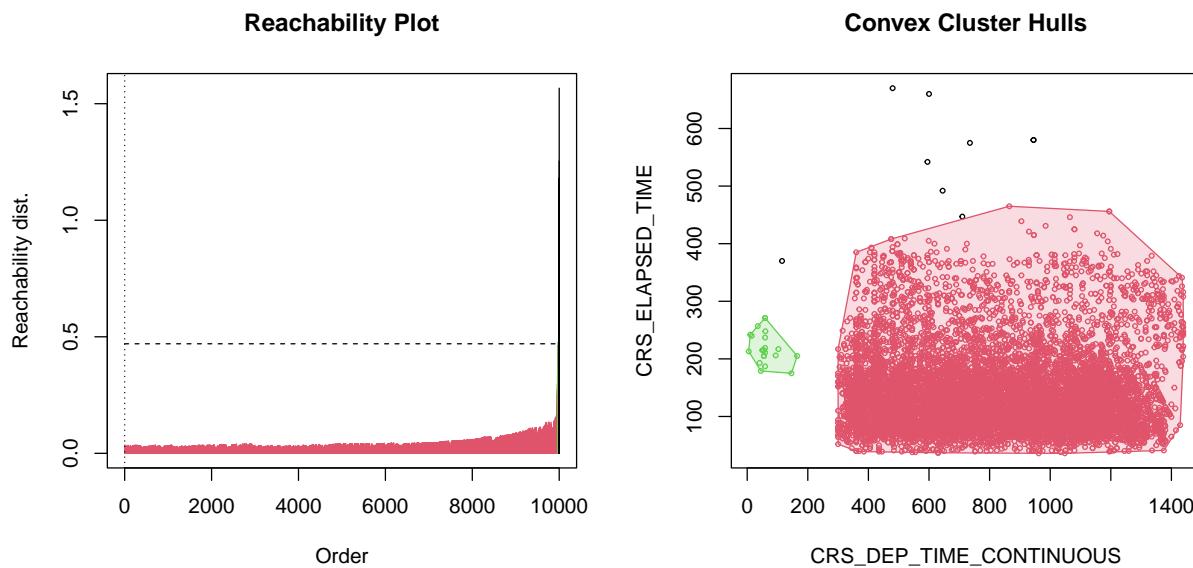
```

Con este ajuste, la cantidad de valores atípicos se reduce de 11 a 9. Estos 2 puntos adicionales se han integrado en el clúster 1 (color rojo):

```

par(mfrow=c(1, 2))
plot(minPts5_eps0_47)
hullplot(sampled_data, minPts5_eps0_47)

```



No obstante, al aumentar `eps_cl` más, hasta 0.5, perdemos el segundo clúster, quedando solamente uno de gran tamaño:

```

minPts5_eps0_5 <- extractDBSCAN(optics_minPts5, eps_cl = 0.5)
minPts5_eps0_5

```

```

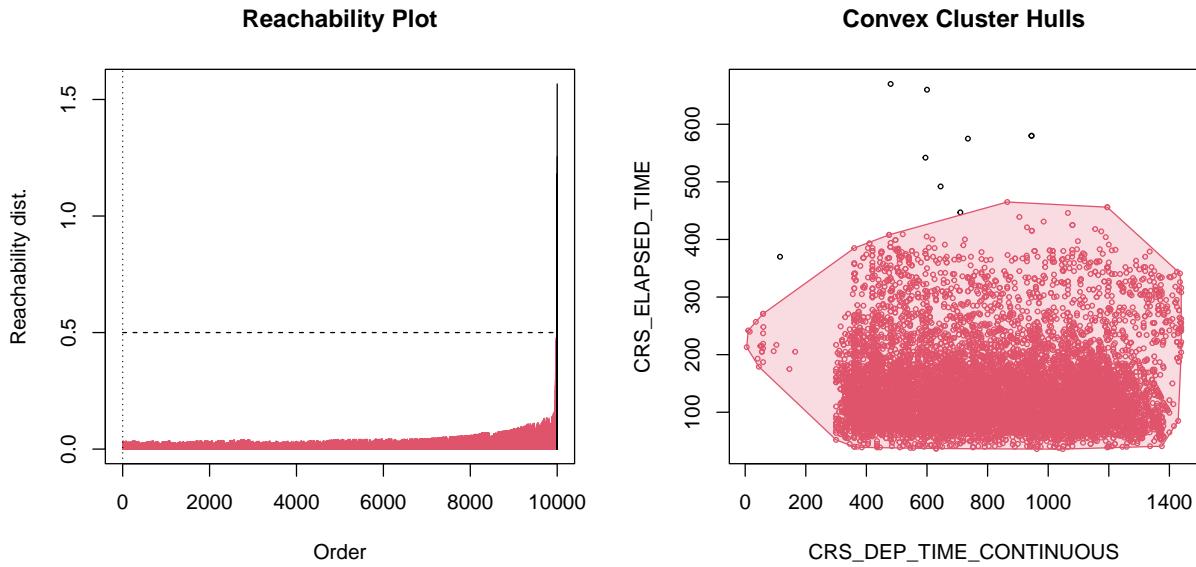
## OPTICS ordering/clustering for 10000 objects.
## Parameters: minPts = 5, eps = 2.00761620370805, eps_cl = 0.5, xi = NA
## The clustering contains 1 cluster(s) and 9 noise points.
##
##      0     1
##    9 9991
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, cluster

```

```

par(mfrow=c(1, 2))
plot(minPts5_eps0_5)
hullplot(sampled_data, minPts5_eps0_5)

```



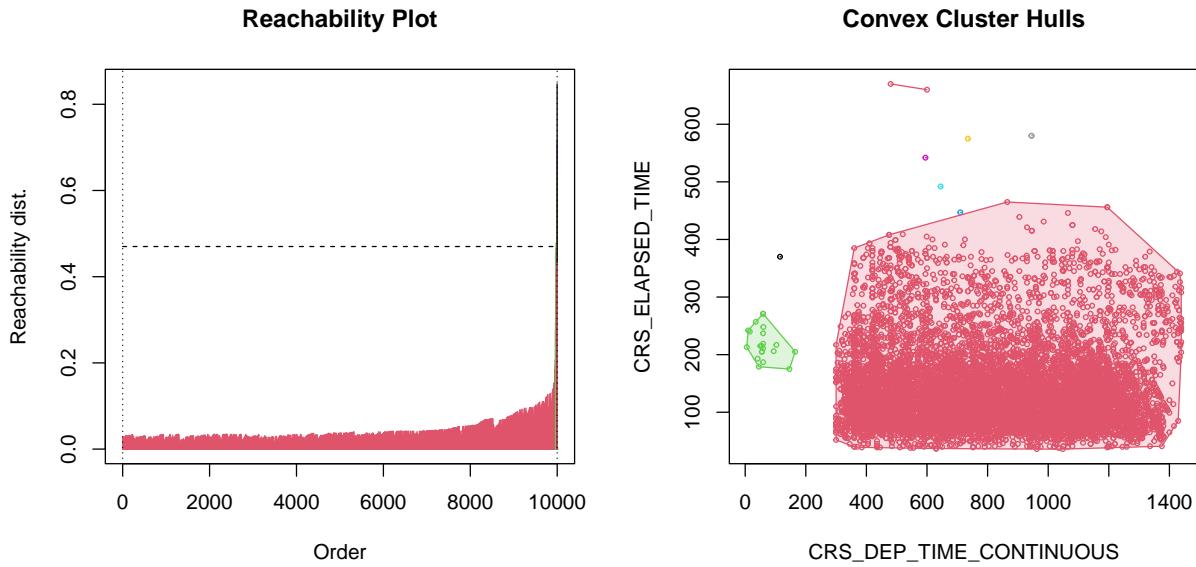
Si nuestro objetivo es asignar cada punto a un clúster, sin clasificar ningún punto como ruido, podemos reducir `minPts` a 1.

```
optics_minPts1 <- optics(scaled_data, minPts = 1)
minPts1_eps0_47 <- extractDBSCAN(optics_minPts1, eps_cl = 0.47)
minPts1_eps0_47
```

```
## OPTICS ordering/clustering for 10000 objects.
## Parameters: minPts = 1, eps = 0.84718851609248, eps_cl = 0.47, xi = NA
## The clustering contains 9 cluster(s) and 0 noise points.
##
##      1   2   3   4   5   6   7   8   9
## 9970   21    1    1    1    1    2    1    2
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, cluster
```

De este modo, obtenemos un total de 9 clústeres:

```
par(mfrow=c(1, 2))
plot(minPts1_eps0_47)
hullplot(sampled_data, minPts1_eps0_47)
```



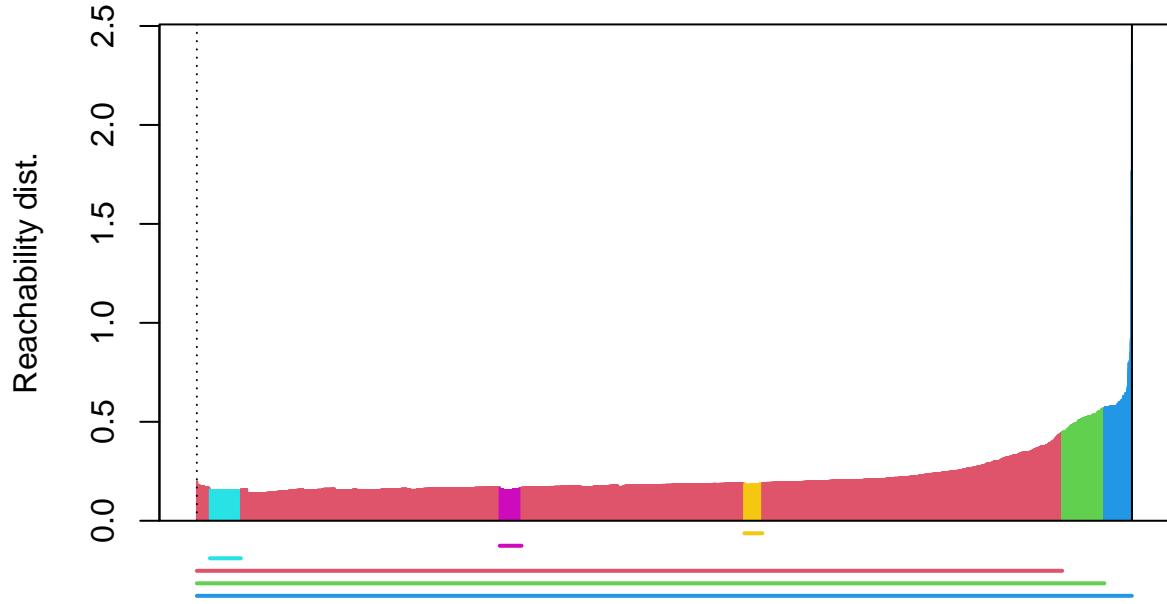
Para detectar áreas de mayor densidad dentro de un clúster, empleamos `extractXi()`. Para prevenir la formación de clústeres demasiado pequeños, incrementamos `minPts` a 150. Fijamos `xi` (el umbral de inclinación) en 0.007.

```
optics_minPts150 <- optics(scaled_data, minPts=150)
minPts150_xi007 <- extractXi(optics_minPts150, xi = 0.007)
minPts150_xi007
```

```
## OPTICS ordering/clustering for 10000 objects.
## Parameters: minPts = 150, eps = 4.42929010083714, eps_cl = NA, xi = 0.007
## The clustering contains 6 cluster(s) and 1 noise points.
##
## Available fields: order, reachdist, coredist, predecessor, minPts, eps,
##                   eps_cl, xi, clusters_xi, cluster
```

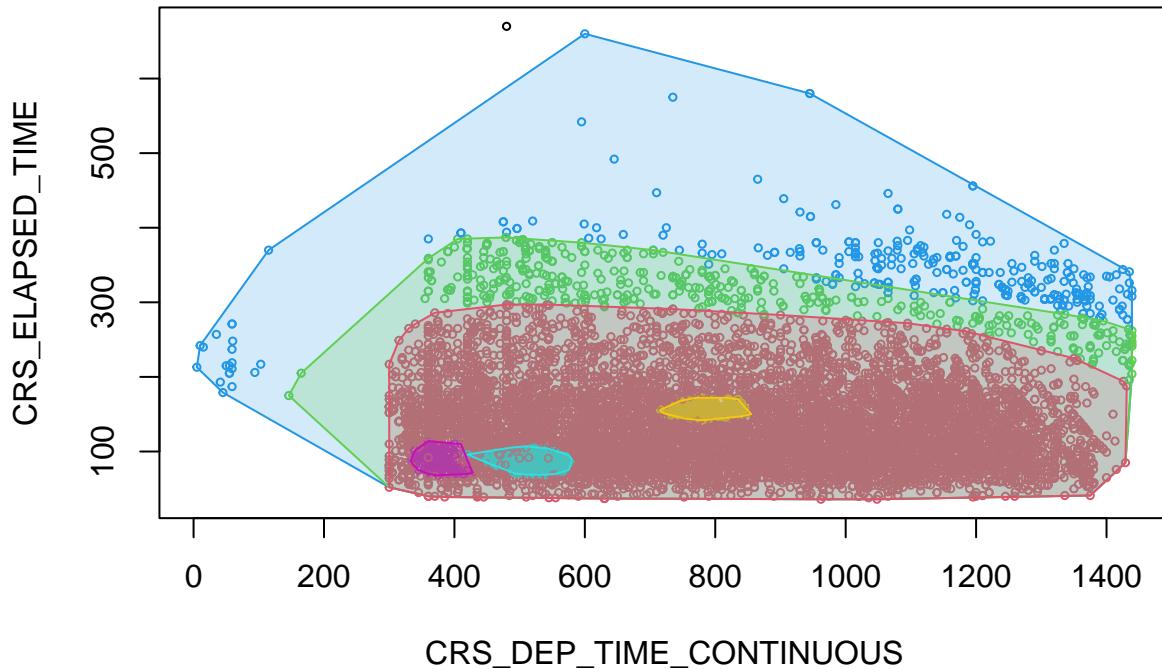
```
plot(minPts150_xi007)
```

Reachability Plot



```
hullplot(sampled_data, minPts150_xi007)
```

Convex Cluster Hulls



El resultado es similar a una muñeca rusa. El clúster más grande (en azul oscuro) incluye casi todos los vuelos del conjunto, a excepción de uno. Dentro de este, el algoritmo ha identificado una zona de mayor densidad (el clúster verde), excluyendo los vuelos nocturnos y aquellos de muy larga duración. El clúster rojo, aún más denso, incluye solo los vuelos de menos de 300 minutos (aproximadamente 5 horas).

Finalmente, hay una concentración muy alta de vuelos cortos entre las 5:00 y las 10:00 de la mañana (en colores lila y azul claro), y otra de vuelos de duración intermedia entre las 12:00 y las 14:00 aproximadamente (en amarillo).

Se obtiene una medida de lo bueno que es el agrupamiento.

Para evaluar la calidad de agrupamiento, calculamos la silueta media del segundo modelo (`minPts = 5, eps_cl = 0.47`).

```
sk <- silhouette(minPts5_eps0_47$cluster, d)
sil_values <- sk[, 3]

# Filtramos y mantenemos sólo los valores positivos de la silueta.
# En este contexto, los valores negativos corresponden a puntos que OPTICS asignó al clúster "0",
# lo que indica que no pertenecen a ningún clúster y son tratados como ruido
positive_sil_values <- sil_values[sil_values > 0]

# Calculamos la media de los valores de silueta positivos para considerar
# solo los puntos que pertenecen a algún cluster
optics_sil_result <- mean(positive_sil_values)
```

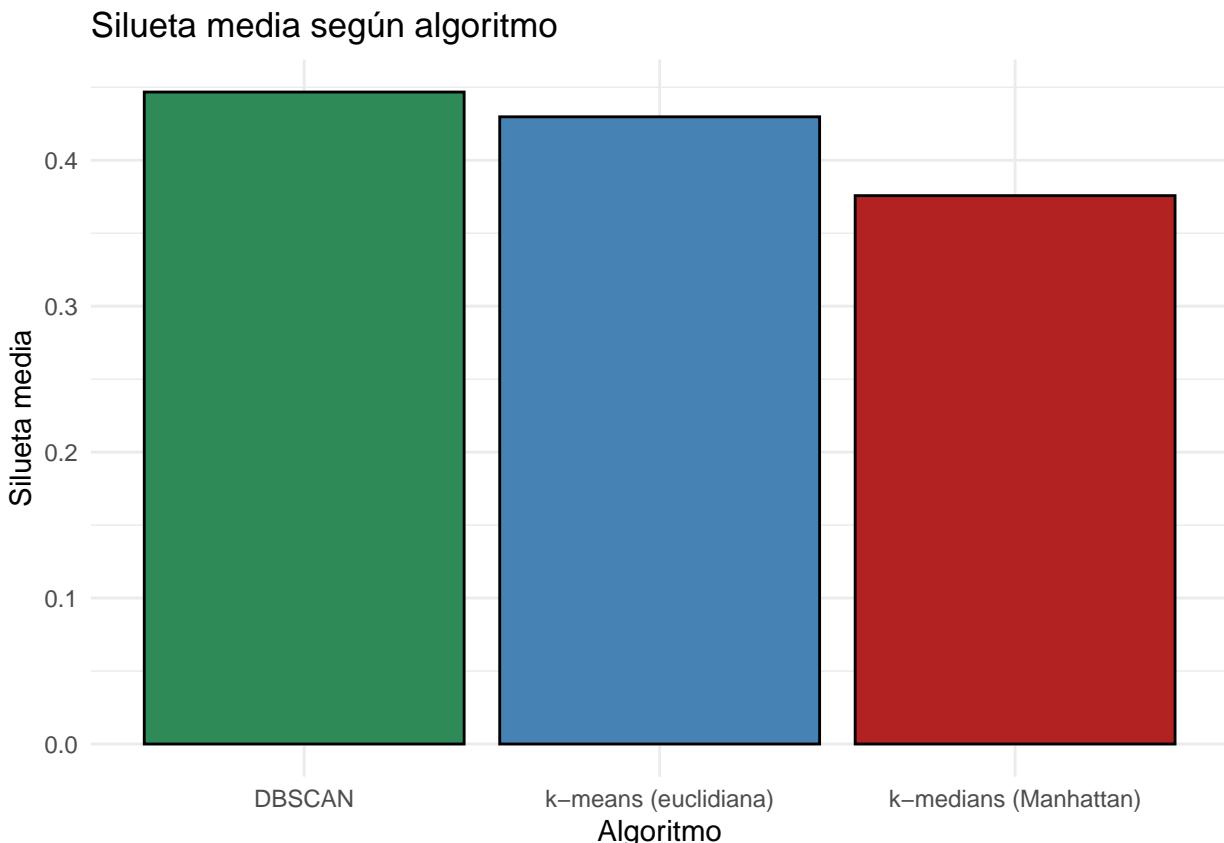
```
# Devolvemos la silueta media  
optics_sil_result
```

```
## [1] 0.446716
```

Se comparan los resultados obtenidos de los modelos anteriores y DBSCAN.

Para esta comparación, seleccionamos la mayor silueta media obtenida de los algoritmos *k-means* y *k-medians* (con $k = 3$ y $k = 2$, respectivamente).

```
sil_results <- data.frame(  
  Algoritmo = c("k-means (eucladiana)", "k-medians (Manhattan)", "DBSCAN"),  
  Silueta = c(max(kmeans_sil_results), max(kmedians_sil_results), optics_sil_result)  
)  
  
ggplot(sil_results, aes(x = Algoritmo, y = Silueta, fill = Algoritmo)) +  
  geom_bar(stat = "identity", color="black") +  
  scale_fill_manual(values = c("seagreen", "steelblue", "firebrick")) +  
  theme_minimal() +  
  theme(legend.position = "none") +  
  labs(title = "Silueta media según algoritmo",  
       x = "Algoritmo",  
       y = "Silueta media")
```



Observamos que la silueta media obtenida con *DBSCAN* (0.45) ha superado ligeramente a la obtenida con *k-means* (0.43), lo que indica que *DBSCAN* ajusta mejor a los datos.

Se comentan las conclusiones.

Mientras *k-means* y *k-medians* no lograron identificar el grupo de vuelos nocturnos como un clúster independiente, *DBSCAN* lo hizo fácilmente, incluso con su valor por defecto de `minPts`. Aunque fue necesario elegir el valor adecuado para `eps_cl`, este proceso resultó bastante intuitivo al apoyarnos en el *reachability plot* para identificar los «valles», que corresponden a los clústeres.

Este análisis destaca que los algoritmos *k-means* y *k-medians* son efectivos para clústeres de tamaños similares. Sin embargo, *DBSCAN* emerge como la mejor opción para clústeres de tamaños desiguales y para identificar zonas de mayor densidad dentro de un mismo clúster.

Ejercicio 4

Se seleccionan las muestras de entrenamiento y test.

Para este análisis, utilizaremos las siguientes variables explicativas:

- `DAY_OF_WEEK`: Día de la semana en que se realiza el vuelo.
- `OP_UNIQUE_CARRIER`: Código de identificación único para la aerolínea operadora.
- `ORIGIN`: Código del aeropuerto de origen.
- `DEST`: Código del aeropuerto de destino.
- `CRS_DEP_TIME_CONTINUOUS`: Hora de salida programada, expresada en minutos desde medianoche.
- `CRS_ARR_TIME_CONTINUOUS`: Hora de llegada programada, expresada en minutos desde medianoche.
- `CRS_ELAPSED_TIME`: Duración programada del vuelo en minutos.

Y la siguiente variable dependiente:

- `ARR_DEL15`: Indicador de si el vuelo llegó con más de 15 minutos de retraso (1 = sí, 0 = no).

Para comenzar, seleccionamos únicamente las columnas que se utilizarán en el modelo.

```
library(dplyr)
flightData_reduced <- select(flightData, DAY_OF_WEEK, OP_UNIQUE_CARRIER, ORIGIN, DEST,
                               CRS_DEP_TIME_CONTINUOUS, CRS_ARR_TIME_CONTINUOUS,
                               CRS_ELAPSED_TIME, ARR_DEL15)
str(flightData_reduced)
```

```
## 'data.frame':      560887 obs. of  8 variables:
##   $ DAY_OF_WEEK       : Factor w/ 7 levels "domingo","jueves",...: 3 3 3 3 3 3 3 3 3 ...
##   $ OP_UNIQUE_CARRIER : Factor w/ 15 levels "Alaska Airlines",...: 5 5 5 5 5 5 5 5 5 ...
##   $ ORIGIN            : Factor w/ 340 levels "ABE","ABI","ABQ",...: 1 1 1 5 5 12 12 13 15 15 ...
##   $ DEST               : Factor w/ 340 levels "ABE","ABI","ABQ",...: 21 21 21 21 21 21 21 21 21 99 ...
##   $ CRS_DEP_TIME_CONTINUOUS: int  414 762 1047 455 845 370 980 615 765 426 ...
##   $ CRS_ARR_TIME_CONTINUOUS: int  540 890 1178 528 912 539 1146 683 868 483 ...
##   $ CRS_ELAPSED_TIME     : int  126 128 131 73 67 109 106 68 103 57 ...
##   $ ARR_DEL15           : int  0 0 0 0 0 0 0 0 0 ...
```

A continuación, transformamos la variable dependiente en un factor y eliminamos los acentos de los días de la semana (como «miércoles», «sábado»), para evitar errores en su procesamiento.

```
flightData_reduced$ARR_DEL15 <- factor(flightData_reduced$ARR_DEL15)

flightData_reduced$DAY_OF_WEEK <- factor(
  flightData_reduced$DAY_OF_WEEK,
  levels=c("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo"),
  labels=c("lunes", "martes", "miercoles", "jueves", "viernes", "sabado", "domingo"))

flightData_reduced$ARR_DEL15 <- factor(
  flightData_reduced$ARR_DEL15,
  levels=c(0, 1),
  labels=c("a tiempo", "retrasado"))

str(flightData_reduced)

## 'data.frame': 560887 obs. of 8 variables:
## $ DAY_OF_WEEK : Factor w/ 7 levels "lunes","martes",...: 1 1 1 1 1 1 1 1 1 ...
## $ OP_UNIQUE_CARRIER : Factor w/ 15 levels "Alaska Airlines",...: 5 5 5 5 5 5 5 5 5 ...
## $ ORIGIN : Factor w/ 340 levels "ABE","ABI","ABQ",...: 1 1 1 5 5 12 12 13 15 15 ...
## $ DEST : Factor w/ 340 levels "ABE","ABI","ABQ",...: 21 21 21 21 21 21 21 21 21 99 ...
## $ CRS_DEP_TIME_CONTINUOUS: int 414 762 1047 455 845 370 980 615 765 426 ...
## $ CRS_ARR_TIME_CONTINUOUS: int 540 890 1178 528 912 539 1146 683 868 483 ...
## $ CRS_ELAPSED_TIME : int 126 128 131 73 67 109 106 68 103 57 ...
## $ ARR_DEL15 : Factor w/ 2 levels "a tiempo","retrasado": 1 1 1 1 1 1 1 1 1 ...
```

Después, dividimos el conjunto de datos en dos partes: un conjunto de entrenamiento (**train**) y otro de prueba (**test**). El conjunto de entrenamiento se utiliza para construir un primer modelo, mientras que el conjunto de prueba sirve para evaluar la calidad del modelo. Optamos por una proporción de 4/5 para entrenamiento y 1/5 para prueba.

```
set.seed(123)
split_prop <- 5
indexes <- sample(1:nrow(flightData_reduced), size=floor(((split_prop-1) / split_prop) * nrow(flightData_reduced)))
train <- flightData_reduced[indexes,]
test <- flightData_reduced[-indexes,]
```

X contiene todas las variables independientes, que son 7 en total. **y** representa nuestra variable dependiente, **ARR_DEL15**.

```
trainX <- select(train, -ARR_DEL15)
trainy <- train$ARR_DEL15
testX <- select(test, -ARR_DEL15)
testy <- test$ARR_DEL15
```

Verificamos la estructura de cada subconjunto.

```
str(trainX)
```

```
## 'data.frame': 448709 obs. of 7 variables:
## $ DAY_OF_WEEK : Factor w/ 7 levels "lunes","martes",...: 3 2 2 4 5 3 7 5 3 4 ...
```

```

## $ OP_UNIQUE_CARRIER      : Factor w/ 15 levels "Alaska Airlines",...: 3 5 14 3 3 4 15 11 15 12 ...
## $ ORIGIN                 : Factor w/ 340 levels "ABE","ABI","ABQ",...: 23 190 118 68 92 54 201 88 263 ...
## $ DEST                   : Factor w/ 340 levels "ABE","ABI","ABQ",...: 70 68 270 92 280 223 110 325 ...
## $ CRS_DEP_TIME_CONTINUOUS: int  475 539 1004 420 812 405 1035 600 360 304 ...
## $ CRS_ARR_TIME_CONTINUOUS: int  702 652 1141 539 873 483 1205 724 470 521 ...
## $ CRS_ELAPSED_TIME       : int  167 113 137 179 181 138 170 124 170 157 ...

```

```
str(testX)
```

```

## 'data.frame':   112178 obs. of  7 variables:
## $ DAY_OF_WEEK          : Factor w/ 7 levels "lunes","martes",...: 1 1 1 1 1 1 1 1 1 1 ...
## $ OP_UNIQUE_CARRIER     : Factor w/ 15 levels "Alaska Airlines",...: 5 5 5 5 5 5 5 5 5 5 ...
## $ ORIGIN                : Factor w/ 340 levels "ABE","ABI","ABQ",...: 1 1 5 12 21 21 21 21 21 21 ...
## $ DEST                  : Factor w/ 340 levels "ABE","ABI","ABQ",...: 21 21 21 21 1 1 5 13 80 93 ...
## $ CRS_DEP_TIME_CONTINUOUS: int  414 762 845 980 590 872 737 505 1268 550 ...
## $ CRS_ARR_TIME_CONTINUOUS: int  540 890 912 1146 716 1001 796 565 1359 552 ...
## $ CRS_ELAPSED_TIME      : int  126 128 67 106 126 129 59 60 91 62 ...

```

```
summary(trainy)
```

```

## a tiempo retrasado
##    364926     83783

```

```
summary(testy)
```

```

## a tiempo retrasado
##    91262     20916

```

Finalmente, nos aseguramos de que la proporción de vuelos con retrasos sea aproximadamente igual en ambos subconjuntos.

```
prop.table(table((train$ARR_DEL15)))
```

```

##
## a tiempo retrasado
## 0.8132799 0.1867201

```

```
prop.table(table((test$ARR_DEL15)))
```

```

##
## a tiempo retrasado
## 0.8135463 0.1864537

```

Se justifican las proporciones seleccionadas.

Dada la cantidad de observaciones en nuestro conjunto de datos (560887 vuelos), contamos con una mayor flexibilidad en cuanto a la elección de proporciones. No obstante, es importante reconocer que cada decisión conlleva sus propias ventajas y desventajas.

Si optamos por una menor cantidad de datos para el entrenamiento, deberíamos esperar una mayor variabilidad en las estimaciones de los parámetros del modelo (las reglas del árbol de decisión). Por otro lado, si reducimos los datos destinados al test, la precisión de nuestras medidas de calidad en la clasificación también presentará una variabilidad mayor.

Para este estudio, hemos elegido una proporción de 80 % para el entrenamiento y un 20 % para el test, basándonos en el principio de Pareto.

Fuente 1: <https://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validation> Fuente 2: https://es.wikipedia.org/wiki/Principio_de_Pareto

Ejercicio 5

Se generan reglas y se seleccionan e interpretan las más significativas.

A continuación, creamos un árbol de decisión con C5.0.

```
library(C50)
C50_model <- C5.0(trainX, trainy, rules=TRUE)
summary(C50_model)

##
## Call:
## C5.0.default(x = trainX, y = trainy, rules = TRUE)
##
## 
## C5.0 [Release 2.07 GPL Edition]      Fri Feb  9 20:51:00 2024
## -----
##
## Class specified by attribute 'outcome'
##
## Read 448709 cases (8 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (168995/19191, lift 1.1)
##   CRS_DEP_TIME_CONTINUOUS <= 674
##   -> class a tiempo  [0.886]
##
## Rule 2: (7331/888, lift 1.1)
##   DEST = SLC
##   -> class a tiempo  [0.879]
##
## Rule 3: (283900/40641, lift 1.1)
##   CRS_ARR_TIME_CONTINUOUS <= 1044
##   -> class a tiempo  [0.857]
##
## Rule 4: (182897/27563, lift 1.0)
##   DAY_OF_WEEK in {martes, miercoles, sabado}
##   -> class a tiempo  [0.849]
```

```

##  

## Rule 5: (1493/242, lift 1.0)  

##   DAY_OF_WEEK = jueves  

##   DEST = LGA  

##   -> class a tiempo [0.837]  

##  

## Rule 6: (38196/6512, lift 1.0)  

##   ORIGIN in {BUF, CHS, DCA, EWR, ORH, PVD, SFO, SLC}  

##   -> class a tiempo [0.829]  

##  

## Rule 7: (432216/77680, lift 1.0)  

##   OP_UNIQUE_CARRIER in {Alaska Airlines, Allegiant Air, American Airlines,  

##                         Delta Air Lines, Endeavor Air, Envoy Air,  

##                         Frontier Airlines, Hawaiian Airlines,  

##                         PSA Airlines, Republic Airways, SkyWest Airlines,  

##                         Southwest Airlines, Spirit Airlines,  

##                         United Airlines}  

##   -> class a tiempo [0.820]  

##  

## Rule 8: (13/2, lift 4.3)  

##   DAY_OF_WEEK in {lunes, jueves, viernes, domingo}  

##   OP_UNIQUE_CARRIER = JetBlue Airways  

##   ORIGIN = MCO  

##   DEST = SLC  

##   -> class retrasado [0.800]  

##  

## Rule 9: (41/10, lift 4.0)  

##   OP_UNIQUE_CARRIER = JetBlue Airways  

##   DEST = MIA  

##   CRS_DEP_TIME_CONTINUOUS > 839  

##   CRS_ELAPSED_TIME <= 205  

##   -> class retrasado [0.744]  

##  

## Rule 10: (24/7, lift 3.7)  

##   DAY_OF_WEEK in {lunes, jueves, viernes, domingo}  

##   OP_UNIQUE_CARRIER = JetBlue Airways  

##   DEST = HPN  

##   CRS_DEP_TIME_CONTINUOUS > 839  

##   CRS_ARR_TIME_CONTINUOUS <= 1191  

##   -> class retrasado [0.692]  

##  

## Rule 11: (125/45, lift 3.4)  

##   DAY_OF_WEEK in {lunes, domingo}  

##   OP_UNIQUE_CARRIER = JetBlue Airways  

##   DEST = LGA  

##   CRS_DEP_TIME_CONTINUOUS > 839  

##   -> class retrasado [0.638]  

##  

## Rule 12: (3247/1338, lift 3.1)  

##   DAY_OF_WEEK in {lunes, jueves, viernes, domingo}  

##   OP_UNIQUE_CARRIER = JetBlue Airways  

##   DEST in {ABQ, ACK, ATL, AUS, BDL, BNA, BOS, BUF, CLE, CLT, DTW, EWR,  

##            JAX, JFK, LAS, LAX, MCO, MKE, MSP, MSY, ORD, PBI, PDX, PHL,  

##            PHX, PIT, PSE, RDU, RIC, ROC, RSW, SAV, SEA, SJC, SJU, SMF,

```

```

##           STT, TPA}
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.588]
##
## Rule 13: (224/93, lift 3.1)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## CRS_DEP_TIME_CONTINUOUS > 674
## CRS_DEP_TIME_CONTINUOUS <= 839
## CRS_ARR_TIME_CONTINUOUS > 1045
## -> class retrasado [0.584]
##
## Rule 14: (6023/2863, lift 2.8)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## CRS_DEP_TIME_CONTINUOUS > 674
## -> class retrasado [0.525]
##
## Default class: a tiempo
##
##
## Evaluation on training data (448709 cases):
##
##          Rules
##          -----
##          No      Errors
##
##          14 83141(18.5%)    <<
##
##          (a)      (b)    <-classified as
##          ----  -----
##          363130    1796    (a): class a tiempo
##          81345     2438    (b): class retrasado
##
##          Attribute usage:
##
##          97.67% OP_UNIQUE_CARRIER
##          63.33% CRS_ARR_TIME_CONTINUOUS
##          42.42% DAY_OF_WEEK
##          39.01% CRS_DEP_TIME_CONTINUOUS
##          8.52% ORIGIN
##          2.73% DEST
##          0.01% CRS_ELAPSED_TIME
##
##          ##
##          ##
## Time: 8.3 secs

```

Las 14 reglas generadas por el modelo C5.0 pueden interpretarse de la siguiente manera:

1. Si la hora programada de salida es igual o inferior a 674 minutos (antes de las 11:14 de la mañana), entonces hay una alta probabilidad (88.6 %) de que el vuelo llegue a tiempo.
2. Los vuelos con destino a SLC (Salt Lake City) tienen una probabilidad del 87.9 % de llegar a tiempo.

3. Si la hora programada de llegada es igual o antes de las 17:44, entonces el vuelo tiene un 85.7 % de probabilidad de llegar a tiempo.
4. Los vuelos que operan los martes, miércoles y sábados tienen un 84.9 % de probabilidad de llegar a tiempo.
5. Los vuelos del jueves con destino a LGA (LaGuardia Airport, Nueva York) tienen un 83.7 % de probabilidad de llegar a tiempo.
6. Los vuelos originados en ciertos aeropuertos (BUF, CHS, DCA, EWR, ORH, PVD, SFO, SLC) tienen un 82.9 % de probabilidad de llegar a tiempo.
7. Los vuelos operados por cualquier aerolínea que no sea JetBlue, tienen un 82 % de probabilidad de llegar a tiempo.
8. Los vuelos operados por JetBlue los lunes, jueves, viernes y domingos, desde MCO (Orlando) a SLC (Salt Lake City) tienen un 80 % de probabilidad de retraso.
9. Los vuelos de JetBlue con destino a MIA (Miami) que salen después de las 13:59 y tienen un tiempo de vuelo programado de 205 minutos o menos, tienen un 74.4 % de probabilidad de retraso.
10. Los vuelos de JetBlue hacia HPN (White Plains, NY) los lunes, jueves, viernes, o domingos, que salen después de las 13:59 y llegan antes de las 19:51, tienen un 69.2 % de probabilidad de retraso.
11. Los vuelos de JetBlue hacia LaGuardia (LGA) los lunes o domingos, que salen después de las 13:59, tienen un 63.8 % de probabilidad de retraso.
12. Los vuelos de JetBlue hacia una variedad de destinos (ABQ, ATL, BOS, CLT, JFK...) los lunes, jueves, viernes, o domingos, que salen después de las 13:59, tienen un 58.8 % de probabilidad de retraso.
13. Los vuelos de JetBlue que operan los lunes, jueves, viernes, o domingos, que salen entre las 11:14–13:59 y llegan después de las 17:25, tienen un 58.4 % de probabilidad de retraso.
14. Los vuelos de JetBlue que operan los lunes, jueves, viernes, o domingos, y que salen después de las 11:14, tienen un 52.5 % de probabilidad de retraso.

En resumen, los vuelos que salen temprano (antes de las 11:14 de la mañana) presentan las mejores probabilidades de llegar a tiempo. Por otro lado, las condiciones más desfavorables se observan en los vuelos operados por JetBlue, sobre todo los que salen los lunes, jueves, viernes y domingos, desde MCO (Orlando) a SLC (Salt Lake City).

Se extraen las reglas del modelo en formato texto y gráfico.

Dado el elevado número de reglas generadas y el formato de los gráficos en los modelos C5.0, no fue posible mostrar todas las reglas en un único gráfico, incluso aumentando las dimensiones de `fig.height` y `fig.width` a 150. Por ello, optamos por crear un modelo complementario utilizando `CART`.

El parámetro `cp` se emplea para regular el tamaño del árbol. Un valor más bajo resulta en un árbol más grande, con más nodos internos, mientras que un valor mayor poda el árbol, disminuyendo su número de nodos.

El valor predeterminado de 0.01 no resultó suficientemente preciso, generando un árbol sin nodos. Por esta razón, recurrimos a `CART_model$cptable` para encontrar el umbral adecuado que permitiera formar nodos (es decir, construir el árbol más simple posible sin eliminar todos los nodos). Así, establecimos `cp = 0.0008`, obteniendo un árbol con 6 nodos internos (*splits*).

Fuente: <https://www.r-bloggers.com/2021/04/decision-trees-in-r/>

```

library(rpart)
library(rpart.plot)

CART_model <- rpart(ARR_DEL15 ~ ., data = train, method = "class", cp = 0.0008)
CART_model$cptable

```

```

##          CP nsplit rel error xerror      xstd
## 1 0.0009572348      0 1.0000000 1.00000 0.003115603
## 2 0.0008000000      6 0.9939725 0.99778 0.003112935

```

A continuación, extraemos las reglas del modelo en formato de texto.

```
CART_model
```

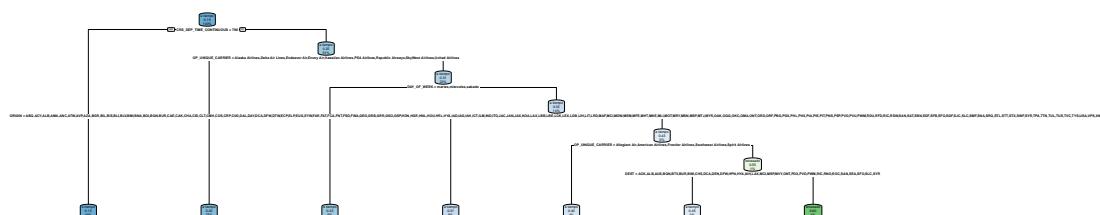
```

## n= 448709
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 448709 83783 a tiempo (0.8132799 0.1867201)
## 2) CRS_DEP_TIME_CONTINUOUS< 789.5 221167 27722 a tiempo (0.8746558 0.1253442) *
## 3) CRS_DEP_TIME_CONTINUOUS>=789.5 227542 56061 a tiempo (0.7536235 0.2463765)
##   6) OP_UNIQUE_CARRIER=Alaska Airlines,Delta Air Lines,Endeavor Air,Envoy Air,Hawaiian Airlines,*
##   7) OP_UNIQUE_CARRIER=Allegiant Air,American Airlines,Frontier Airlines,JetBlue Airways,Southwe
##   14) DAY_OF_WEEK=martes,miercoles,sabado 40709 9519 a tiempo (0.7661696 0.2338304) *
##   15) DAY_OF_WEEK=lunes,jueves,viernes,domingo 64576 22651 a tiempo (0.6492350 0.3507650)
##     30) ORIGIN=ABQ,ACY,ALB,AMA,ANC,ATW,AVP,AZA,BGR,BIL,BIS,BLI,BLV,BMI,BNA,BOI,BQN,BUR,CAE,CAK,
##     31) ORIGIN=ABE,ACK,ATL,AUS,AVL,BDL,BFL,BHM,BOS,BTV,BUF,BWI,BZN,CHS,CLE,DEN,DSM,EWR,FLL,GJT,
##     62) OP_UNIQUE_CARRIER=Allegiant Air,American Airlines,Frontier Airlines,Southwest Airlines
##     63) OP_UNIQUE_CARRIER=JetBlue Airways 3679 1639 retrasado (0.4455015 0.5544985)
##     126) DEST=ACK,ALB,AUS,BQN,BTV,BUR,BWI,CHS,DCA,DEN,DFW,HPN,HYA,IAH,LAX,MCI,MSP,MVY,ONT,PDI
##     127) DEST=ABQ,ATL,BDL,BNA,BOS,BUF,CLE,CLT,DTW,EWR,FLL,JAX,JFK,LAS,LGA,MCO,MIA,MKE,MSY,ORL

```

Mostramos un gráfico con las 6 reglas obtenidas. **Para una mejor visualización, se recomienda abrir la imagen en una nueva pestaña.**

```
rpart.plot(CART_model)
```



Observamos que muchas reglas son similares. Por ejemplo, el primer nodo del modelo C5.0 clasificó los vuelos según su hora de salida programada (antes o después de las 11:14 de la mañana). Por su parte, el modelo CART eligió la misma variable, pero con un umbral aproximadamente dos horas más tarde (13:10).

Además, ambos modelos coinciden en que los vuelos de JetBlue tienen una mayor probabilidad de retraso. La hoja más desfavorable en el modelo CART (con un 60 % de probabilidad de retraso) corresponde a vuelos operados por JetBlue, los lunes, jueves, viernes y domingos, que parten después de las 13:10 desde ciertos aeropuertos (ABQ, ACY, ALB...) con destino a otros específicos (ACK, ALB, AUS...).

Se genera la matriz de confusión para medir la capacidad predictiva del algoritmo, teniendo en cuenta las distintas métricas asociadas a dicha matriz (precisión, sensibilidad, especificidad...).

Primero, evaluamos la capacidad predictiva del modelo C5.0.

```
library(caret)
predicted <- predict(C50_model, testX, type="class" )
confusionMatrix(data = predicted, reference = testy, positive = "retrasado")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction a tiempo retrasado
##   a tiempo      90797     20311
##   retrasado      465       605
##
##                   Accuracy : 0.8148
##                           95% CI : (0.8125, 0.8171)
##   No Information Rate : 0.8135
##   P-Value [Acc > NIR] : 0.1424
##
##                   Kappa : 0.0376
##
##   Mcnemar's Test P-Value : <2e-16
##
##                   Sensitivity : 0.028925
##                   Specificity  : 0.994905
##   Pos Pred Value : 0.565421
##   Neg Pred Value : 0.817196
##           Prevalence : 0.186454
##   Detection Rate : 0.005393
## Detection Prevalence : 0.009538
##   Balanced Accuracy : 0.511915
##
##   'Positive' Class : retrasado
##
```

La matriz de confusión se divide en cuatro cuadrantes:

- **Verdaderos Positivos (TP):** 605 (casos donde el modelo acierta en predecir la clase positiva)
- **Verdaderos Negativos (TN):** 90797 (casos donde el modelo acierta en predecir la clase negativa)
- **Falsos Positivos (FP):** 465 (casos donde el modelo erróneamente predice como positiva la clase negativa)
- **Falsos Negativos (FN):** 20311 (casos donde el modelo no detecta la clase positiva)

La especificidad (*Specificity*) del modelo, que evalúa su habilidad para identificar correctamente los casos negativos (vuelos a tiempo), es muy alta, alcanzando un 99.49 %:

$$\frac{TN}{TN + FP} = \frac{90797}{90797 + 465} = 99.49 \%$$

Sin embargo, dado las clases desbalanceadas en nuestro dataset, la especificidad tiende a sobreestimar la capacidad de nuestro modelo. Las medidas más relevantes en nuestro caso serán la precisión (*Pos Pred Value*) y sensibilidad (*Sensitivity/Recall*).

La precisión es una medida de cuán fiable es la clasificación positiva del modelo. En otras palabras, de todas las predicciones que el modelo clasificó como «retrasado», la precisión nos dice qué proporción de esas predicciones eran realmente correctas. En el caso de este modelo, la precisión es del 56.54 %:

$$\frac{TP}{TP + FP} = \frac{605}{605 + 465} = 56.45 \%$$

Una precisión de 56.54 % está notablemente por encima de la prevalencia (18.64 %), lo que sugiere que el modelo tiene cierta capacidad para identificar vuelos retrasados más allá de la selección aleatoria. Sin embargo, también indica que casi la mitad de los vuelos que el modelo predice como retrasados en realidad llegan a tiempo, lo cual podría llevar a una gestión ineficiente si la predicción se utilizara para tomar decisiones operativas, como la reasignación de recursos en aeropuertos o el cambio de itinerarios de los pasajeros.

Por otra parte, la sensibilidad, que mide la capacidad del modelo para detectar correctamente los casos positivos (vuelos retrasados), es casi inexistente, solo un 2.89 %:

$$\frac{TP}{TP + FN} = \frac{605}{605 + 20311} = 2.89 \%$$

Interpretando conjuntamente la precisión moderada y la baja sensibilidad, la capacidad predictiva de nuestro modelo se resume así: identifica tan solo el 3 % de los vuelos retrasados; no obstante, cuando predice que un vuelo se retrasará, acierta con una frecuencia significativamente más alta que la aleatoriedad (56.45 % frente al 18.64 %).

Por defecto, se establece un umbral de decisión del 50 %. Esto significa que si el modelo calcula una probabilidad de retraso menor al 50 % para un vuelo específico, lo clasifica en la categoría negativa (en este caso, «a tiempo»); si la probabilidad es mayor al 50 %, el vuelo se clasifica como «retrasado».

Ajustando este umbral, es posible incrementar la sensibilidad a costa de reducir la precisión, o viceversa. Esta relación se visualiza a través de la curva de precisión-recall (PR).

```
library(PRROC)

# Predecimos las probabilidades para el conjunto de test
predicted_prob <- predict(C50_model, testX, type="prob")[, "retrasado"]

# Generamos la curva PR utilizando las etiquetas verdaderas (testy)
# y las probabilidades predichas (predicted_prob)
C50_model_pr <- pr.curve(scores.class0 = predicted_prob, weights.class0 = testy == "retrasado", curve = 1)

# Definimos una función wrapper para evitar la repetición del código más adelante
plot_pr_curve <- function(model_pr) {
  # Valor constante que corresponde a la proporción de vuelos retrasados
  no_skill_precision <- mean(flightData$ARR_DEL15 == 1)
```

```

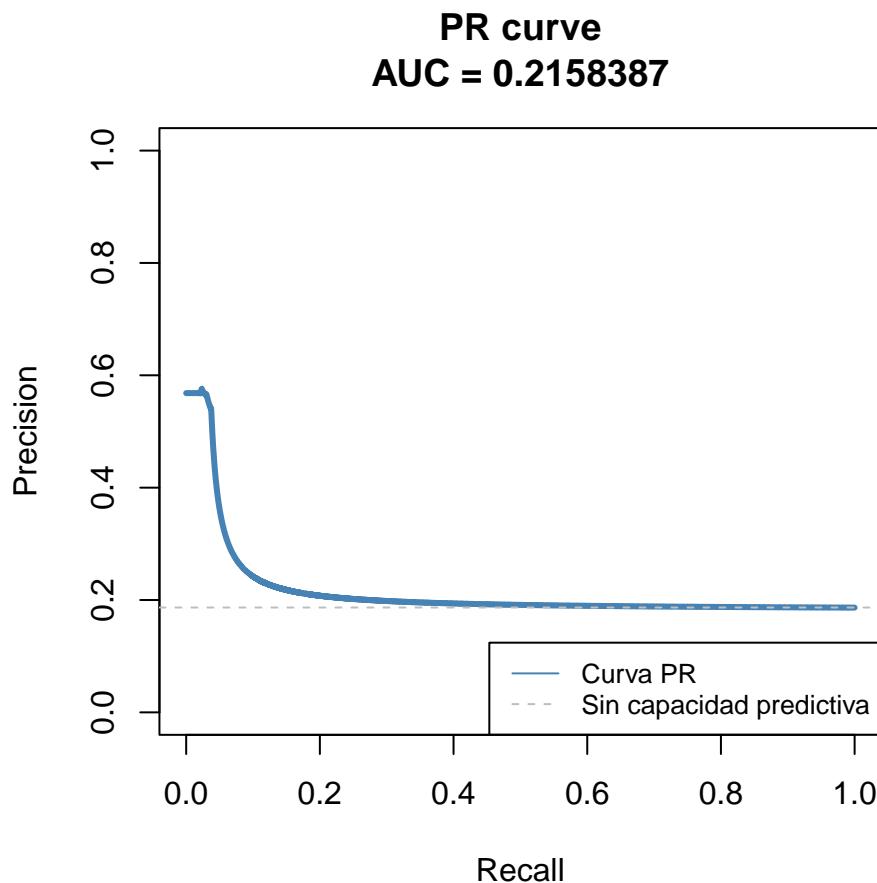
# Creamos un gráfico con la curva PR del modelo
plot(model_pr, color="steelblue")

# Añadimos la línea de capacidad nula
abline(h = no_skill_precision, col = 'grey', lty = 2)

# Añadimos una leyenda
legend("bottomright", legend=c("Curva PR", "Sin capacidad predictiva"),
       col=c("steelblue", "grey"), lty=c(1, 2), cex=0.8)
}

# Creamos el gráfico con la curva PR
plot_pr_curve(C50_model_pr)

```



El gráfico muestra dos líneas: la curva PR, en azul, y la línea gris, que representa la proporción de vuelos retrasados en el conjunto de datos (18.64 %). Nuestro objetivo es que la curva se posicione lo más cerca posible de la esquina superior derecha. En contraste, si la curva PR se approxima a la línea gris, indica que nuestro modelo no es mejor que hacer predicciones al azar a ese nivel de recall.

Del análisis del gráfico se deduce que nuestro modelo exhibe una capacidad predictiva aceptable únicamente cuando el nivel de recall es extremadamente bajo. No obstante, al intentar reducir la cantidad de falsos negativos, la cantidad de falsos positivos se incrementa de manera significativa, lo cual menoscaba la utilidad

del modelo a medida que el nivel de recall se sobrepasa el 10 %.

El área bajo la curva PR, conocida como AUC (*Area Under the Curve*), nos aporta una estimación sobre la capacidad predictiva global del modelo. Similarmente a la precisión, el AUC deseado debería exceder la prevalencia de la clase positiva (18.64 %), y acercarse tanto como sea posible al valor ideal de 1.

En este escenario, el AUC es de 0.2158 (21.58 %), lo que señala que, aunque el modelo tiene una capacidad predictiva superior a la del azar (18.64 %), esta es limitada.

Supongamos que estamos dispuestos a sacrificar algo de precisión para mejorar el recall, aceptando una precisión mínima de 0.50. Buscamos entonces el umbral óptimo.

```
library(knitr)
library(kableExtra)

# Definimos una función para evitar la repetición del código más adelante
find_best_recall <- function(pr_matrix, min_precision) {
  # Convertimos la matriz en un dataframe
  pr_matrix <- as.data.frame(pr_matrix)

  # Etiquetamos las columnas del dataframe
  names(pr_matrix) <- c("Recall", "Precision", "Threshold")

  # Filtramos los puntos de la curva PR donde la precisión es al menos min_precision y el umbral no es 0
  pr_above_min <- subset(pr_matrix, Precision >= min_precision & Threshold != 0)

  # Buscamos el punto con el recall más alto
  highest_recall <- pr_above_min[which.max(pr_above_min$Recall),]

  # Devolvemos la tabla en formato kable
  kable(highest_recall, digits = 4, row.names = FALSE) %>%
    kable_styling(bootstrap_options = c("striped", "hover"))
}

find_best_recall(C50_model_pr$curve, min_precision = 0.50)
```

| Recall | Precision | Threshold |
|--------|-----------|-----------|
| 0.0377 | 0.5342 | 0.1729 |

Si disminuyéramos el umbral de decisión de 0.50 a 0.1729, observaríamos los siguientes cambios:

- La precisión descendería del 56.45 % al 53.42 %.
- El recall aumentaría del 2.89 % al 3.77 %.

Esto implicaría una reducción de los falsos negativos, pero un incremento en los falsos positivos.

A continuación, evaluaremos la capacidad predictiva del modelo CART.

```
predicted_classes <- predict(CART_model, testX, type="class")
caret::confusionMatrix(data = predicted_classes, reference = testy, positive = "retrasado")
```

```

## Confusion Matrix and Statistics
##
##             Reference
## Prediction a tiempo retrasado
##      a tiempo     90990     20548
##      retrasado      272      368
##
##                  Accuracy : 0.8144
##                  95% CI : (0.8121, 0.8167)
##      No Information Rate : 0.8135
##      P-Value [Acc > NIR] : 0.2322
##
##                  Kappa : 0.0233
##
## Mcnemar's Test P-Value : <2e-16
##
##                  Sensitivity : 0.017594
##                  Specificity : 0.997020
##      Pos Pred Value : 0.575000
##      Neg Pred Value : 0.815776
##                  Prevalence : 0.186454
##                  Detection Rate : 0.003281
##      Detection Prevalence : 0.005705
##      Balanced Accuracy : 0.507307
##
##      'Positive' Class : retrasado
##

```

A primera vista, los resultados del modelo CART parecen inferiores a los del modelo C5.0. La precisión es ligeramente más alta, pero el recall es casi la mitad.

| | Modelo | Precisión | Recall |
|------|----------|-----------|--------|
| C5.0 | 0.565421 | 0.028925 | |
| CART | 0.575000 | 0.017594 | |

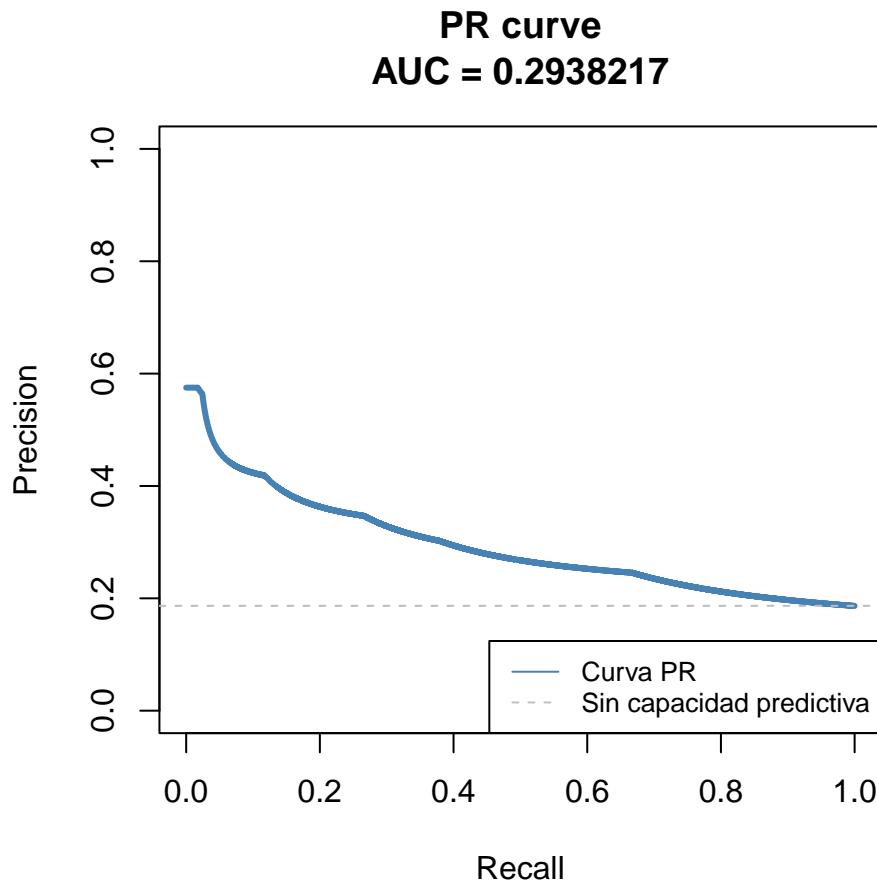
A continuación, visualizamos la curva PR para el modelo CART.

```

predicted_prob <- predict(CART_model, testX, type="prob")[, "retrasado"]

CART_model_pr <- pr.curve(scores.class0 = predicted_prob, weights.class0 = testy == "retrasado", curve =
plot_pr_curve(CART_model_pr)

```



Aunque la precisión máxima en el modelo CART es similar a la del modelo C5.0, su curva PR desciende más gradualmente, lo que sugiere que mantiene cierta capacidad predictiva incluso a niveles altos de recall.

Además, el AUC para el modelo CART alcanza 0.294, frente al 0.216 del modelo C5.0. Esto nos indica que, en términos generales, la capacidad predictiva del modelo CART es más robusta.

No obstante, si mantenemos el mismo estándar previo y optamos por no aceptar una precisión por debajo del 50 %, entonces:

```
find_best_recall(CART_model_pr$curve, min_precision = 0.50)
```

| Recall | Precision | Threshold |
|--------|-----------|-----------|
| 0.0342 | 0.5 | 0.4033 |

Bajo estas condiciones, el recall del modelo CART alcanza solo un 3.42 %, en contraste con el 3.77 % del modelo C5.0. Por lo tanto, si priorizamos una alta precisión y queremos minimizar los falsos positivos, el modelo C5.0 emerge como la opción preferente para identificar una mayor proporción de vuelos retrasados.

Fuente 1: <https://cran.r-project.org/web/packages/PRROC/vignettes/PRROC.pdf>

Fuente 2: <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/>

Se comparan e interpretan los resultados (sin y con opciones de poda), explicando las ventajas e inconvenientes del modelo generado respecto a otro método de construcción.

Por defecto, el modelo C5.0 utiliza la poda, pero se puede deshabilitar con el argumento noGlobalPruning = TRUE.

Fuente: <https://topepo.github.io/C5.0/reference/C5.0Control.html>

```
C50_model12 <- C5.0(trainX, trainy, rules=TRUE, control = C5.0Control(noGlobalPruning = TRUE))
summary(C50_model12)
```

```
##
## Call:
## C5.0.default(x = trainX, y = trainy, rules = TRUE, control
##   = C5.0Control(noGlobalPruning = TRUE))
##
##
## C5.0 [Release 2.07 GPL Edition]      Fri Feb  9 20:51:59 2024
## -----
##
## Class specified by attribute 'outcome'
##
## Read 448709 cases (8 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (17863/2096, lift 1.1)
##   DEST = ATL
##   CRS_ELAPSED_TIME <= 146
##   -> class a tiempo [0.883]
##
## Rule 2: (182897/27563, lift 1.0)
##   DAY_OF_WEEK in {martes, miercoles, sabado}
##   -> class a tiempo [0.849]
##
## Rule 3: (265812/56220, lift 1.0)
##   DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
##   -> class a tiempo [0.788]
##
## Rule 4: (10, lift 4.9)
##   DAY_OF_WEEK in {lunes, viernes, domingo}
##   OP_UNIQUE_CARRIER = JetBlue Airways
##   ORIGIN = SJU
##   DEST = TPA
##   CRS_DEP_TIME_CONTINUOUS > 674
##   -> class retrasado [0.917]
##
## Rule 5: (6, lift 4.7)
##   DAY_OF_WEEK = viernes
##   OP_UNIQUE_CARRIER = JetBlue Airways
##   DEST = ACK
##   CRS_DEP_TIME_CONTINUOUS > 839
##   CRS_ELAPSED_TIME > 60
```

```

##  -> class retrasado [0.875]
##
## Rule 6: (6, lift 4.7)
## DAY_OF_WEEK in {lunes, jueves}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = FLL
## DEST = ATL
## CRS_DEP_TIME_CONTINUOUS > 1225
## -> class retrasado [0.875]
##
## Rule 7: (13/1, lift 4.6)
## DAY_OF_WEEK = domingo
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = ATL
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.867]
##
## Rule 8: (5, lift 4.6)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = MCO
## DEST = BUF
## -> class retrasado [0.857]
##
## Rule 9: (5, lift 4.6)
## DAY_OF_WEEK in {lunes, viernes}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = IAH
## CRS_DEP_TIME_CONTINUOUS > 877
## -> class retrasado [0.857]
##
## Rule 10: (5, lift 4.6)
## DAY_OF_WEEK = jueves
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = BOS
## DEST = LAX
## CRS_DEP_TIME_CONTINUOUS > 674
## -> class retrasado [0.857]
##
## Rule 11: (5, lift 4.6)
## DAY_OF_WEEK = lunes
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = JFK
## DEST = LAX
## CRS_ELAPSED_TIME > 373
## -> class retrasado [0.857]
##
## Rule 12: (4, lift 4.5)
## DAY_OF_WEEK = lunes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = BNA
## CRS_DEP_TIME_CONTINUOUS > 893
## -> class retrasado [0.833]
##

```

```

## Rule 13: (4, lift 4.5)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = RNO
## CRS_DEP_TIME_CONTINUOUS <= 973
## -> class retrasado [0.833]
##
## Rule 14: (20/3, lift 4.4)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = CLE
## CRS_DEP_TIME_CONTINUOUS > 920
## CRS_ELAPSED_TIME <= 121
## -> class retrasado [0.818]
##
## Rule 15: (19/3, lift 4.3)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = PHL
## CRS_DEP_TIME_CONTINUOUS > 674
## CRS_ELAPSED_TIME <= 98
## -> class retrasado [0.810]
##
## Rule 16: (3, lift 4.3)
## DAY_OF_WEEK = domingo
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = BNA
## CRS_DEP_TIME_CONTINUOUS > 674
## CRS_ELAPSED_TIME <= 156
## -> class retrasado [0.800]
##
## Rule 17: (3, lift 4.3)
## DAY_OF_WEEK = domingo
## DEST = BQN
## CRS_DEP_TIME_CONTINUOUS > 674
## CRS_DEP_TIME_CONTINUOUS <= 909
## -> class retrasado [0.800]
##
## Rule 18: (8/1, lift 4.3)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = BOS
## DEST = LAX
## CRS_DEP_TIME_CONTINUOUS > 674
## CRS_ELAPSED_TIME <= 391
## -> class retrasado [0.800]
##
## Rule 19: (3, lift 4.3)
## DAY_OF_WEEK = viernes
## ORIGIN = BUF
## DEST = LAX
## -> class retrasado [0.800]
##
## Rule 20: (3, lift 4.3)

```

```

##  DAY_OF_WEEK = jueves
##  ORIGIN = BOS
##  DEST = PBI
##  CRS_DEP_TIME_CONTINUOUS > 990
##  -> class retrasado [0.800]
##
## Rule 21: (3, lift 4.3)
##  DAY_OF_WEEK = viernes
##  ORIGIN = PVD
##  DEST = PBI
##  -> class retrasado [0.800]
##
## Rule 22: (13/2, lift 4.3)
##  DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
##  OP_UNIQUE_CARRIER = JetBlue Airways
##  ORIGIN = MCO
##  DEST = SLC
##  -> class retrasado [0.800]
##
## Rule 23: (7/1, lift 4.2)
##  DAY_OF_WEEK in {lunes, jueves, domingo}
##  OP_UNIQUE_CARRIER = JetBlue Airways
##  ORIGIN = MIA
##  DEST = EWR
##  CRS_DEP_TIME_CONTINUOUS > 839
##  -> class retrasado [0.778]
##
## Rule 24: (16/3, lift 4.2)
##  DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
##  OP_UNIQUE_CARRIER = JetBlue Airways
##  ORIGIN in {MCO, MIA}
##  DEST = LAX
##  CRS_DEP_TIME_CONTINUOUS > 839
##  -> class retrasado [0.778]
##
## Rule 25: (33/7, lift 4.1)
##  DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
##  OP_UNIQUE_CARRIER = JetBlue Airways
##  DEST = DTW
##  CRS_ARR_TIME_CONTINUOUS > 1091
##  CRS_ELAPSED_TIME <= 135
##  -> class retrasado [0.771]
##
## Rule 26: (11/2, lift 4.1)
##  DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
##  OP_UNIQUE_CARRIER = JetBlue Airways
##  ORIGIN = BDL
##  DEST = PBI
##  CRS_DEP_TIME_CONTINUOUS > 839
##  -> class retrasado [0.769]
##
## Rule 27: (11/2, lift 4.1)
##  DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
##  OP_UNIQUE_CARRIER = JetBlue Airways

```

```

## DEST = PIT
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_DEP_TIME_CONTINUOUS <= 1132
## CRS_ELAPSED_TIME <= 110
## -> class retrasado [0.769]
##
## Rule 28: (10/2, lift 4.0)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = BNA
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.750]
##
## Rule 29: (22/5, lift 4.0)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = LGA
## DEST = ATL
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME > 146
## -> class retrasado [0.750]
##
## Rule 30: (62/15, lift 4.0)
## DAY_OF_WEEK in {lunes, jueves, viernes}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = FLL
## CRS_DEP_TIME_CONTINUOUS > 1225
## -> class retrasado [0.750]
##
## Rule 31: (10/2, lift 4.0)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = RIC
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.750]
##
## Rule 32: (41/10, lift 4.0)
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = MIA
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME <= 205
## -> class retrasado [0.744]
##
## Rule 33: (12/3, lift 3.8)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = ABQ
## CRS_DEP_TIME_CONTINUOUS > 1189
## -> class retrasado [0.714]
##
## Rule 34: (5/1, lift 3.8)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = LAS

```

```

##  DEST = LAX
## -> class retrasado [0.714]
##
## Rule 35: (5/1, lift 3.8)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = RNO
## DEST = LAX
## -> class retrasado [0.714]
##
## Rule 36: (82/23, lift 3.8)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = MCO
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.714]
##
## Rule 37: (19/5, lift 3.8)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## ORIGIN = JFK
## DEST = PBI
## CRS_DEP_TIME_CONTINUOUS > 674
## CRS_ELAPSED_TIME <= 185
## -> class retrasado [0.714]
##
## Rule 38: (15/4, lift 3.8)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## ORIGIN = BOS
## DEST = SJU
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_DEP_TIME_CONTINUOUS <= 1395
## -> class retrasado [0.706]
##
## Rule 39: (167/49, lift 3.8)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN in {AUS, BNA, BOS, BQN, HPN, JAX, JFK, LAS, LAX, LGA, MSY, SAN,
## SJU}
## DEST = FLL
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.704]
##
## Rule 40: (48/14, lift 3.7)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = BDL
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME <= 279
## -> class retrasado [0.700]
##
## Rule 41: (80/24, lift 3.7)
## DAY_OF_WEEK = lunes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = LGA

```

```

## CRS_DEP_TIME_CONTINUOUS > 674
## -> class retrasado [0.695]
##
## Rule 42: (24/7, lift 3.7)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = HPN
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ARR_TIME_CONTINUOUS <= 1191
## -> class retrasado [0.692]
##
## Rule 43: (11/3, lift 3.7)
## DAY_OF_WEEK in {lunes, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = PSE
## -> class retrasado [0.692]
##
## Rule 44: (49/15, lift 3.7)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = DCA
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME <= 109
## -> class retrasado [0.686]
##
## Rule 45: (10/3, lift 3.6)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = RSW
## DEST = EWR
## CRS_DEP_TIME_CONTINUOUS > 1007
## -> class retrasado [0.667]
##
## Rule 46: (22/7, lift 3.6)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = LGA
## DEST = MSY
## CRS_DEP_TIME_CONTINUOUS > 674
## -> class retrasado [0.667]
##
## Rule 47: (22/7, lift 3.6)
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = STT
## CRS_DEP_TIME_CONTINUOUS <= 938
## -> class retrasado [0.667]
##
## Rule 48: (21/7, lift 3.5)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = RSW
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ARR_TIME_CONTINUOUS > 544
## CRS_ELAPSED_TIME > 173

```

```

##  -> class retrasado [0.652]
##
## Rule 49: (38/13, lift 3.5)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN in {FLL, JFK, LGA}
## DEST = JAX
## CRS_DEP_TIME_CONTINUOUS > 674
## -> class retrasado [0.650]
##
## Rule 50: (38/13, lift 3.5)
## DAY_OF_WEEK = jueves
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = MCO
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME > 178
## -> class retrasado [0.650]
##
## Rule 51: (15/5, lift 3.5)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = JFK
## DEST = PDX
## -> class retrasado [0.647]
##
## Rule 52: (46/16, lift 3.5)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN in {FLL, MCO, SJU}
## DEST = EWR
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.646]
##
## Rule 53: (29/10, lift 3.5)
## DAY_OF_WEEK in {jueves, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN = JFK
## DEST = LAX
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.645]
##
## Rule 54: (12/4, lift 3.4)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## ORIGIN = FLL
## DEST = SAN
## CRS_DEP_TIME_CONTINUOUS > 1095
## -> class retrasado [0.643]
##
## Rule 55: (12/4, lift 3.4)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = SMF
## CRS_ARR_TIME_CONTINUOUS <= 1314
## -> class retrasado [0.643]

```

```

## Rule 56: (45/16, lift 3.4)
## DAY_OF_WEEK = domingo
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = LGA
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME <= 166
## -> class retrasado [0.638]
##
## Rule 57: (202/73, lift 3.4)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = MCO
## CRS_DEP_TIME_CONTINUOUS > 839
## CRS_ELAPSED_TIME <= 183
## -> class retrasado [0.637]
##
## Rule 58: (9/3, lift 3.4)
## DAY_OF_WEEK = viernes
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = BQN
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.636]
##
## Rule 59: (82/31, lift 3.3)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN in {FLL, JFK, MCO}
## DEST = SJU
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.619]
##
## Rule 60: (1698/658, lift 3.3)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST in {AUS, BOS, CLT, JFK, LAS, MKE, MSP, ORD, PHX, RDU, ROC, SAV,
##           SEA, SJC}
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.612]
##
## Rule 61: (13/5, lift 3.2)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## ORIGIN = FLL
## DEST = LAX
## CRS_ARR_TIME_CONTINUOUS > 1280
## -> class retrasado [0.600]
##
## Rule 62: (13/5, lift 3.2)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## DEST = STT
## CRS_DEP_TIME_CONTINUOUS <= 849
## -> class retrasado [0.600]
##

```

```

## Rule 63: (79/32, lift 3.2)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN in {BOS, FLL, JFK, LAX}
## DEST = BUF
## CRS_DEP_TIME_CONTINUOUS > 674
## -> class retrasado [0.593]
##
## Rule 64: (39/16, lift 3.1)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## ORIGIN in {BOS, FLL}
## DEST = CHS
## -> class retrasado [0.585]
##
## Rule 65: (4534/2007, lift 3.0)
## DAY_OF_WEEK in {lunes, jueves, viernes, domingo}
## OP_UNIQUE_CARRIER = JetBlue Airways
## CRS_DEP_TIME_CONTINUOUS > 839
## -> class retrasado [0.557]
##
## Default class: a tiempo
##
##
## Evaluation on training data (448709 cases):
##
##          Rules
##          -----
##          No      Errors
##          -----
##          65 82831(18.5%)  <<
##
##          (a)      (b)      <-classified as
##          -----  -----
##          363846    1080      (a): class a tiempo
##          81751     2032      (b): class retrasado
##
##          Attribute usage:
##          -
##          100.00% DAY_OF_WEEK
##          4.68% DEST
##          4.11% CRS_ELAPSED_TIME
##          1.02% OP_UNIQUE_CARRIER
##          1.02% CRS_DEP_TIME_CONTINUOUS
##          0.17% ORIGIN
##          0.02% CRS_ARR_TIME_CONTINUOUS
##
##          -
##          -
## Time: 12.6 secs

```

Con esta modificación, el número de reglas ha incrementado de 14 a 65. Procedemos a recrear la matriz de confusión.

```

predicted <- predict(C50_model2, testX, type="class")
caret::confusionMatrix(data = predicted, reference = testy, positive = "retrasado")

```

```

## Confusion Matrix and Statistics
##
##             Reference
## Prediction a tiempo retrasado
##   a tiempo      90954     20454
##   retrasado      308       462
##
##                  Accuracy : 0.8149
##                  95% CI : (0.8126, 0.8172)
##      No Information Rate : 0.8135
##      P-Value [Acc > NIR] : 0.1196
##
##                  Kappa : 0.0298
##
## McNemar's Test P-Value : <2e-16
##
##                  Sensitivity : 0.022088
##                  Specificity  : 0.996625
##      Pos Pred Value : 0.600000
##      Neg Pred Value : 0.816405
##      Prevalence    : 0.186454
##      Detection Rate : 0.004118
##      Detection Prevalence : 0.006864
##      Balanced Accuracy : 0.509357
##
##      'Positive' Class : retrasado
##

```

Sorprendentemente, la precisión del modelo C5.0 sin poda resulta superior a la del modelo con poda, al menos con un umbral de decisión de 0.5. Sin embargo, el recall del modelo sin poda es notablemente menor.

| Modelo | Precisión | Recall |
|---------------|-----------|----------|
| C5.0 con poda | 0.565421 | 0.028925 |
| C5.0 sin poda | 0.600000 | 0.022088 |

Volvemos a calcular la curva PR y el AUC para el modelo C5.0 sin poda.

```

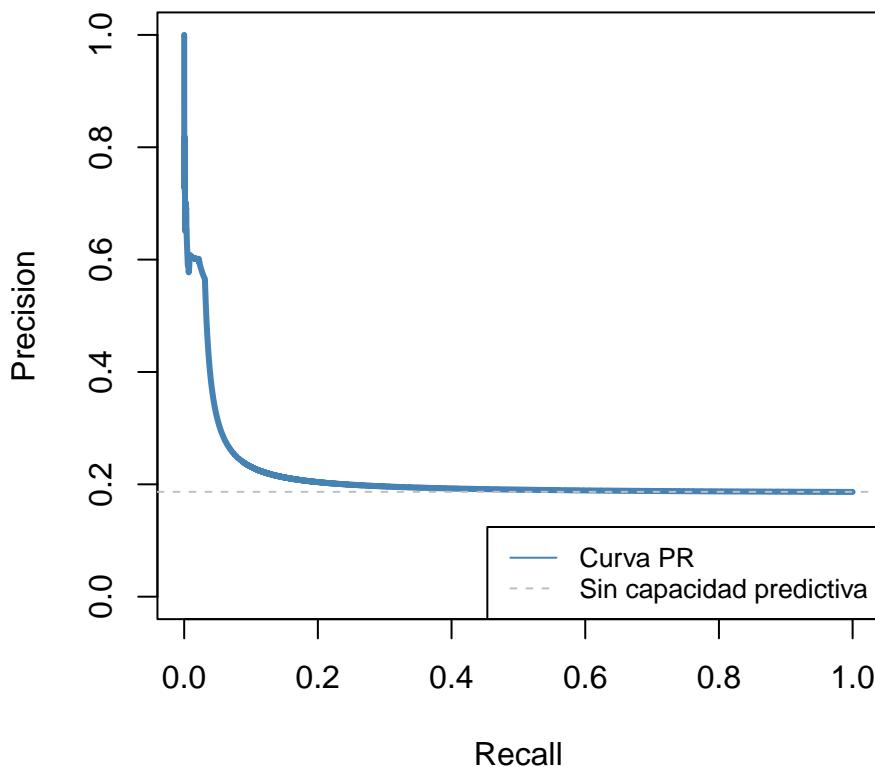
predicted_prob <- predict(C50_model2, testX, type="prob")[, "retrasado"]

C50_model2_pr <- pr.curve(scores.class0 = predicted_prob, weights.class0 = testy == "retrasado", curve = TRUE)

plot_pr_curve(C50_model2_pr)

```

PR curve
AUC = 0.2133291



Observamos que la precisión llega hasta el 100 % para los niveles más bajos de recall. Sin embargo, el AUC es ligeramente inferior, llegando a 0.2133 sin poda frente a 0.2158 con poda.

Y si mantenemos el mismo estándar previo y optamos por no aceptar una precisión por debajo del 50 %, entonces:

```
find_best_recall(C50_model2_pr$curve, min_precision = 0.50)
```

| Recall | Precision | Threshold |
|--------|-----------|-----------|
| 0.031 | 0.5653 | 0.2611 |

Bajo estas condiciones, el recall del modelo C5.0 sin poda alcanza solo un 3.10 %, en contraste con el 3.77 % del modelo con poda.

Aunque el modelo CART no ofrece una opción para activar o desactivar la poda directamente, es posible reducir el valor de `cp` a 0.000165 para incrementar el número de nodos de 6 a 50.

```
CART_model2 <- rpart(ARR_DEL15 ~., data = train, method = "class", cp = 0.000165)
CART_model2$cptable
```

```
##          CP nsplit rel error      xerror       xstd

```

```

## 1 0.0009572348      0 1.0000000 1.0000000 0.003115603
## 2 0.0007917278      6 0.9939725 0.9965745 0.003111485
## 3 0.0007320499      9 0.9915973 0.9954525 0.003110133
## 4 0.0006087154     12 0.9894012 0.9945693 0.003109068
## 5 0.0005490374     13 0.9887925 0.9938890 0.003108246
## 6 0.0004535526     14 0.9882434 0.9940083 0.003108391
## 7 0.0003401645     15 0.9877899 0.9939964 0.003108376
## 8 0.0003222611     19 0.9864292 0.9937099 0.003108030
## 9 0.0002327441     20 0.9861070 0.9941874 0.003108607
## 10 0.0002227978    22 0.9856415 0.9953809 0.003110046
## 11 0.0002208085    25 0.9849731 0.9953690 0.003110032
## 12 0.0002088729    27 0.9845315 0.9953929 0.003110061
## 13 0.0002029051    29 0.9841137 0.9954167 0.003110090
## 14 0.0001909695    30 0.9839108 0.9958345 0.003110593
## 15 0.0001885824    31 0.9837198 0.9959300 0.003110708
## 16 0.0001830125    39 0.9819056 0.9957629 0.003110507
## 17 0.0001814211    43 0.9810940 0.9958822 0.003110651
## 18 0.0001670983    48 0.9801869 0.9962522 0.003111096
## 19 0.0001650000    50 0.9798527 0.9963835 0.003111255

```

Generamos la matriz de confusión correspondiente.

```

predicted <- predict(CART_model2, testX, type="class")
caret::confusionMatrix(data = predicted, reference = testy, positive = "retrasado")

```

```

## Confusion Matrix and Statistics
##
##                  Reference
## Prediction a tiempo retrasado
##   a tiempo      90271     19777
##   retrasado      991      1139
##
##                  Accuracy : 0.8149
##                  95% CI : (0.8126, 0.8171)
##      No Information Rate : 0.8135
##      P-Value [Acc > NIR] : 0.129
##
##                  Kappa : 0.0667
##
## McNemar's Test P-Value : <2e-16
##
##                  Sensitivity : 0.05446
##                  Specificity  : 0.98914
##      Pos Pred Value : 0.53474
##      Neg Pred Value : 0.82029
##      Prevalence    : 0.18645
##      Detection Rate : 0.01015
##      Detection Prevalence : 0.01899
##      Balanced Accuracy : 0.52180
##
##      'Positive' Class : retrasado
##

```

En comparación con el modelo CART de 6 nodos, la precisión ha descendido un poco, pero el recall ha mejorado notablemente.

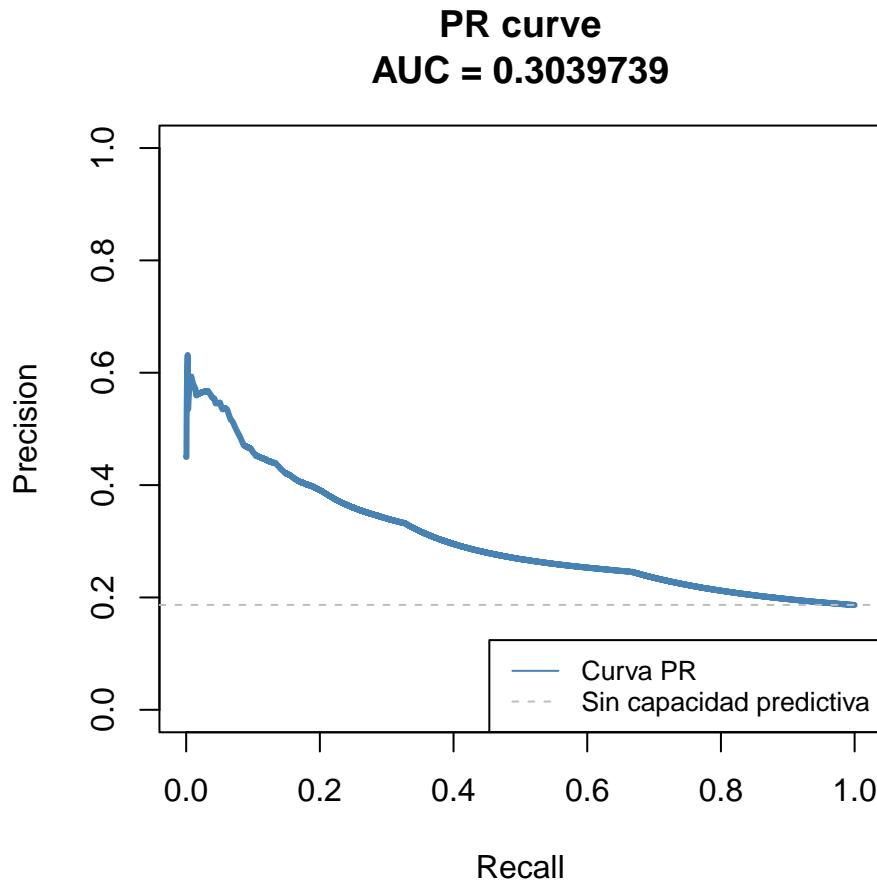
| Modelo | Precisión | Recall |
|-----------------|-----------|----------|
| CART (6 nodos) | 0.57500 | 0.017594 |
| CART (50 nodos) | 0.53474 | 0.054460 |

A continuación, visualizamos la curva PR y calculamos el AUC para el modelo CART con 50 nodos.

```
predicted_prob <- predict(CART_model2, testX, type="prob")[, "retrasado"]

CART_model2_pr <- pr.curve(scores.class0 = predicted_prob, weights.class0 = testy == "retrasado", curve = 1)

plot_pr_curve(CART_model2_pr)
```



El AUC del modelo con 50 nodos es de 0.304, comparado con el 0.294 alcanzado por el modelo de 6 nodos.

Además, si mantenemos el mismo estándar previo y optamos por no aceptar una precisión por debajo del 50 %, entonces:

```
find_best_recall(CART_model2_pr$curve, min_precision = 0.50)
```

| Recall | Precision | Threshold |
|--------|-----------|-----------|
| 0.0743 | 0.5001 | 0.4081 |

El modelo de 50 nodos logra un recall del 7.43 %, superando con creces el 3.42 % alcanzado por el modelo de 6 nodos. Esto sugiere que el modelo inicial podría haber sido demasiado simple.

Una de las ventajas de los árboles de decisión es su facilidad de visualización e interpretación. Además, requieren menos complejidad computacional para su entrenamiento en comparación con otros modelos, como los *Random Forest*, que examinaremos en el ejercicio 6.

Entre sus inconvenientes, los árboles de decisión son más propensos al sobreajuste (*overfitting*), aunque este riesgo puede mitigarse ligeramente a través de parámetros como *cp* en los modelos CART. Otra desventaja es que suelen presentar un rendimiento predictivo inferior en comparación con otros algoritmos más complejos.

Se evalúa la tasa de error en cada nivel de árbol, la eficiencia en la clasificación (en las muestras de entrenamiento y test) y la comprensibilidad del resultado.

Nota: Este apartado se centra en el modelo CART con 6 nodos.

Primero, transformamos cada regla del árbol en una máscara.

```
rule1 <- testX$CRS_DEPENDENT_CONTINUOUS < 789.5

rule2 <- testX$OP_UNIQUE_CARRIER %in% c("Alaska Airlines", "Delta Air Lines", "Endeavor Air",
                                         "Envoy Air", "Hawaiian Airlines", "PSA Airlines",
                                         "Republic Airways", "SkyWest Airlines", "United Airlines")

rule3 <- testX$DAY_OF_WEEK %in% c("martes", "miercoles", "sabado")

rule4 <- testX$ORIGIN %in% c("ABQ", "ACY", "ALB", "AMA", "ANC", "ATW", "AVP", "AZA", "BGR", "BIL",
                            "BIS", "BLI", "BLV", "BMI", "BNA", "BOI", "BQN", "BUR", "CAE", "CAK",
                            "CHA", "CID", "CLT", "CMH", "COS", "CRP", "CVG", "DAL", "DAY", "DCA",
                            "DFW", "DTW", "ECP", "ELP", "EUG", "EYW", "FAR", "FAT", "FCA", "FNT",
                            "FSD", "FWA", "GEG", "GRB", "GRR", "GSO", "GSP", "HDN", "HGR", "HNL",
                            "HOU", "HRL", "HYA", "IAD", "IAG", "IAH", "ICT", "ILM", "IND", "ITO",
                            "JAC", "JAN", "JAX", "KOA", "LAX", "LBB", "LBE", "LCK", "LEX", "LGB",
                            "LIH", "LIT", "LRD", "MAF", "MCI", "MDW", "MEM", "MFE", "MHT", "MKE",
                            "MLI", "MOT", "MRY", "MSN", "MSP", "MTJ", "MYR", "OAK", "OGG", "OKC",
                            "OMA", "ONT", "ORD", "ORF", "PBG", "PDX", "PHL", "PHX", "PIA", "PIE",
                            "PIT", "PNS", "PSP", "PVD", "PVU", "PWM", "RDU", "RFD", "RIC", "RSW",
                            "SAN", "SAT", "SBN", "SDF", "SFB", "SFO", "SGF", "SJC", "SLC", "SMF",
                            "SNA", "SRQ", "STL", "STT", "STX", "SWF", "SYR", "TPA", "TTN", "TUL",
                            "TUS", "TVC", "TYS", "USA", "VPS", "XNA")

rule5 <- testX$OP_UNIQUE_CARRIER %in% c("Allegiant Air", "American Airlines", "Frontier Airlines",
                                         "Southwest Airlines", "Spirit Airlines")

rule6 <- testX$DEST %in% c("ACK", "ALB", "AUS", "BQN", "BTW", "BUR", "BWI", "CHS", "DCA", "DEN",
                           "DFW", "HPN", "HYA", "IAH", "LAX", "MCI", "MSP", "MVY", "ONT", "PDX",
                           "PVD", "PWM", "RIC", "RNO", "ROC", "SAN", "SEA", "SFO", "SLC", "SYR")
```

A continuación, calculamos la tasa de error en cada nivel del árbol.

```
node1_error_rate <- mean(testy[rule1] == "retrasado")
node2_error_rate <- mean(testy[!rule1 & rule2] == "retrasado")
node3_error_rate <- mean(testy[!rule1 & !rule2 & rule3] == "retrasado")
node4_error_rate <- mean(testy[!rule1 & !rule2 & !rule3 & rule4] == "retrasado")
node5_error_rate <- mean(testy[!rule1 & !rule2 & !rule3 & !rule4 & rule5] == "retrasado")
node6_error_rate <- mean(testy[!rule1 & !rule2 & !rule3 & !rule4 & !rule5 & rule6] == "retrasado")
```

Presentamos los resultados en forma de tabla y gráfico.

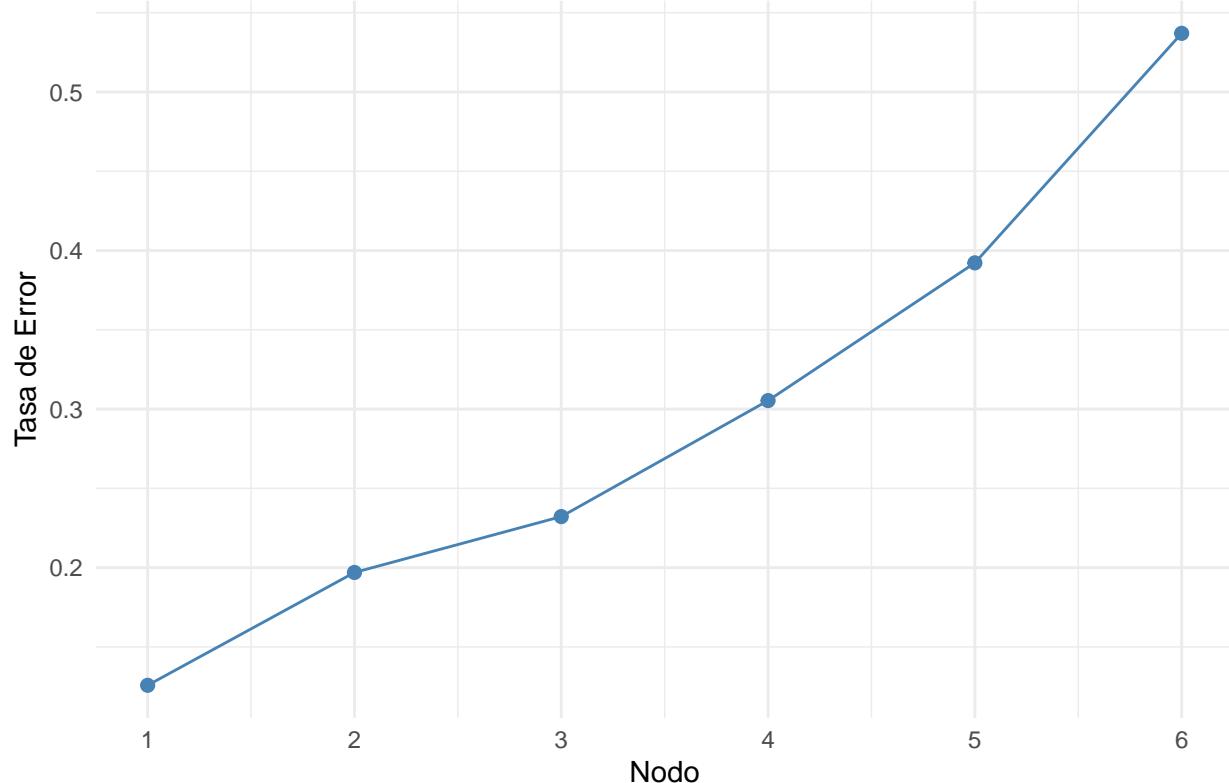
```
error_rates <- data.frame(
  node = 1:6,
  error_rate = c(node1_error_rate, node2_error_rate, node3_error_rate,
                 node4_error_rate, node5_error_rate, node6_error_rate)
)

kable(error_rates, digits=3, col.names = c("Nodo", "Tasa de Error")) %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

| Nodo | Tasa de Error |
|------|---------------|
| 1 | 0.126 |
| 2 | 0.197 |
| 3 | 0.232 |
| 4 | 0.305 |
| 5 | 0.392 |
| 6 | 0.537 |

```
ggplot(error_rates, aes(x = node, y = error_rate)) +
  geom_line(color = "steelblue") +
  geom_point(size = 2, color = "steelblue") +
  scale_x_continuous(breaks = 1:6) +
  labs(title = "Tasa de error en cada nivel del árbol (modelo CART)",
       x = "Nodo",
       y = "Tasa de Error") +
  theme_minimal()
```

Tasa de error en cada nivel del árbol (modelo CART)



Notamos que la tasa de error del primer nodo es moderada, situándose en un 12.6 %. Sin embargo, esta tasa incrementa progresivamente a lo largo de los nodos, alcanzando un 53.7 % en el nodo final.

Para analizar la eficiencia en la clasificación, recurrimos al AUC. Ya se ha calculado el AUC para el conjunto `test`, y a continuación, calculamos el AUC para cada modelo en el conjunto `train`.

```
predicted_prob <- predict(C50_model, trainX, type="prob")[, "retrasado"]
C50_model_train_pr <- pr.curve(scores.class0 = predicted_prob,
                                 weights.class0 = trainy == "retrasado",
                                 curve = TRUE)
C50_model_train_pr$auc.integral
```

```
## [1] 0.2161497
```

```
predicted_prob <- predict(C50_model2, trainX, type="prob")[, "retrasado"]
C50_model2_train_pr <- pr.curve(scores.class0 = predicted_prob,
                                 weights.class0 = trainy == "retrasado",
                                 curve = TRUE)
C50_model2_train_pr$auc.integral
```

```
## [1] 0.2160281
```

```
predicted_prob <- predict(CART_model, trainX, type="prob")[, "retrasado"]
CART_model_train_pr <- pr.curve(scores.class0 = predicted_prob,
                                 weights.class0 = trainy == "retrasado",
```

```

curve = TRUE)
CART_model_train_pr$auc.integral

## [1] 0.2974847

predicted_prob <- predict(CART_model2, trainX, type="prob")[, "retrasado"]
CART_model2_train_pr <- pr.curve(scores.class0 = predicted_prob,
                                 weights.class0 = trainy == "retrasado",
                                 curve = TRUE)
CART_model2_train_pr$auc.integral

```

```
## [1] 0.317803
```

A continuación, creamos un gráfico para comparar el AUC de cada modelo en los conjuntos **train** y **test**.

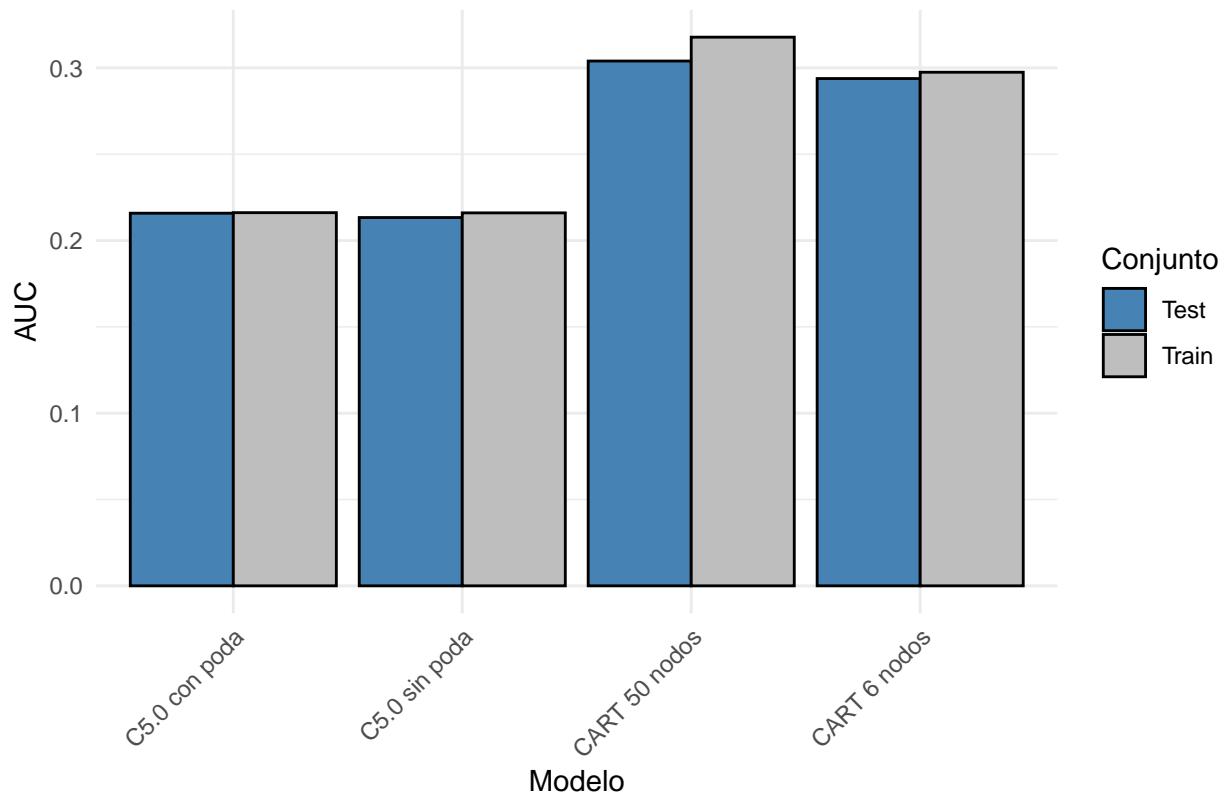
```

# Creamos un dataframe con todos los valores AUC
auc_data <- data.frame(
  Model = rep(c("C5.0 con poda", "C5.0 sin poda", "CART 6 nodos", "CART 50 nodos"), 2),
  AUC = c(C50_model1_pr$auc.integral, C50_model2_pr$auc.integral,
          CART_model1_pr$auc.integral, CART_model2_pr$auc.integral,
          C50_model1_train_pr$auc.integral, C50_model2_train_pr$auc.integral,
          CART_model1_train_pr$auc.integral, CART_model2_train_pr$auc.integral),
  Conjunto = c(rep("Test", 4), rep("Train", 4))
)

# Creamos un gráfico de barras y mostramos por pantalla
ggplot(auc_data, aes(x = Model, y = AUC, fill = Conjunto)) +
  geom_bar(stat = "identity", position = position_dodge(), color="black") +
  theme_minimal() +
  labs(title = "Comparación de AUCs para Modelos de Train y Test",
       x = "Modelo",
       y = "AUC") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_manual(values = c("steelblue", "grey"))

```

Comparación de AUCs para Modelos de Train y Test



El AUC de cada modelo presenta similitudes en los conjuntos de **train** y **test**, excepto en el caso del modelo CART con 50 nodos, donde el conjunto **train** muestra un AUC más elevado. Esto sugiere la existencia de sobreajuste (*overfitting*) en este modelo, y que el número óptimo de nodos podría estar en un punto intermedio (superior a 6 nodos, pero inferior a 50).

En cuanto a la comprensibilidad del resultado, para los modelos C5.0 con poda y CART con 6 nodos, los resultados son relativamente fáciles de interpretar, aunque algunas reglas son extensas debido a la variedad de aeropuertos en las variables **ORIGEN** y **DEST**. Cada regla nos ayuda a entender las condiciones asociadas con una mayor o menor probabilidad de retraso.

Además, las reglas son coherentes con los patrones observados en el análisis exploratorio realizado en la primera parte de la práctica. Por ejemplo, en la PR1, notamos que el porcentaje de vuelos retrasados tiende a incrementarse a lo largo del día, lo cual concuerda con la primera regla de cada modelo.

Se comentan las conclusiones.

Los AUC de todos los modelos mostraron una capacidad predictiva superior a la del azar, siendo de 18.64 %, pero esta capacidad era limitada. Al imponer un requisito de precisión mínima del 50 %, ninguno de los modelos logró detectar más del 10 % de los casos positivos.

En comparación con los modelos CART, los modelos C5.0 demostraron una capacidad predictiva relativamente inferior, especialmente a niveles más altos de recall.

De tener que recomendar un único modelo entre los cuatro analizados, elegiríamos el modelo CART con 50 nodos. Este modelo no solo presentó el AUC más alto, sino también el mayor recall, alcanzando un 7.43 %, manteniendo un mínimo del 50 % de precisión. Sin embargo, la cantidad de reglas podría dificultar su interpretación. Para ampliar el estudio, sería ideal experimentar con distintos niveles de nodos para determinar si es posible simplificar el modelo sin reducir su capacidad predictiva.

Ejercicio 6

Se prueba con una variación u otro enfoque algorítmico.

A continuación, creamos un modelo *Random Forest*.

Este modelo tiene la limitación de no poder gestionar variables explicativas con más de 53 categorías. Dado que las variables `ORIGIN` y `DEST` comprenden 340 categorías distintas (los aeropuertos), optamos por seleccionar únicamente los 52 aeropuertos más comunes. Los demás aeropuertos los agrupamos en una categoría denominada «`Other`», logrando así un total de 53 categorías.

```
# Creamos una función para reemplazar los aeropuertos con menos vuelos con «Other»
replace_with_other <- function(data, column_name) {
  # Convertimos la columna especificada en caracteres
  data[[column_name]] <- as.character(data[[column_name]])
  # Obtenemos los 52 aeropuertos más comunes
  top_airports <- names(sort(table(data[[column_name]]), decreasing = TRUE)[1:52])
  # Reemplazamos los aeropuertos no comunes con «Other»
  data[[column_name]] <- ifelse(data[[column_name]] %in% top_airports, data[[column_name]], 'Other')
  # Volvemos a convertir la columna modificada en un factor
  data[[column_name]] <- as.factor(data[[column_name]])
  return(data)
}

# Aplicamos la función a las columnas 'ORIGIN' y 'DEST'
flightData_simplified <- flightData_reduced
flightData_simplified <- replace_with_other(flightData_simplified, 'ORIGIN')
flightData_simplified <- replace_with_other(flightData_simplified, 'DEST')

# Mostramos la estructura del conjunto de datos simplificado
str(flightData_simplified)
```

```
## 'data.frame': 560887 obs. of 8 variables:
## $ DAY_OF_WEEK : Factor w/ 7 levels "lunes","martes",...: 1 1 1 1 1 1 1 1 1 ...
## $ OP_UNIQUE_CARRIER : Factor w/ 15 levels "Alaska Airlines",...: 5 5 5 5 5 5 5 5 5 ...
## $ ORIGIN : Factor w/ 53 levels "ATL","AUS","BNA",...: 38 38 38 38 38 38 38 38 38 38 ...
## $ DEST : Factor w/ 53 levels "ATL","AUS","BNA",...: 1 1 1 1 1 1 1 1 15 27 ...
## $ CRS_DEP_TIME_CONTINUOUS: int 414 762 1047 455 845 370 980 615 765 426 ...
## $ CRS_ARR_TIME_CONTINUOUS: int 540 890 1178 528 912 539 1146 683 868 483 ...
## $ CRS_ELAPSED_TIME : int 126 128 131 73 67 109 106 68 103 57 ...
## $ ARR_DEL15 : Factor w/ 2 levels "a tiempo","retrasado": 1 1 1 1 1 1 1 1 1 ...
```

Además, debido a las restricciones de nuestro hardware, ajustar el modelo *Random Forest* con el 80 % de los datos (448709 observaciones) tardaría demasiado tiempo. Por lo tanto, en este análisis, utilizamos una división de datos del 20 % para el entrenamiento (112178 observaciones) y 80 % para el test.

```
set.seed(123)
split_prop <- 5
indexes <- sample(1:nrow(flightData_simplified),
                  size=floor(((split_prop-1) / split_prop) * nrow(flightData_simplified)))
```

```
train2 <- flightData_simplified[-indexes,]
test2 <- flightData_simplified[indexes,]
```

Como antes, X contiene todas las variables independientes, que son 7 en total. y representa nuestra variable dependiente, ARR_DEL15.

```
trainX2 <- dplyr::select(train2, -ARR_DEL15)
trainy2 <- train2$ARR_DEL15
testX2 <- dplyr::select(test2, -ARR_DEL15)
testy2 <- test2$ARR_DEL15
```

Volvemos a comprobar la estructura y las proporciones.

```
str(trainX2)
```

```
## 'data.frame': 112178 obs. of 7 variables:
## $ DAY_OF_WEEK : Factor w/ 7 levels "lunes","martes",...: 1 1 1 1 1 1 1 1 1 1 ...
## $ OP_UNIQUE_CARRIER : Factor w/ 15 levels "Alaska Airlines",...: 5 5 5 5 5 5 5 5 5 5 ...
## $ ORIGIN : Factor w/ 53 levels "ATL","AUS","BNA",...: 38 38 38 38 1 1 1 1 1 1 ...
## $ DEST : Factor w/ 53 levels "ATL","AUS","BNA",...: 1 1 1 1 38 38 38 38 38 38 ...
## $ CRS_DEP_TIME_CONTINUOUS: int 414 762 845 980 590 872 737 505 1268 550 ...
## $ CRS_ARR_TIME_CONTINUOUS: int 540 890 912 1146 716 1001 796 565 1359 552 ...
## $ CRS_ELAPSED_TIME : int 126 128 67 106 126 129 59 60 91 62 ...
```

```
str(testX2)
```

```
## 'data.frame': 448709 obs. of 7 variables:
## $ DAY_OF_WEEK : Factor w/ 7 levels "lunes","martes",...: 3 2 2 4 5 3 7 5 3 4 ...
## $ OP_UNIQUE_CARRIER : Factor w/ 15 levels "Alaska Airlines",...: 3 5 14 3 3 4 15 11 15 12 ...
## $ ORIGIN : Factor w/ 53 levels "ATL","AUS","BNA",...: 2 27 17 7 14 38 29 12 38 38 ...
## $ DEST : Factor w/ 53 levels "ATL","AUS","BNA",...: 8 7 38 14 44 33 16 38 37 13 ...
## $ CRS_DEP_TIME_CONTINUOUS: int 475 539 1004 420 812 405 1035 600 360 304 ...
## $ CRS_ARR_TIME_CONTINUOUS: int 702 652 1141 539 873 483 1205 724 470 521 ...
## $ CRS_ELAPSED_TIME : int 167 113 137 179 181 138 170 124 170 157 ...
```

```
prop.table(table((trainy2)))
```

```
##
## a tiempo retrasado
## 0.8135463 0.1864537
```

```
prop.table(table((testy2)))
```

```
##
## a tiempo retrasado
## 0.8132799 0.1867201
```

A continuación, procedemos con el ajuste del modelo *Random Forest*.

```

library(randomForest)
rf_model <- randomForest(trainX2, trainy2)
rf_model

## 
## Call:
##   randomForest(x = trainX2, y = trainy2)
##   Type of random forest: classification
##   Number of trees: 500
##   No. of variables tried at each split: 2
##
##       OOB estimate of error rate: 22.09%
## Confusion matrix:
##             a tiempo retrasado class.error
## a tiempo     83424      7838  0.0858846
## retrasado    16943      3973  0.8100497

```

Aunque no es posible visualizar las reglas de un modelo *Random Forest* tan fácilmente como en un único árbol de decisión, podemos recurrir a la métrica `MeanDecreaseGini` para evaluar la contribución de cada variable a la homogeneidad de los nodos y los resultados en un árbol de decisión dentro del conjunto.

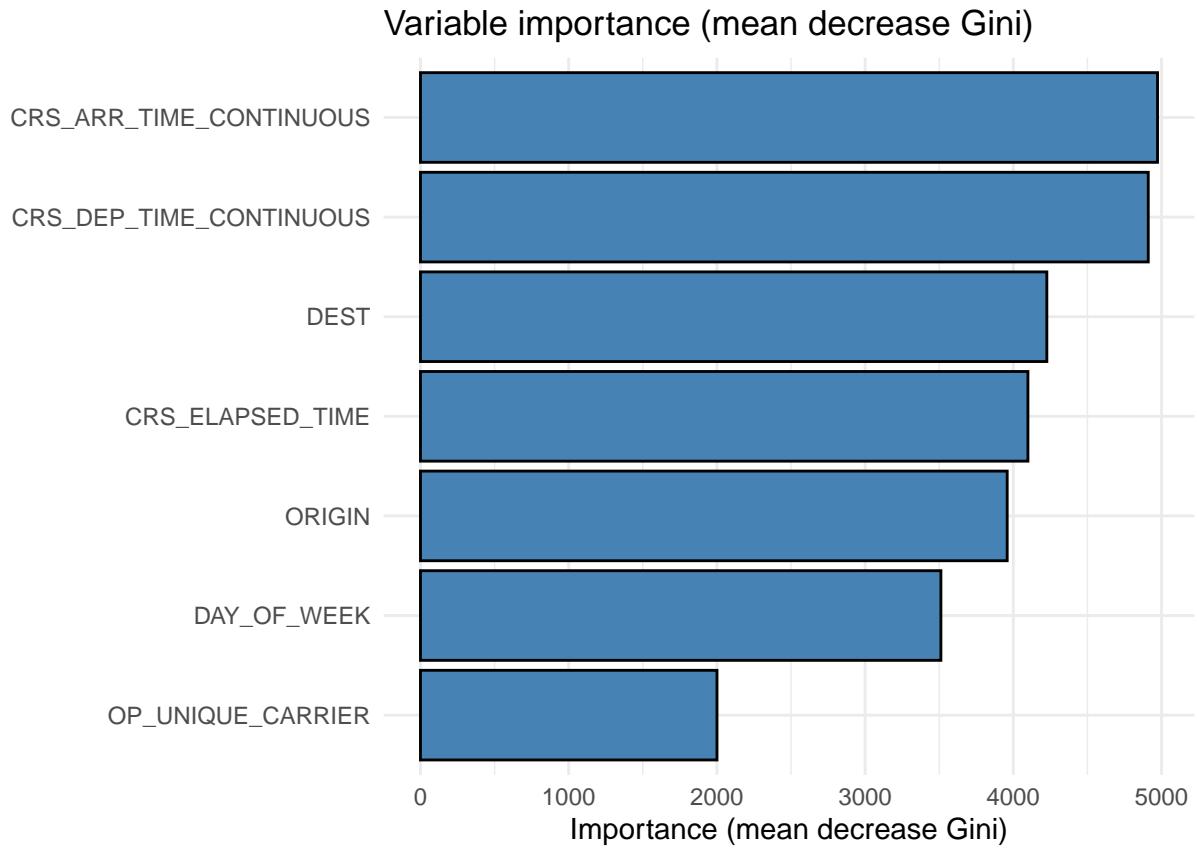
```

mean_decrease_gini <- as.data.frame(importance(rf_model))
mean_decrease_gini

##                                     MeanDecreaseGini
## DAY_OF_WEEK                      3511.421
## OP_UNIQUE_CARRIER                  2000.613
## ORIGIN                           3958.667
## DEST                             4226.197
## CRS_DEP_TIME_CONTINUOUS          4910.422
## CRS_ARR_TIME_CONTINUOUS          4973.070
## CRS_ELAPSED_TIME                   4098.676

ggplot(mean_decrease_gini, aes(x = reorder(rownames(mean_decrease_gini), MeanDecreaseGini),
                                y = MeanDecreaseGini)) +
  geom_bar(stat = "identity", fill = "steelblue", color = "black") +
  coord_flip() +
  labs(x = "",
       y = "Importance (mean decrease Gini)",
       title = "Variable importance (mean decrease Gini)") +
  theme_minimal()

```



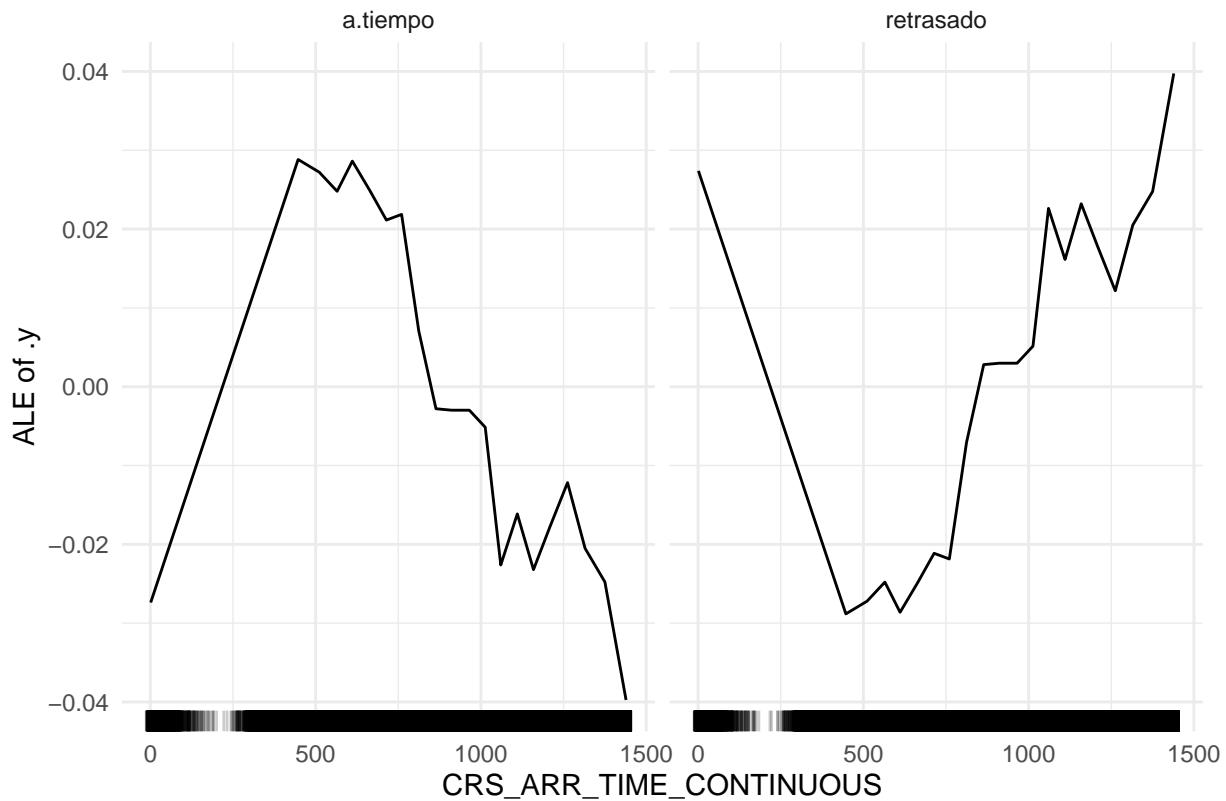
Notamos que las variables más relevantes en el modelo son la hora programada de llegada `CRS_ARR_TIME_CONTINUOUS` y la hora de salida `CRS_DEP_TIME_CONTINUOUS`.

Además, es posible calcular los efectos de las variables (*features*):

```
library(iml)

set.seed(123)
predictor <- Predictor$new(rf_model, data = trainX2, y = trainy2)
effs <- FeatureEffects$new(predictor)

plot(effs, features = "CRS_ARR_TIME_CONTINUOUS") +
  theme_minimal()
```



En el gráfico, se observa cómo la hora programada de llegada influye en las decisiones del modelo. Los vuelos con hora programada de llegada por la mañana, especialmente alrededor de las 8:00–10:00 (480–600 minutos desde medianoche), presentan la mayor probabilidad de llegar a tiempo. La posibilidad de retrasos incrementa progresivamente a lo largo del día, alcanzando su punto máximo cerca de la medianoche. Esta tendencia es coherente con los hallazgos del análisis exploratorio realizado en la PR1.

Nota: Me habría gustado presentar un gráfico para todas las 7 variables, pero el código tarda demasiado en ejecutarse (10 minutos para un solo gráfico).

Se detalla, comenta y evalúa la calidad de clasificación.

Para el conjunto `train`, podemos ver la matriz de confusión directamente en la salida del modelo.

```
rf_model
```

```
##
## Call:
##   randomForest(x = trainX2, y = trainy2)
##             Type of random forest: classification
##                       Number of trees: 500
## No. of variables tried at each split: 2
##
##           OOB estimate of error rate: 22.09%
## Confusion matrix:
##   a tiempo retrasado class.error
## a tiempo     83424      7838  0.0858846
```

```
## retrasado     16943      3973   0.8100497
```

La tasa de error es igual a 22.09 %, o lo que es lo mismo, una exactitud de 77.81 %.

A continuación, calculamos la matriz de confusión para el conjunto test.

```
predicted <- predict(rf_model, testX2, type="class")
caret::confusionMatrix(data = predicted, reference = testy2, positive = "retrasado")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction a tiempo retrasado
##   a tiempo     333891     67466
##   retrasado     31035     16317
##
##                   Accuracy : 0.7805
##                           95% CI : (0.7793, 0.7817)
##   No Information Rate : 0.8133
##   P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1318
##
##   Mcnemar's Test P-Value : <2e-16
##
##                   Sensitivity : 0.19475
##                   Specificity  : 0.91496
##   Pos Pred Value  : 0.34459
##   Neg Pred Value : 0.83191
##   Prevalence     : 0.18672
##   Detection Rate : 0.03636
##   Detection Prevalence : 0.10553
##   Balanced Accuracy : 0.55485
##
##   'Positive' Class : retrasado
##
```

Observamos que la exactitud (*Accuracy*) es igual a 78.05 %, correspondiendo casi perfectamente a la estimación del conjunto de test, y señalando que no hay sobreajuste en el modelo.

La sensibilidad (*Sensitivity*) es igual a 19.48 %, mientras que la precisión (*Pos Pred Value*) es igual a 34.46 %.

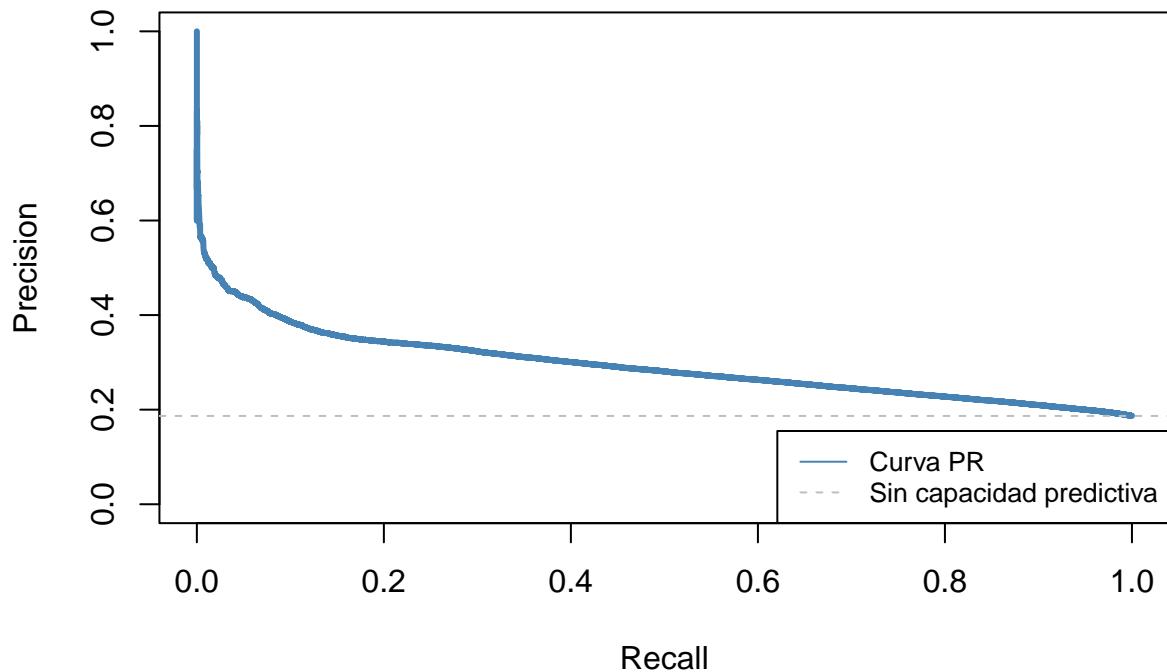
A continuación, calculamos la curva PR y el AUC para el modelo *Random Forest*.

```
predicted_prob <- predict(rf_model, testX2, type="prob")[, "retrasado"]

rf_model_pr <- pr.curve(scores.class0 = predicted_prob, weights.class0 = testy2 == "retrasado", curve = 0)

plot_pr_curve(rf_model_pr)
```

PR curve
AUC = 0.292807



La curva PR tiene una forma similar a las anteriores. Igual que el modelo C5.0 sin poda, la precisión llega hasta el 100 % para los niveles más bajos de recall, pero disminuye muy rápidamente a medida que aumenta el recall. El AUC llega a 0.293, parecido al AUC del modelo CART con 6 nodos.

Si mantenemos el mismo estándar previo y optamos por no aceptar una precisión por debajo del 50 %, entonces:

```
find_best_recall(rf_model_pr$curve, min_precision = 0.50)
```

| Recall | Precision | Threshold |
|--------|-----------|-----------|
| 0.0177 | 0.5 | 0.842 |

Lamentablemente, el modelo *Random Forest* solo logra un recall del 1.77 % en este caso.

Se comparan los resultados de manera exhaustiva con el método supervisado del ejercicio 5.

Empezamos con una evaluación general de los valores AUC de los distintos modelos.

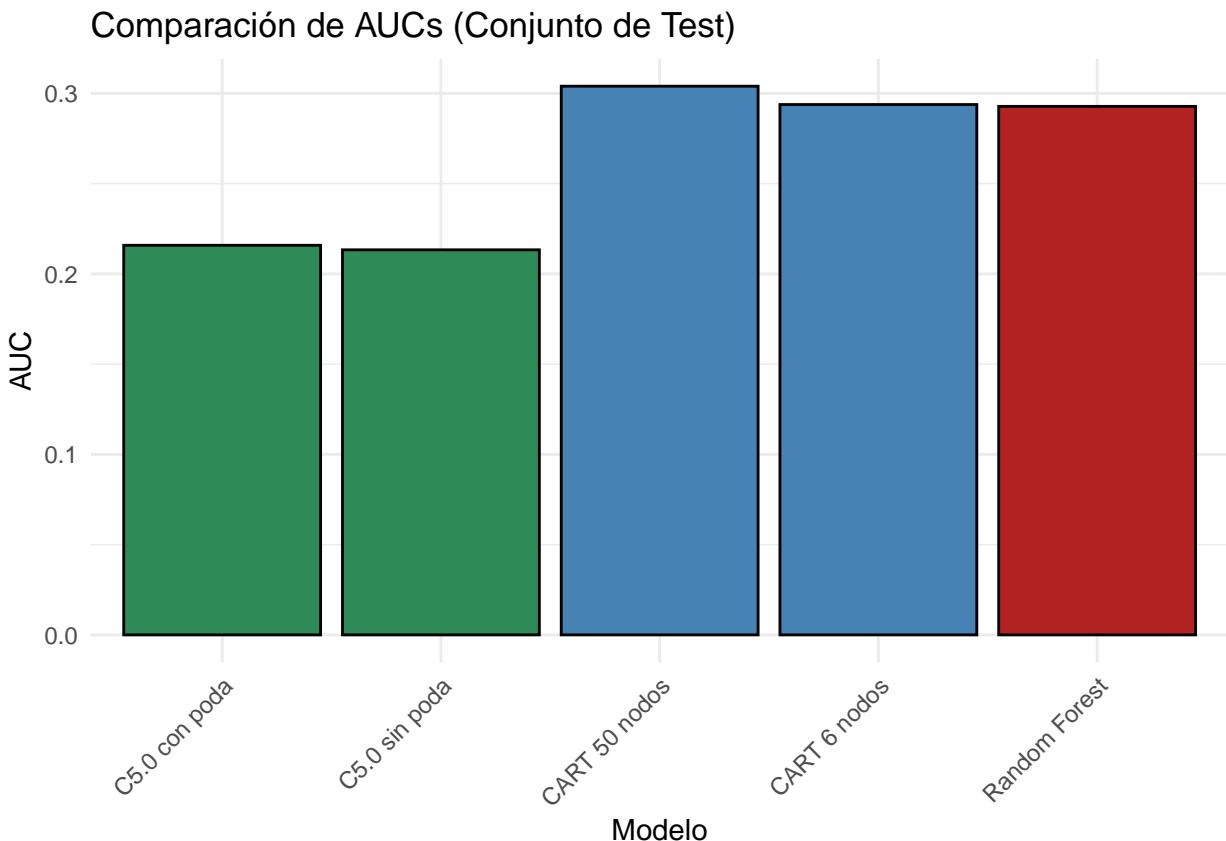
```
# Creamos un dataframe con todos los valores AUC
auc_data <- data.frame(
  Model = c("C5.0 con poda", "C5.0 sin poda", "CART 6 nodos", "CART 50 nodos", "Random Forest"),
  AUC = c(C50_model_pr$auc.integral, C50_model2_pr$auc.integral,
```

```

        CART_model_pr$auc.integral, CART_model2_pr$auc.integral, rf_model_pr$auc.integral)
)

# Generamos y visualizamos un gráfico de barras
ggplot(auc_data, aes(x = Model, y = AUC, fill = Model)) +
  geom_col(color="black") +
  theme_minimal() +
  labs(title = "Comparación de AUCs (Conjunto de Test)",
       x = "Modelo",
       y = "AUC") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none") +
  scale_fill_manual(values = c("seagreen", "seagreen", "steelblue", "steelblue", "firebrick"))

```



Notamos que el modelo CART con 50 nodos presenta el AUC más elevado, superando ligeramente al modelo CART con 6 nodos y el modelo *Random Forest*. En general, los modelos C5.0 tienen un rendimiento inferior.

A continuación, comparamos la capacidad de los modelos de detectar los vuelos retrasados con una precisión mínima del 50 %.

```

# Creamos un dataframe con todos los valores recall
recall_data <- data.frame(
  Model = c("C5.0 con poda", "C5.0 sin poda", "CART 6 nodos", "CART 50 nodos", "Random Forest"),
  Recall = c(0.0377, 0.0310, 0.0342, 0.0743, 0.0177)
)

# Generamos y visualizamos un gráfico de barras

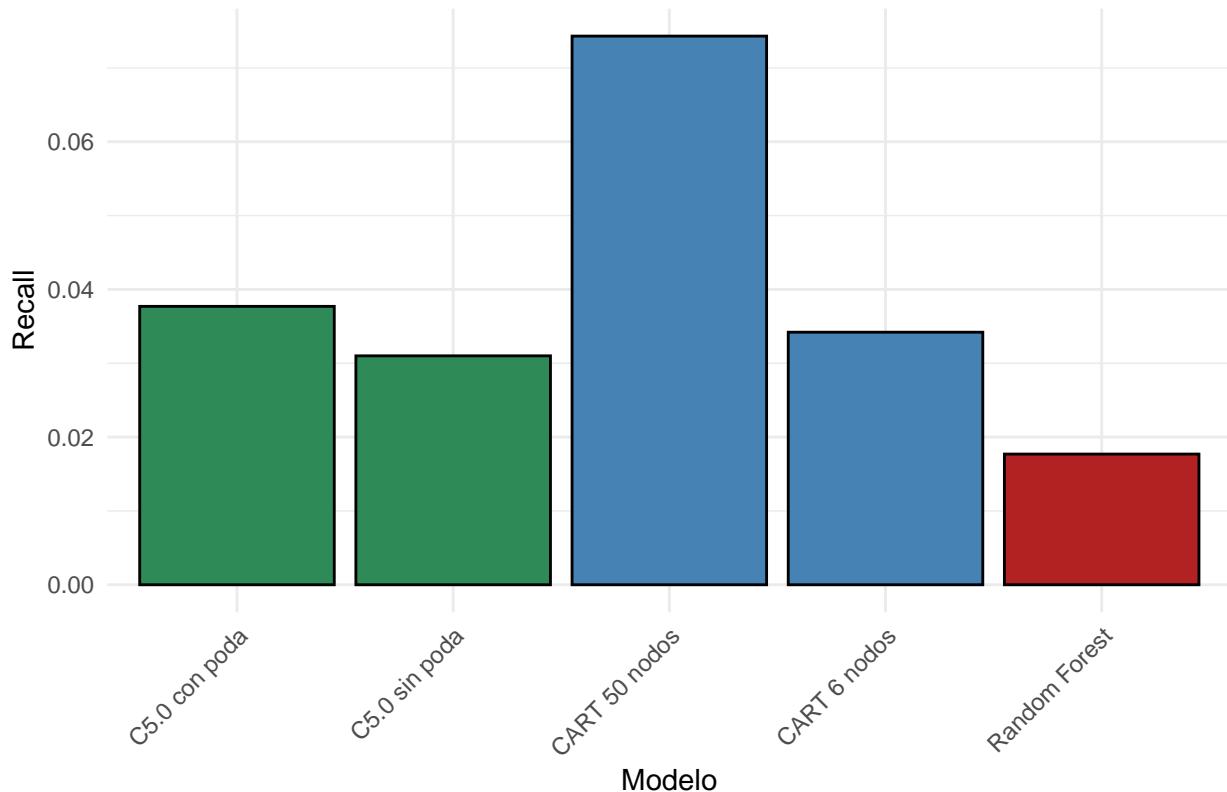
```

```

ggplot(recall_data, aes(x = Model, y = Recall, fill = Model)) +
  geom_col(color="black") +
  theme_minimal() +
  labs(title = "Comparación de recall con al menos 50 % de precisión (conjunto de test)",
       x = "Modelo",
       y = "Recall") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1), legend.position = "none") +
  scale_fill_manual(values = c("seagreen", "seagreen", "steelblue", "steelblue", "firebrick"))

```

Comparación de recall con al menos 50 % de precisión (conjunto de test)



Con el requisito de una precisión mínima de 50 %, el modelo CART con 50 nodos también destaca como el mejor modelo para detectar el mayor porcentaje de vuelos retrasados. No obstante, hay otros factores a considerar al momento de seleccionar un modelo adecuado.

Por ejemplo, los árboles de decisión nos permitieron incluir las variables `ORIGEN` y `DEST` con todas sus categorías, abarcando 340 aeropuertos. También ofrecieron gran flexibilidad en cuanto al tamaño de los conjuntos de entrenamiento y prueba. Utilizando el 80 % de los datos (448709 observaciones), los modelos basados en árboles de decisión (C5.0 y CART) se ajustaron en menos de un minuto.

La complejidad computacional del modelo *Random Forest* representó un desafío, especialmente con nuestros recursos limitados. El algoritmo nos obligó a reducir el número de categorías (aeropuertos) de 340 a 53, lo cual implicó la eliminación de información valiosa sobre aeropuertos más pequeños. Además, para evitar tiempos de ejecución prolongados, redujimos el tamaño del conjunto de entrenamiento de un 80 % a un 20 % de los datos. Es muy probable que este recorte haya tenido un impacto negativo en la capacidad predictiva del modelo.

Los modelos también difieren significativamente en términos de interpretabilidad. Mientras que los árboles de decisión se basan en reglas claras y visualizables, los *Random Forest* operan más como una «caja negra».

Aunque es posible analizar la importancia y el efecto de las variables, este análisis se realiza de manera generalizada, sin considerar las interacciones entre ellas.

Es crucial señalar que ninguno de los modelos logró un recall superior al 10 % manteniendo al menos un 50 % de precisión. Es posible que una organización esté dispuesta a tolerar un porcentaje más alto de falsos positivos para reducir el número de falsos negativos, aunque esto implicaría el desperdicio de más recursos.

Ejercicio 7

Se identifica qué posibles limitaciones tienen los datos que has seleccionado para obtener conclusiones con los modelos (supervisado y no supervisado)

Hemos obtenido nuestro conjunto de datos del Bureau of Transportation Statistics del Departamento de Transporte de Estados Unidos, una fuente reconocida por su alta fiabilidad. Los datos, en su mayoría, eran muy limpios. Sin embargo, una limitación importante es que el conjunto de datos solo incluye vuelos con origen y destino dentro de Estados Unidos. Esto significa que los modelos derivados no serían aplicables a vuelos internacionales o en otros países. Adicionalmente, debido a restricciones de hardware, nuestro análisis se centró únicamente en los vuelos del último mes disponible, septiembre de 2023. A pesar de limitarnos a un solo mes, contábamos con una considerable cantidad de observaciones: 569,338 vuelos.

En cuanto a los modelos no supervisados, optamos por seleccionar una muestra aleatoria de 10,000 vuelos para reducir el tiempo de ejecución del código y facilitar la visualización e interpretación de los gráficos. Sin embargo, esta decisión podría haber tenido efectos no deseados. Dentro del conjunto de datos completo, había vuelos excepcionalmente largos pero poco frecuentes, como aquellos que van desde Estados Unidos continental hasta Hawái. Si hubiéramos utilizado el conjunto de datos completo, es posible que el algoritmo DBSCAN habría agrupado estos vuelos muy largos en un clúster en vez de identificarlos como ruido o valores atípicos.

Para los modelos supervisados, hay que considerar que podrían existir factores temporales que influyeran en las conclusiones. Un ejemplo claro se observó con los vuelos de JetBlue, los cuales mostraban una mayor probabilidad de retraso. Este patrón podría deberse a problemas específicos de la aerolínea durante septiembre que ya han sido resueltos. Para aumentar la generalización de los modelos, sería ideal incluir datos de un periodo más extenso. Además, el modelo *Random Forest* requirió una reducción en el número de categorías (aeropuertos) de 340 a 53, lo que podría introducir un sesgo hacia los aeropuertos más grandes.

Se identifican posibles riesgos del uso del modelo.

El riesgo de generalización inadecuada es significativo, ya que los modelos se basan en datos de vuelos domésticos en Estados Unidos durante un mes específico. Esta limitación podría afectar su eficacia en otros contextos, como vuelos internacionales o diferentes épocas del año, donde los patrones de retraso varían por estaciones, diferencias regionales o eventos particulares.

Otro riesgo es el cambio en las operaciones de las aerolíneas a lo largo del tiempo. Las modificaciones operativas pueden volver obsoletos los modelos basados en datos históricos, a menos que se actualicen periódicamente para reflejar estos cambios.

El desbalance de clases y el sesgo predictivo también son preocupaciones importantes. Los modelos presentan un desbalance, con una mayoría de vuelos clasificados como «a tiempo». Esto puede llevar a un sesgo hacia la clase mayoritaria y a una alta tasa de falsos negativos, lo que dificulta la planificación y la toma de decisiones al fallar en identificar vuelos retrasados eficientemente.

Los modelos de *Random Forest*, aunque útiles, presentan desafíos debido a su naturaleza de «caja negra», complicando la justificación de las decisiones basadas en sus predicciones en escenarios que requieren explicaciones detalladas. Los árboles de decisión con muchos nodos también pueden ser difíciles de interpretar debido a su complejidad creciente.

Elegir un umbral de decisión óptimo es otro desafío. Un porcentaje alto de falsos positivos puede llevar a una asignación ineficiente de recursos o a medidas preventivas innecesarias. En contraste, una alta tasa de falsos negativos puede resultar en una preparación insuficiente ante retrasos reales.

Finalmente, la dependencia exclusiva en estos modelos para la toma de decisiones es problemática, ya que no capturan completamente la complejidad del entorno operativo de las aerolíneas. Hay múltiples factores, como las condiciones meteorológicas adversas, que pueden influir en los retrasos y no están incluidos en el modelo.

La toma de decisiones debe ser equilibrada, teniendo en cuenta tanto los resultados del modelo como otros factores operativos y contextuales relevantes.
