



QueenCard

UNIVERSIDAD NACIONAL AUTÓNOMA DE
MÉXICO

FACULTAD DE CIENCIAS

ALMACENES Y MINERÍA DE DATOS

(2024-1)

PROYECTO FINAL



- Barón Herrera Victoria Elena 315650383
- Brito Juárez Britny 421074668
- Illescas Coria Janet 318219309
- Vargas Bravo Paola 318074755

FECHA : 06 DE DICIEMBRE 2023

Contents

| | | |
|----|---|----|
| 1 | Introducción | 3 |
| 2 | Desarrollo | 4 |
| 3 | Conclusiones Análisis exploratorio | 11 |
| 4 | Conclusiones Preprocesamiento de datos | 19 |
| 5 | Conclusiones Clasificación | 20 |
| 6 | Conclusiones Evaluación de modelos de clasificación | 22 |
| 7 | Conclusiones Reglas de Asociación | 24 |
| 8 | Conclusiones Agrupación | 26 |
| 9 | Conclusiones Finales | 29 |
| 1. | | |

1 Introducción

Este proyecto de minería de datos se enfoca en entender mejor el terrorismo mediante el análisis del conjunto de datos "Global Terrorism Database" (GTD). Con información que abarca desde 1970 hasta 2017, este conjunto de datos es clave para descubrir patrones importantes que ayudarán en la seguridad nacional y la toma de decisiones para combatir el terrorismo.

Además de aplicar cada una de las técnicas vistas en el curso, para probar de esta manera nuestros conocimientos en este proyecto de minería de datos.

Por lo que como sabemos, usaremos la metodología CRISP (Cross-Industry Standard Process for Data Mining) para guiar nuestro análisis, desde entender los datos hasta evaluar modelos. Queremos aplicar un enfoque práctico basado en lo aprendido durante el curso.

- **Descripción de Datos:**

Empezamos mirando los datos para entender su estructura. Queremos saber de qué tipo son los atributos, cuántos valores faltan y qué tendencias iniciales podemos encontrar.

- **Preprocesamiento de Datos:**

Luego, prepararemos los datos para el análisis. Decidiremos qué atributos usar, cómo manejar valores que faltan y si hay datos extraños. Queremos asegurarnos de que nuestros datos estén en buena forma antes de aplicar modelos.

- **Minado de Datos:**

Probaremos diferentes técnicas para clasificar, como árboles de decisión y redes neuronales. Ajustaremos los parámetros para ver qué patrones emergen y cuán efectivos son nuestros modelos.

- **Asociación y Agrupación:**

Exploraremos si podemos encontrar reglas interesantes en los datos y si hay grupos de eventos similares. Queremos ver si hay conexiones o patrones más allá de lo evidente.

Este enfoque práctico nos permitirá poner en acción técnicas avanzadas de minería de datos y proponer medidas concretas para abordar la problemática del terrorismo.

2 Desarrollo

(a) Conocimiento de los datos (análisis exploratorio)

- Tipo de atributo (nominal, ordinal, numérico, etc.).

Para determinar el tipo de cada atributo en nuestro conjunto de datos, utilizamos la función `class()` en R. Esta función nos proporciona información sobre la clase a la que pertenece cada objeto. En cada iteración sobre el conjunto de datos, aplicamos esta función a cada columna, asignando su tipo correspondiente a la celda de la columna dos para cada atributo.

```
for (i in 1:num_filas) {  
    atributo = gtd_data[[nombres_gtd_data[i]]]  
  
    # Col 1: Nombre del atributo  
    info_atributos[i, 1] = nombres_gtd_data[i]  
  
    # Col 2: Tipo de atributo (nominal, ordinal, numérico, etc.).  
    info_atributos[i, 2] = class(atributo)}
```

- Valores permitidos (si aplica).

Obtenemos de forma general los valores permitidos de cada variable seleccionando sus los valores únicos y omitiendo los valores nulos, que no son de nuestro interés.

Para su representación únicamente consideraremos aquellas variables con menos de 500 valores diferentes, las que no cumplan con esto serán representadas con NA. En caso de ser numéricos mostraremos el rango y en caso de ser cadenas simplemente separamos los valores con comas.

```
99  
100    allowed_values <- vector( length=length(names(gtd_data)))  
101  
102    j <- 1  
103    for(i in gtd_data) {  
104        # Obtenemos sus valores únicos  
105        unique_values <- unique(i)  
106        # Omitimos valores nulos  
107        unique_values <- na.omit(unique_values)  
108  
109        # Verificamos que no sean más de 500 valores diferentes para considerarlos  
110        # valores permitidos  
111        if (length(unique_values) < 500) {  
112            if (is.numeric(i)){  
113                allowed_values[j] = paste(min(unique_values), "-", max(unique_values))  
114            } else {  
115                allowed_values[j] = paste(unique_values, collapse = ", ")  
116            }  
117        } else {  
118            allowed_values[j] = "NA"  
119        }  
120        j <- j+1  
121    }  
122  
123    #-----  
124    # Mostrar los valores permitidos para cada columna  
125    print(allowed_values)  
126
```

- Porcentaje de valores perdidos.

```
Valores_Perdidos = sum(is.na(atributo)) / length(atributo) * 100
```

La fórmula Valores_Perdidos evalúa el porcentaje de valores faltantes en un atributo al identificar la cantidad de valores perdidos (NA), dividiéndolos por el número total de observaciones en el atributo y multiplicando el resultado por 100. Este cálculo proporciona una medida que ayuda a entender la proporción de información ausente con respecto al conjunto de datos completo, siendo útil para evaluar la integridad y completitud de los datos.

- Valor mínimo, máximo, media, desviación estándar (si aplica).

Para obtener estos datos, asignamos a las columnas 5,6,7 y 8 de la tabla las etiquetas "Min", "Max", "Mean", "DevEstandar" respectivamente, para colocar ahí los valores correspondientes de los atributos numéricos. Para ello, utilizamos las funciones min(), max(), mean() y sd() en sus respectivas columnas. Para saber si era posible obtener esos valores para el atributo en cuestión (esto depende de las iteraciones del ciclo for), utilizamos la función booleana is.numeric(). Si es verdadero, obtiene los valores, en caso contrario, omite los cálculos.

```
189      # Para estadísticas:
190      numeric <- is.numeric(atributo)
191      # Col 5: Valor mínimo (si aplica)
192      if (numeric) {
193          info_atributos[i, 5] = min(atributo, na.rm = TRUE)
194      # Col 6: máximo (si aplica)
195
196          info_atributos[i, 6] = max(atributo, na.rm = TRUE)
197      # Col 7: media (si aplica)
198
199          info_atributos[i, 7] = mean(atributo, na.rm = TRUE)
200
201      # Col 8: desviación estándar (si aplica).
202          info_atributos[i, 8] = sd(atributo, na.rm = TRUE)
203      }
204
```

- Si es numérico, indicar el tipo de distribución que parece seguir (p.e. normal).

En este caso, y como se muestra en la siguiente imagen, iteramos sobre las columnas del data set (línea 23). Así para todas las columnas de clase *numeric* (línea 24) lo que se hizo fue crear su respectivo histograma (línea 28). Después procedimos, basándonos en el histograma obtenido, a seleccionar la distribución cuya gráfica se pareciera más al histograma. Lo anterior para cada variable (líneas 34-43).

```

18  # Vector donde almacenaremos la distribución de las variables categóricas o NA
19  # Vara las que no aplique
20  distribucion <- vector( length=length(names(gtd_data)))
21  cols_names <- colnames(gtd_data)
22  j <- 1
23  for(i in gtd_data) {
24      if (class(i) == "numeric"){
25          # Para saber en qué posición guardar la distribución que observemos
26          print(j)
27          # Gráficamente histogramas de variables categóricas para aproximar distribución
28          hist((i), main = paste("Histograma de ", cols_names[j]), xlab = cols_names[j])
29      }else{
30          distribucion[j] = "NA"
31      }
32      j <-j+1
33  }
34  distribucion[1] = "normal sesgada a la izquierda"
35  distribucion[14] = "normal"
36  distribucion[15] = "normal sesgada a la izquierda"
37  distribucion[71] = "lognormal"
38  distribucion[102] = "lognormal"
39  distribucion[108] = "lognormal"
40  distribucion[113] = "lognormal"
41  distribucion[118] = "lognormal"
42  distribucion[119] = "lognormal"
43  distribucion[120] = "lognormal"

```

- Si es categórico, los niveles y frecuencia de cada uno.

Para el caso si tenemos atributos que son categóricos, por lo que en un inicio cuando se trato obtener sus niveles y frecuencias obteníamos datos erróneos como NA, dado que aunque eran categóricos no eran considerado ninguno como factor.

```

### Análisis de Categorías como factores
# Verificar cuáles atributos son factores
sapply(gtd_data, function(x) is.factor(x))
# Obtener variables numéricas
cols_numeric <- colnames(gtd_data[, sapply(gtd_data, is.numeric)])
numeric_gtd_data <- gtd_data[, cols_numeric]
# Ver las variables categóricas
resumen_categoricas <- summary(gtd_data)
# Identificar las variables categóricas
variables_categoricas <- sapply(gtd_data, function(columna) is.factor(columna) | is.character(columna))
# Mostrar las variables categóricas
nombres_categoricos <- names(variables_categoricas[variables_categoricas])
print(nombres_categoricos)
# Clase de las variables
sapply(gtd_data, class)
# Verificamos cuáles columnas son factores
factores <- sapply(gtd_data, is.factor)
columnas_factores <- names(factores[factores])

```

Por lo que tenemos que hacer un análisis nosotras para considerar cuales de los mismos son candidatos a factores, que es el siguiente programa :

```

# Obtener las columnas de caracteres
columnas_caracter <- sapply(gtd_data, is.character)

# Inicializar un vector para almacenar las columnas que podrían ser factores
columnas_factores <- character()

# Definir un umbral para el número máximo de niveles permitidos
umbral_niveles <- 10

# Iterar sobre las columnas de caracteres
for (col in names(gtd_data[columnas_caracter])) {
  # Contar los niveles únicos en la columna
  num_niveles <- length(unique(gtd_data[[col]]))

  # Verificar si el número de niveles es menor que el umbral
  if (num_niveles <= umbral_niveles) {
    columnas_factores <- c(columnas_factores, col)
  }
}

# Mostrar las columnas identificadas como factores
print(columnas_factores)

```

Usamos el umbral de 10 dado que se elige como una regla general para identificar variables categóricas con un número manejable de niveles.

Esta elección busca equilibrar la eficiencia en el almacenamiento y en el análisis con la practicidad y la interpretabilidad de las variables categóricas en el contexto del análisis de datos.

Dado que si el umbral se establece demasiado alto, podrías terminar tratando como factores variables que tienen un número excesivo de niveles. Los factores con un gran número de niveles pueden tener un impacto negativo en la eficiencia de almacenamiento y dificultar la interpretación.

```

#-----
# Ubicamos las columnas que se convierten a factores
gtd_data[columnas_factores] <- lapply(gtd_data[columnas_factores], as.factor)
#-----
```

Posteriormente se identifican las columnas en el conjunto de datos (gtd_data) que podrían representarse de manera eficiente como factores. Estas columnas se convierten explícitamente a factores usando lapply y as.factor().

```

# Calcular las frecuencias y los niveles
Niveles = sapply(gtd_data, function(x) ifelse(is.factor(x), paste(levels(x), collapse = ", "), NA))
Frecuencia = sapply(gtd_data, function(x) ifelse(is.factor(x), table(x), NA))

```

Estos bloques de código calculan información resumida para cada columna en el conjunto de datos:

- **Niveles:** Para columnas que son factores, se obtienen los niveles únicos y se concatenan en una cadena. Si la columna no es un factor, se asigna NA.
- **Frecuencia:** Para columnas que son factores, se utiliza table() para calcular la frecuencia de cada nivel. Si la columna no es un factor, se asigna NA. Estos resúmenes son útiles para comprender la distribución y la variabilidad de las variables categóricas
- Indicar si el atributo presenta valores atípicos.

Para el análisis de valores atípicos en las variables numéricas primero observamos su respectivo diagrama de caja, pero para agregarlo a nuestra tabla final decidimos obtener los valores atípicos que nos da el mismo diagrama. Si se devolvieron valores atípicos de la variable se marcará con la cadena *yes* y en caso contrario con la cadena *no*.

```
127     cols_names <- colnames(gtd_data)
128
129     # Vector donde almacenaremos si una variable cuenta con valores atípicos
130     atypical <- vector( length=length(names(gtd_data)))
131
132     j <-1
133     for(i in gtd_data) {
134         if(class(i) == "numeric") {
135             # Graficamos el boxplot de las variables numéricas y lo guardamos
136             box <- boxplot(i, main = paste("Boxplot de ", cols_names[j]), xlab = cols_names[j], plot = FALSE)
137             # Obtenemos los valores que identifica como atípicos
138             valores_atípicos <- box$out
139
140             atypical[j] <- ifelse(length(valores_atípicos) > 0, "yes", "no")
141
142         } else {
143             atypical[j] <- "no"
144         }
145
146         j <-j+1
147     }
148     print(atypical)
```

PREGUNTAS

Para ello decidimos hacer una matriz de correlación, para de alguna forma darnos una idea de como se comportan las variables.

```
#-----  
#CORRELACIÓN (Preguntas)  
#-----  
  
#convertir factor y char a numeric  
cols_names <- colnames(gtd_data)  
j <- 1  
set.seed(1)  
for(i in gtd_data) {  
  n = nlevels(i)  
  if(class(i) == 'character'){  
    x <- as.factor(i)  
    gtd_data[cols_names[j]]<- as.numeric(x)  
  }  
  if(class(i) == 'factor'){  
    gtd_data[cols_names[j]]<- as.numeric(i) #as.factor(sample(i, n, replace=TRUE))  
  }  
  j <-j+1  
}  
str(gtd_data)  
correlation_matrix <- cor(gtd_data)  
  
print(correlation_matrix)  
#
```

El código realiza una serie de operaciones sobre un conjunto de datos. En primer lugar, convierte las variables categóricas de tipo "character" y "factor" a variables numéricas. Luego, calcula la matriz de correlación para todas las variables en el conjunto de datos. Posteriormente, selecciona un subconjunto específico de variables para un análisis más detallado y calcula la matriz de correlación para este subconjunto. Finalmente, identifica pares de variables con correlaciones fuertes, superiores a un umbral predefinido, en la matriz de correlación, mostrando las variables involucradas en estas correlaciones fuertes.

- Elección de variable objetivo.

Guiadas por el hecho de que se quiere utilizar los datos para obtener información que "permite prevenir ataques, tomar decisiones políticas informadas". Decidimos que lo principal es saber que es lo que hace que un ataque sea o no exitoso. Ya que al saber lo anterior se podrá saber a que elementos prestarles más atención. De manera que la variable objetivo será *success*. Si bien se podrían buscar otras variable objetivo, por motivos de tiempo y equipos disponibles, se decidió enfocar todos los recursos en una única variable.

- ¿cuáles atributos parecen estar más ligados a alguna de las variables objetivo del dataset?.

En este caso basándonos en los resultados basados al principio del código notamos que, en su estado inicial sin la realización completa de la limpieza de datos, para un umbral de 0.7 podemos notar que las variables que más influyen en nuestra variable objetivo *success* son, en orden de más a menos, targtype_txt, aproxdate y extended. Sin embargo, es importante señalar que después de llevar a cabo la limpieza de datos completa más adelante en el código, reconsideramos su inclusión y decidimos no incluir algunas en la lista final de variables influyentes. Es por eso que, si bien también podríamos agregar eventid a la lista lo cierto es que al ser un valor meramente arbitrario decidimos no contarlo como tal.

```

> print("Correlaciones con success:")
[1] "Correlaciones con success:"
> print(cor_success)
      eventid          iyear         imonth        iday    approxdate
-0.0829722552 -0.0829630140 -0.0028450418 -0.0118023942 -0.1098708797
  extended      resolution       country   country_txt      region
  0.0732331388  0.0310559786 -0.0378274368 -0.0213416084 -0.0309094260
  region_txt     provstate       city    latitude  longitude
-0.0263613092  0.0008519609 -0.0045521842           NA           NA
 specificity    vicinity      location   summary crit1
           NA -0.0022234312 -0.0483886146 -0.0549507185 -0.0107333046
 crit2          crit3      doubtterr alternative alternative_txt
-0.0134764551 -0.0136776470           NA           NA  0.0219485892
 multiple       success      suicide attacktype1 attacktype1_txt
           NA  1.0000000000 -0.0311550827  0.0484080266 -0.0026830780

```

```

      NA  0.0397027368           NA  0.0141838010 -0.0596371352
targtype1_txt targsubtype1 targsubtype1_txt corp1 target1
-0.0665930043           NA  0.1251060708           NA           NA
  natlty1 natlty1_txt targtype2 targtype2_txt targsubtype2
  NA  0.0006531566           NA  0.0374423415           NA
targsubtype2_txt corp2 target2 natlty2 natlty2_txt
  0.0384882958  0.0357857457  0.0284392863           NA  0.0303366138
 targtype3 targtype3_txt targsubtype3 targsubtype3_txt corp3
  NA  0.0177561139           NA  0.0169045575  0.0142632906
 target3 natlty3 natlty3_txt gname gsubname
  0.0144712855           NA  0.0149635194 -0.0452736236  0.0174496828
 gname2 gsubname2 gname3 gsubname3 motive
  0.0112896171  0.0043489425  0.0018918099  0.0033462411  0.0524824156
guncertain1 guncertain2 guncertain3 individual nperps
  NA           NA           NA -0.0138676100           NA

```

```

      nperpcap      claimed     claimmode claimmode_txt      claim2
  NA           NA           NA  0.0210319284           NA
claimmode2 claimmode2_txt      claim3   claimmode3 claimmode3_txt
  NA  0.0085266435           NA           NA -0.0005193985
 compclaim weaptype1 weaptype1_txt weapsubtype1 weapsubtype1_txt
  NA -0.0081782348  0.0415774003           NA -0.0318125919
 weaptype2 weaptype2_txt weapsubtype2 weapsubtype2_txt weaptype3
  NA  0.0360842686           NA  0.0168911340           NA
weaptype3_txt weapsubtype3 weapsubtype3_txt weaptype4 weaptype4_txt
  0.0172429439           NA  0.0104603227           NA  0.0008688119
 weapsubtype4 weapsubtype4_txt weapdetail nkill nkillus
  NA  0.0011383529  0.0142313700           NA           NA
 nkillter nwound nwoundus nwoundte property
  NA           NA           NA           NA -0.0368764029
 propextent propextent_txt propvalue propcomment ishostkid
  NA           NA           NA           NA           NA

```

257:1 # (Untitled) R

| Console | Terminal | Background Jobs | | |
|---|---------------|-----------------|--------------------|---------------|
| R 4.1.2 · ~/Documentos/SeptimoSemestre/MYAD/Practicas/PGIT6/Mineria_Practicas-Proyecto/ProyectoFinal/ | | | | |
| nhostkid | nhostkidus | nhours | ndays | divert |
| NA | NA | NA | NA | 0.0078499131 |
| kidhijcountry | ransom | ransomamt | ransomamtus | ransompaid |
| 0.0221050744 | NA | NA | NA | NA |
| ransompaidus | ransomnote | hostkidoutcome | hostkidoutcome_txt | nreleased |
| NA | 0.0151642297 | NA | 0.0807461592 | NA |
| addnotes | scite1 | scite2 | scite3 | dbsource |
| 0.0292797090 | -0.0177177462 | 0.0260178122 | 0.0323618644 | -0.0683329712 |
| INT_LOG | INT_IDEO | INT_MISC | INT_ANY | related |
| 0.0598753117 | 0.0591830010 | -0.0149064255 | 0.0382449803 | 0.0017117286 |

```
print("Correlaciones con city:")
```

- ¿cuáles atributos parecen estar menos ligados a la variable objetivo del dataset?.

```
Rage tuyverse required but is not installed. Install Don't Show Again
#Menos correlacionadas
umbral <- 0.4

# Encontrar pares de variables con correlación inferior al umbral
correlation_pairs <- which(abs(correlation_matrix) < umbral & correlation_matrix != 1, arr.ind = TRUE)

# Filtrar las variables únicas de los pares encontrados
variables_con_correlacion_baja <- unique(c(row.names(correlation_pairs), colnames(correlation_pairs)))

# Variables menos relacionadas con success
cor_success <- correlation_matrix["success", ]
```

El código establece un umbral de correlación del 0.4 (para verificar las que tiene una correlación débil como sabemos) y busca pares de variables en el conjunto de datos que tengan una correlación inferior a este umbral. Luego, identifica las variables únicas de estos pares con correlaciones bajas. El enfoque se centra en encontrar las variables menos relacionadas con la variable objetivo "success". La información específica sobre las correlaciones menos relevantes se almacena en la variable "cor_success". Este análisis ayuda a identificar aquellas variables que tienen una débil asociación lineal con la variable objetivo, lo que puede ser valioso para comprender la importancia de diversas características en el contexto del análisis de éxito en el conjunto de datos. Por lo tanto, las variables menos correlacionadas con "success" en tu conjunto de datos incluyen "approxdate", "extended", "resolution", "country", "region", "region_txt", "provstate", "city", "latitude", "longitude" y otras. Estas correlaciones son cercanas a cero, lo que indica una débil relación lineal con la variable objetivo "success".

- Atributos Correlacionados : Reducción de dimensiones.

Después de realizar el análisis anterior y de revisar el conjunto de datos, planeamos la forma en la que se llevaría el proceso de selección de atributos y decidimos que las estrategias que utilizaríamos para un análisis más profundo sería limpiar los datos de forma que pudieramos mejorar la calidad del conjunto de datos y así realizar conclusiones más precisas sobre los atributos que debemos conservar. Al finalizar al limpieza, decidimos que usar χ^2 sería adecuado para conservar los atributos que tengan una correlación significativa y que realmente tengan una influencia en la variable objetivo, que concluimos que será success. Esta decisión nos dejó con 49 atributos.

3 Conclusiones Análisis exploratorio

El conjunto de datos es de particular interés, ya que se vislumbra la necesidad de llevar a cabo una exhaustiva fase de preprocesamiento de datos. Se destaca la presencia significativa de valores atípicos, siendo la mayoría de las variables de tipo carácter o entero, con algunas de naturaleza numérica. En un primer vistazo, ninguna variable se presenta como un factor. La prevalencia de valores atípicos es notable, sin embargo, es más pronunciada la presencia de valores perdidos.

En términos de valores permitidos, se ha identificado un rango que abarca desde -9 hasta -1, considerando la posibilidad de tratar estos valores como NA y aplicar un tratamiento adecuado en la siguiente etapa del proyecto. Además, las distribuciones de las variables muestran, en su mayoría, sesgo. Este análisis preliminar sienta las bases para abordar de manera rigurosa la limpieza y preparación de datos en la fase subsiguiente del proyecto.

(b) Preprocesamiento de datos

- Selección de atributos. Selecciona los atributos que consideres apropiados para una tarea predictiva. Justifica tu respuesta.

Para esta parte del preprocesamiento de datos, empleamos distintos métodos para asegurar la calidad y utilidad de nuestro conjunto de datos. En primer lugar, identificamos y tratamos los valores vacíos, que incluían casos de "Unknown", espacios en blanco, y valores predeterminados como -9, -99, e incluso puntos. Estos fueron convertidos a valores NA para facilitar el manejo de datos nulos y mejorar la consistencia en nuestro análisis.

```
7 # Convertimos a NA los espacios en blanco:
8 library(dplyr)
9 gtd_data_limpieza <- gtd_data %>%
10   mutate_all(~ ifelse(. %in% c("", " ", "Unknown", -9,-99), NA, .))
11 str(gtd_data_limpieza)
12
```

Posteriormente, observamos que las columnas con terminación "_txt" tenían un equivalente numérico. Optamos por completar las columnas numéricas con sus respectivos valores de las columnas de texto y eliminar las columnas de texto asociadas. Este paso se llevó a cabo para preservar la mayor cantidad de atributos numéricos posible.

```
15 columnas_txt <- grep("_txt$", names(gtd_data_limpieza), value = TRUE)
16 print(columnas_txt)
17
18 quitar_terminacion_txt <- function(columna_txt) {
19   col_sin_txt <- sub("_txt$", "", columna_txt)
20   return(col_sin_txt)
21 }
22
23 # Crear una función para asociar las filas únicas y llenar valores faltantes
24 crear_asociacion_unica_y_llenar <- function(gtd_data, variable1, variable2) {
25
26   # Inicializar un diccionario vacío
27   asociacion_unica <- list()
28   index1 <- which(names(gtd_data) == variable1)
29   index2 <- which(names(gtd_data) == variable2)
30
31   gtd_data[, variable2] <- as.character(gtd_data[, variable2])
32   gtd_data[, variable2][gtd_data[, variable2] == "") <- NA
33   gtd_data[, variable2][gtd_data[, variable2] == " "] <- NA
34 }
```

```
35 # Iterar sobre las filas del data frame
36 for (i in 1:nrow(gtd_data)) {
37   # Verificar si la asociación ya existe en el diccionario
38   #print(gtd_data[i, index1])
39   if (!is.na(gtd_data[i, index1]) && !is.null(gtd_data[i, index2]) &&
40     !gtd_data[i, index1] %in% names(asociacion_unica)) {
41     # Agregar la asociación al diccionario
42     asociacion_unica[[as.character(gtd_data[i, index1])]] <- gtd_data[i, index2]
43   }
44 }
45
46 for (i in 1:nrow(gtd_data)) {
47   #Llenar valores faltantes en variable1 usando la asociación
48   if (is.na(gtd_data[i, index1]) && !is.na(gtd_data[i, index2])) {
49     num_asociado <- names(asociacion_unica)[asociacion_unica == gtd_data[i, index2]]
50     gtd_data[i, index1] <- as.numeric(num_asociado)
51   }
52 }
53 return(gtd_data)
54 }
```

```

57 # Recorrer las columnas que terminan en "_txt"
58 for (columna in columnas_txt) {
59   col_sin_txt <- quitar_terminacion_txt(columna)
60   gtd_data_limpieza <- crear_asociacion_unica_y_llenar(gtd_data_limpieza, col_sin_txt, columna)
61 }
62
63 # Eliminar columnas con terminación en _txt
64 for (columna in columnas_txt) {
65   gtd_data_limpieza <- gtd_data_limpieza[, -which(names(gtd_data_limpieza) == columna), drop = FALSE]
66 }
67
68 # Vemos el resultado:
69 str(gtd_data_limpieza)
70 ncol(gtd_data_limpieza)
71

```

En el siguiente paso, eliminamos aquellas columnas que contenían más del 90% de valores perdidos. Esta decisión se basó en la premisa de que no aportarían información sustancial, ya que al intentar llenar los valores perdidos podríamos añadir una cantidad significativa de datos que no serían precisos. Consideraremos que de esta forma evitamos la inclusión de información poco fiable que podría afectar la predicción de la variable objetivo (success).

```

73 # -- Eliminación de columnas con más del 90% de valores perdidos: --
74
75 eliminar_columnas_valores_perdidos <- function(datos, umbral = 90) {
76   # Transforma cadenas vacías a NAs
77   gtd_data_con_na <- datos %>%
78     mutate_all(~ ifelse(. %in% c("", " "), NA, .))
79
80   # Calcula el porcentaje de valores perdidos por columna
81   porcentaje_perdido <- colMeans(is.na(gtd_data_con_na)) * 100
82
83   # Encuentra las columnas que superan el umbral
84   columnas_a_eliminar <- names(porcentaje_perdido[porcentaje_perdido > umbral])
85
86   # Elimina las columnas identificadas
87   datos_limpio <- datos[, setdiff(names(datos), columnas_a_eliminar)]
88
89   return(datos_limpio)
90 }
91
92 # Aplicamos a gtd_data
93 gtd_data_limpieza <- eliminar_columnas_valores_perdidos(gtd_data_limpieza, 90)
94 str(gtd_data_limpieza)
95 ncol(gtd_data_limpieza)
96

```

Después de eliminar las columnas con la mayoría de los valores perdidos, decidimos realizar una eliminación manual de columnas que consideramos que no eran de gran utilidad para la tarea predictiva:

```

97 # Eliminamos columnas de forma manual
98 columnas_a_eliminar <- c("location", "summary", "nwound", "propcomment", "addnotes", "scite2", "scite3", "related")
99 gtd_data_limpieza <- gtd_data_limpieza[, !(names(gtd_data_limpieza) %in% columnas_a_eliminar)]
100 str(gtd_data_limpieza)
101

```

En el paso siguiente, llevamos a cabo un balanceo en el conjunto de datos original, garantizando una cantidad equitativa de instancias para las clases "success" igual a 0 y "success" igual a 1. Esta práctica es importante para prevenir desequilibrios en la cantidad de ejemplos de cada clase, especialmente en tareas de clasificación.

```

110 # ---- IGUALAMOS LA CANTIDAD DE INSTANCIAS DE SUCCESS = 0 CON las = a 1 ----
111 set.seed(123)
112 gtd_data_s1 <- gtd_data_limpieza[gtd_data_limpieza$success == 1,]
113 gtd_data_s0 <- gtd_data_limpieza[gtd_data_limpieza$success == 0,]
114 occurrences_s0 <- nrow(gtd_data_s0)
115 gtd_data_s1 <- gtd_data_s1[sample(nrow(gtd_data_s1), occurrences_s0),]
116 muestra <- bind_rows(gtd_data_s1, gtd_data_s0)

```

Finalmente, para la selección de atributos, hicimos uso de χ^2 para conservar los atributos que en efecto, tuvieran alguna influencia en la variable success. Es por eso, que decidimos conservar los que tuvieran un resultado mayor a cero.

```

---  

120 library(dplyr)  

121 library(mlbench)  

122 library(FSelector)  

123  

124 # Utilizando el enfoque de filtros por chi cuadrada  

125 pesos <- chi.squared(success ~ ., data = muestra)  

126 pesos  

127  

128 # Pesos en orden de importancia  

129 orden <- order(pesos$attr_importance)  

130 dotchart(pesos$attr_importance[orden], labels=rownames(pesos)[orden], xlab="Importancia")  

131 subconjunto <- rownames(pesos)[pesos$attr_importance > 0]  

132  

133 gtd_data_seleccion <- muestra[, subconjunto]  

134 # Paso eventid al principio:  

135 gtd_data_seleccion <- gtd_data_seleccion %>% select(eventid, everything())  

136 # Añado el atributo success al final:  

137 success <- (muestra$success)  

138 gtd_data_seleccion <- cbind(gtd_data_seleccion, success)  

139  

140 str(gtd_data_seleccion)  

141 ncol(gtd_data_seleccion)

```

Al finalizar este paso, podemos observar que mantuvimos un total de 49 atributos para realizar nuestra predicción.

Los atributos eliminados son:

| | | | |
|------|--------------------|--------------------|----------------------|
| [1] | "approxdate" | "resolution" | "country_txt" |
| [4] | "region_txt" | "location" | "summary" |
| [7] | "crit1" | "crit2" | "alternative_txt" |
| [10] | "attacktype1_txt" | "attacktype2" | "attacktype2_txt" |
| [13] | "attacktype3" | "attacktype3_txt" | "targtype1_txt" |
| [16] | "targsubtype1_txt" | "natlty1_txt" | "targtype2" |
| [19] | "targtype2_txt" | "targsubtype2" | "targsubtype2_txt" |
| [22] | "corp2" | "target2" | "natlty2" |
| [25] | "natlty2_txt" | "targtype3" | "targtype3_txt" |
| [28] | "targsubtype3" | "targsubtype3_txt" | "corp3" |
| [31] | "target3" | "natlty3" | "natlty3_txt" |
| [34] | "gsubname" | "gname2" | "gsubname2" |
| [37] | "gname3" | "gsubname3" | "guncertain2" |
| [40] | "guncertain3" | "individual" | "claimmode_txt" |
| [43] | "claim2" | "claimmode2" | "claimmode2_txt" |
| [46] | "claim3" | "claimmode3" | "claimmode3_txt" |
| [49] | "complaint" | "weaptype1_txt" | "weapsubtype1_txt" |
| [52] | "weaptype2" | "weaptype2_txt" | "weapsubtype2" |
| [55] | "weapsubtype2_txt" | "weaptype3" | "weaptype3_txt" |
| [58] | "weapsubtype3" | "weapsubtype3_txt" | "weaptype4" |
| [61] | "weaptype4_txt" | "weapsubtype4" | "weapsubtype4_txt" |
| [64] | "nwound" | "propextent_txt" | "propvalue" |
| [67] | "propcomment" | "nhostkid" | "nhostkidus" |
| [70] | "nhours" | "ndays" | "divert" |
| [73] | "kidhijcountry" | "ransom" | "ransomamt" |
| [76] | "ransomamtus" | "ransompaid" | "ransompaidus" |
| [79] | "ransomnote" | "hostkidoutcome" | "hostkidoutcome_txt" |
| [82] | "nreleased" | "addnotes" | "scite2" |
| [85] | "scite3" | "related" | |

Por otro lado, los 49 atributos conservados son:

```
'eventid' · 'iyear' · 'imonth' · 'iday' · 'extended' · 'country' · 'region' · 'provstate' · 'city' · 'latitude' · 'longitude' · 'specificity' ·  

'vicinity' · 'crit3' · 'doubtterr' · 'alternative' · 'multiple' · 'suicide' · 'attacktype1' · 'targtype1' · 'targsubtype1' · 'corp1' · 'target1' ·  

'natlty1' · 'gname' · 'motive' · 'guncertain1' · 'nperps' · 'nperpcap' · 'claimed' · 'claimmode' · 'weaptype1' · 'weapsubtype1' ·  

'weapdetail' · 'nkill' · 'nkillus' · 'nkiller' · 'nwoundus' · 'nwoundte' · 'property' · 'propextent' · 'ishostkid' · 'scite1' · 'dbsource' ·  

'INT_LOG' · 'INT_IDEO' · 'INT_MISC' · 'INT_ANY' · 'success'
```

- Manejo de valores perdidos. Considera los siguientes métodos para tratar con valores perdidos:
 - Reemplaza los valores perdidos por la media, mediana o la moda, de acuerdo con el tipo de dato del atributo. En este caso decidimos usar la media para los valores de clase *numeric*, esto pues todos los atributos de este tipo tienen una distribución normal o que deriva de la distribución normal, por lo que la media resultaba una medida aceptable para sustituir los valores.

En el caso de los valores enteros y para evitar los problemas que los decimales dados por la división al calcular la media, se optó por usar la mediana. Finalmente en los atributos que tienen clase *character* o *factor* lo que hicimos fue aplicar la moda, esto pues no se pueden aplicar alguna de las otras dos medidas. En específico usamos la función *mode2* que se adapta mejor a atributos de este tipo.

```
mode2 <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

imputacion <- function(data){
  for (var in 1:ncol(data)) {
    if (class(data[,var])=="numeric") {
      data[is.na(data[,var]),var] <- mean(data[,var], na.rm = TRUE)
    } else if (class(data[,var]) %in% c("character", "factor")) {
      no_empty <- na.omit(data[,var][data[,var] != ""])
      m <- mode2(no_empty)
      data[is.na(data[,var]),var] <- m
      data[data[,var]== "",var] <- m
    } else if (class(data[,var]) == "integer"){
      data[is.na(data[,var]),var] <- median(data[,var], na.rm = TRUE)
    }
  }
  return(data)
}

# Aplicamos:
gtd_data_sin_vp <- imputacion(gtd_data_seleccion)
```

- Utiliza algún método de Aprendizaje Automático para predecir los valores perdidos de cada atributo.

En este caso decidimos utilizar la biblioteca missForest para imputar los valores perdidos. Sin embargo por la complejidad que implica la regresión y demás operaciones realizadas el relleno de valores, solo pudimos ejecutarlo con el 1% de los valores.

```
porcentaje_muestreo <- 0.01
tamano_muestra <- round(nrow(muestra) * porcentaje_muestreo)
muestra <- muestra[sample(nrow(muestra), tamano_muestra),]
```

Es por la cual nos vimos obligados a utilizar los valores imputados por la media, moda o mediana para el resto del procedimiento.

Además cabe aclarar que por las características del missForest se tuvo que realizar un mapeo a números de todos los atributos de caracteres.

```

gtd_data_numeric<-data.frame(muestra)
cols_names <- colnames(gtd_data)
j <- 1
set.seed(1)
for(i in gtd_data_numeric) {
  n = nlevels(i)
  if(class(i) == 'character'){
    x <- as.factor(i)
    gtd_data_numeric[cols_names[j]]<- as.numeric(x)
  }
  if(class(i) == 'factor'){
    gtd_data_numeric[cols_names[j]]<- as.numeric(i) #as.factor(sample(i, n, replace=TRUE))
  }
  j <-j+1
}

# Imputar los valores perdidos, usando los parámetros con valores default
gtd_data_sin_vp2 <- missForest(gtd_data_numeric)

```

- Eliminación de valores atípicos.

```

----- ELIMINACIÓN DE VALORES ATÍPICOS -----
columnas_con_nas <- colSums(is.na(gtd_data_sin_vp)) > 0
columnas_con_nas <- names(columnas_con_nas[columnas_con_nas])
print(columnas_con_nas)
quitar_atipicos <- function(columna, threshold = 1.5) {
  # Calcular el rango intercuartílico para la columna
  iqr <- IQR(columna, na.rm = TRUE)

  # Calcular los límites para identificar valores atípicos
  lower_limit <- median(columna, na.rm = TRUE) - threshold * iqr
  upper_limit <- median(columna, na.rm = TRUE) + threshold * iqr

  # Filtrar los valores atípicos
  columna_filtrada <- ifelse(columna < lower_limit | columna > upper_limit, NA, columna)

  return(columna_filtrada)
}

columnas_con_nans <- colSums(is.na(gtd_data_sin_vp)) > 0

# Mostrar las columnas con valores faltantes
print(columnas_con_nans)
# Aplicamos a nuestro dataset:
str(gtd_data_sin_vp)

```

```

columnas_con_nans <- colSums(is.na(gtd_data_sin_vp)) > 0

# Mostrar las columnas con valores faltantes
print(columnas_con_nans)
# Aplicamos a nuestro dataset:
str(gtd_data_sin_vp)
columnas_numericas <- gtd_data_sin_vp[sapply(gtd_data_sin_vp, is.numeric)]
str(columnas_numericas)
print(dim(gtd_data_sin_vp))
hay_infinitos <- sapply(columnas_numericas, function(col) any(is.infinite(col)))
# Imprimir el resultado
print(hay_infinitos)

hay_nas <- sapply(columnas_numericas, function(col) any(is.na(col)))

# Imprimir el resultado
print(hay_nas)
# Verificar las columnas seleccionadas y sus tipos
print(sapply(columnas_numericas, class))
for (nombre_columna in names(columnas_numericas)) {
  gtd_data_sin_vp[[nombre_columna]] <- quitar_atipicos(gtd_data_sin_vp[[nombre_columna]])
}

#gtd_data_sin_atipicos <- quitar_atipicos(gtd_data_sin_vp)
head(gtd_data_sin_vp)

```

Definimos la función `quitar_atipicos`, la cual toma una columna y un umbral como argumentos, empleando el rango intercuartílico (IQR) para identificar y filtrar valores atípicos. El valor "1.5" utilizado como umbral es una elección convencional respaldada por la regla general del IQR, ofreciendo un equilibrio efectivo para clasificar observaciones como atípicas.

Esta elección de "1.5" se ha convertido en una convención estándar en la comunidad estadística debido a su eficacia y capacidad para equilibrar la detección de valores atípicos moderados sin clasificar en exceso. Posteriormente, aplicamos esta función a las columnas numéricas del conjunto de datos, reemplazando los valores atípicos con NA según los límites establecidos por el rango intercuartílico.

La decisión de utilizar este enfoque se basó en la necesidad de manejar valores atípicos de manera efectiva. Inicialmente, utilizamos la función `colMeans` para calcular medias, pero se reportaron problemas de NA's que no estaban presentes. Después de realizar verificaciones exhaustivas antes y después de nuestro código, confirmamos que este enfoque no identificaba NA's ni valores infinitos, permitiéndonos trabajar con el conjunto de datos de manera más consistente.

- Discretización de atributos numéricos. Si usas alguna discretización específica, cuál usaste.

Consideramos que la discretización sería útil para casos como árboles CART y reglas de asociación porque facilita la interpretación y el manejo de variables numéricas al dividirlas en categorías o intervalos.

Optamos por la discretización por rango debido a que inicialmente consideramos utilizar la discretización por frecuencia, la cual es más efectiva en presencia de valores atípicos. Sin embargo, la versión de R que estamos utilizando no admitía fácilmente esta opción. Por lo tanto, elegimos la discretización por rango, utilizando el método de Sturges para determinar la cantidad de bins necesarios. La fórmula para calcular los bins utilizando el método de Sturges es:

$$\text{bins} = \lceil \log_2(\text{muestras}) + 1 \rceil$$

Esta fórmula proporciona una estimación del número adecuado de bins (intervalos) para la discretización, considerando la cantidad de muestras en el conjunto de datos.

```

262 #---- DISCRETIZACIÓN ----
263 library(dplyr)
264 library(ggplot2)
265
266 # Discretizar por rango
267 discretizar_por_rango <- function(data) {
268   # Identificar columnas numéricas
269   columnas_numericas <- sapply(data, is.numeric)
270
271   for (columna in names(data)[columnas_numericas]) {
272
273     # Calcular el número de bins utilizando la fórmula de Sturges
274     num_bins <- ceiling(log2(length(data[[columna]])) + 1)
275
276     data[[columna]] <- cut_interval(data[[columna]], n = num_bins, dig.lab = 9)
277   }
278
279   return(data)
280 }
281

```

- Normalización. Justifica la necesidad de la normalización.

Llevamos a cabo la normalización min-max de los datos con ayuda de la función que se muestra en la siguiente imagen:

```

302
303   #---- NORMALIZACIÓN ----
304
305   # Normalización min-max
306   min_max <- function(x) {
307     return((x - min(x)) / (max(x) - min(x)))
308   }
309

```

La normalización de los datos es esencial para evitar que las variables con grandes rangos numéricos dominen sobre aquellas con rangos numéricos pequeños, evitando sesgos y permitiendo un mejor rendimiento en los algoritmos que utilizaremos en la etapa de minería, como lo son k-means y la red neuronal.

4 Conclusiones Preprocesamiento de datos

Con ayuda del proceso de análisis en la etapa inicial se pudo identificar ciertos valores que no estaban identificados como NA, pero que sin embargo representaban una falta de conocimiento en los datos, entre ellos cadenas vacías "", cadena "Unknown", valores de -9 y -99. Estos valores solo causarían sesgo en nuestros resultados así que se asignaron como valores NA.

También fueron identificadas columnas con terminación *.txt* asociadas con su representación numérica en la columna anterior por lo que estás no serían necesarias. Pero para ciertas filas no tenían un número asignado por lo que fueron completados con la información ya recopilada en el resto de las tuplas.

Identificados los puntos anteriores pudimos proceder a una limpieza de datos más general, en la que eliminamos columnas con un umbral de valores perdidos del 90% que no aportarán información.

Adicionalmente detectamos columnas con información poco relevante para nuestro objetivo por ser información muy detallada o específica, tales como *location*, *summary*, *addnotes*, *scite2*, *scite3*, *etc*, y que fueron también eliminadas.

En esta sección buscamos, de forma general, limpiar nuestra base de datos de columnas con poca información, datos atípicos, columnas que no nos servirán para la etapa de minería, tratar en la medida de lo posible reemplazar valores perdidos, y dejar preparados nuestros datos para las funciones requeridas la etapa posterior (discretización y normalización).

Por último es importante mencionar que debido a que nuestros equipos no pudieron trabajar con la cantidad de datos que resultaron de este proceso, y para balancear los datos de acuerdo a nuestra variable objetivo *success* se seleccionaron de manera aleatoria pero en igual proporción entradas con valores de 0 y 1 para dicha variable.

(c) Minado

- Clasificación

5 Conclusiones Clasificación

Después de explorar y experimentar con distintas configuraciones de árboles CART y redes neuronales, evaluamos su desempeño en nuestro conjunto de datos. Probamos diversas opciones de ajuste de hiperparámetros para los árboles CART, incluyendo la variación de minsplit, el podado y la profundidad máxima del árbol. A pesar de obtener resultados decentes con diferentes configuraciones, optamos por utilizar un árbol con poda y minsplit ajustado, ya que proporcionó un equilibrio efectivo entre precisión y generalización y nos mostró mejores resultados.

En cuanto a las redes neuronales, probamos varias configuraciones, y después de evaluar su rendimiento, decidimos utilizar la red neuronal número 5 (red5) que se repite 50 veces. Esta elección se basó en su rendimiento tanto en el conjunto de entrenamiento como en el de prueba.

```
51 arbol1 <- rpart(success ~ .,
52                     control = rpart.control(minsplit = 30,maxdepth = 10),
53                     data = gtd_entrenamiento,method = "class")
54
55 arbol1 <- prune(arbol1, cp = 0.1) # podando árbol
56 rpart.plot(arbol1,main = "Exito de ataque")
57 prediccion1 <- predict(arbol1, newdata = gtd_prueba, type = "class")
58 confusionMatrix(prediccion1, as.factor(gtd_prueba[["success"]]))
59
60
61 # Red Neuronal 5 : se repite 50 veces
62 red5 <- neuralnet(form, entrena, hidden = 9, linear.output = FALSE, rep = 50)
63
```

- **Evaluación de modelos de clasificación**

Usamos el siguiente código para obtener las medidas: exactitud, error de clasificación, sensibilidad, especificidad, precisión, valor predicho negativo, falsas alarmas, tasa de falsos descubrimientos, tasa de falsos negativos, exactitud balanceada, F-measure e índice kappa

```

exactitud <- function (TP,TN, FP,FN){
  return ((TP + TN) / (TP + FN +FP + TN))
}

error_clasificaccion <- function (TP,TN, FP,FN){
  return ((FP + FN) / (TP + FN +FP + TN))
}

sensibilidad<- function (TP,FN){
  return (TP / (TP + FN ))
}

especificidad<- function (TN, FP){
  return (TN / (TN + FP))
}

precision <-function (TP, FP){
  return (TP / (TP + FP))
}

NPV <- function (TN, FN){
  return (TN / ( TN + FN))
}

FP_rate <- function (FP,TN){
  return (FP / (FP + TN))
}

FDR<- function (TP, FP){
  return (FP / (FP + TP))
}

FN_rate <- function (TP,FN){
  return (FN / (FN + TP))
}

BACC <- function (TP,TN, FP,FN){
  return (((TP / (TP + FN)) + (TN / (FP + TN))) / 2)
}

F_measure<- function (p,r){
  return ((2*r*p)/(r + p))
}

kappa <- function (TP,TN, FP,FN,E){
  return ((TP + TN - E) / (TP + FN +FP + TN - E))
}

```

. Así con el apoyo de la siguiente función:

```

tabla_medidas <- function(TP,TN, FP, FN){
  num_filas <- 1
  num_columnas <- 12
  medidas <- matrix(nrow = num_filas, ncol = num_columnas)

  colnames(medidas) <- c("Exactitud", "Error de clasificación","Sensibilidad","Especificidad",
                         "Precisión", "NPV", "Falsas alarmas", "FDR", "FN rate", "BACC",
                         "F-measure", "Indice kappa")
  medidas[1, 1] = exactitud(TP,TN, FP, FN)
  medidas[1, 2] = error_clasificaccion (TP,TN, FP, FN)
  medidas[1, 3] = sensibilidad(TP,FN)
  medidas[1, 4] = especificidad(TN, FP)
  medidas[1, 5] = precision(TP, FP)
  medidas[1, 6] = NPV (TN, FN)
  medidas[1, 7] = FP_rate(TP, FP)
  medidas[1, 8] = FDR(TP, FP)
  medidas[1, 9] = FN_rate(TP,FN)
  medidas[1, 10] = BACC(TP,TN, FP, FN)
  r <- medidas[1, 3]
  p <- medidas[1, 5]
  medidas[1, 11] = F_measure(r,p)
  E <- 1#revisar E(TP+TN)
  medidas[1, 12] = kappa(TP,TN, FP, FN,E)
  return(medidas)
}

```

pudimos obtener acorde al mejor caso para cada clasificador las respectivas medidas:

```

confusionArbolCART <- c(365,352,327,320)
confusionRedNeuronal <- c(5966,6068,0,1)
medidas_arbol <- tabla_medidas(confusionArbolCART[1],confusionArbolCART[2],confusionArbolCART[3],confusionArbolCART[4])
print(medidas_arbol)
medidas_red <- tabla_medidas(confusionRedNeuronal[1],confusionRedNeuronal[2],confusionRedNeuronal[3],confusionRedNeuronal[4])
print(medidas_red)

```

Esto donde cada vector que representa a la matriz de clasificación de cada clasificador sigue el orden: TP, TN, FP, FN .

Así obtuvimos:

```

> medidas_arbol <- tabla_medidas(confusionArbolCART[1],confusionArbolCART[2],confusionArbolCART[3],confusionArbolCART[4])
> print(medidas_arbol)
  Exactitud Error de clasificación Sensibilidad Especificidad Precisión      NPV
[1,] 0.5256598   0.4743402   0.5328467   0.5184094 0.5274566 0.5238095
  Falsas alarmas      FDR   FN rate     BACC F-measure Indice kappa
[1,] 0.5274566 0.4725434 0.4671533 0.5256281 0.530138 0.5253118
> medidas_red <- tabla_medidas(confusionRedNeuronal[1],confusionRedNeuronal[2],confusionRedNeuronal[3],confusionRedNeuronal[4])
> print(medidas_red)
  Exactitud Error de clasificación Sensibilidad Especificidad Precisión      NPV
[1,] 0.9999169   8.309098e-05  0.9998324   1           1 0.9998352
  Falsas alarmas      FDR   FN rate     BACC F-measure Indice kappa
[1,] 1           0 0.0001675884 0.9999162 0.9999162 0.9999169

```

6 Conclusiones Evaluación de modelos de clasificación

En este caso podemos ver que todas las medidas nos marcan como un mejor clasificador al árbol CART por sobre la red neuronal. Siendo que lo anterior es algo que su pudo predecir desde la matriz de confusión pues la diferencia de números en la diagonal es bastante grande. Tenemos además que dichas medidas son confiables pues teníamos que había igual número de ejemplares con *succes* igual a 0 como con *succes* igual a 1. Así para este conjunto de datos resulta un mejor clasificador el árbol CART, ya que nos solo obtuvo mejores métricas, también costo menos recursos el lograr que legase a dicho resultado.

• Asociación

Dado que nuestra variable objetivo es **success** la misma no dirá si un ataque terrorista tuvo éxito o no; El uso de reglas de asociación como Apriori o ECLAT generalmente se aplica cuando se trabaja con conjuntos de datos que contienen transacciones o elementos que se asocian entre sí. Estos algoritmos buscan patrones de co-ocurrencia o asociación entre los elementos de las transacciones. En el contexto de un conjunto de datos donde la variable de interés es **success**, el uso de reglas de asociación puede no ser la elección más apropiada, para nuestro enfoque.

Las reglas de asociación son más adecuadas cuando se busca descubrir patrones de compra, comportamientos de usuarios, o relaciones entre elementos en transacciones, como sabemos y lo hemos visto en el curso.

Ahora, en el caso de variables binarias como **success**, es posible que otros métodos, como modelos de clasificación (como los árboles de decisión), sean más relevantes y proporcionen una comprensión más específica de los factores que influyen en el éxito de un ataque terrorista.

Por lo que, mientras que las reglas de asociación son poderosas para descubrir patrones en conjuntos de datos de transacciones, para analizar la variable binaria **success**, es posible que desee considerar métodos de modelado de clasificación como los arboles de desición , los cuales sabemos que se centran directamente en predecir y entender el éxito o fracaso de los eventos en función de otras variables explicativas.

A pesar de que la naturaleza de nuestra variable de interés, "success", es binaria, hemos optado por aplicar los algoritmos de reglas de asociación, como Apriori y ECLAT. La razón detrás de esta elección es explorar si estos algoritmos pueden proporcionar información valiosa o patrones interesantes que podrían arrojar luz sobre relaciones ocultas o factores asociados con el éxito. Aunque estos algoritmos están diseñados principalmente para conjuntos de datos de transacciones, creemos que podrían revelar insights inesperados en nuestro contexto particular.

Esta decisión se basa en la idea de que, incluso cuando la variable objetivo es binaria, los algoritmos de reglas de asociación pueden identificar patrones de co-ocurrencia o interacción entre diferentes atributos que podrían ser de interés. Aunque su aplicación no es la convencional, estamos explorando la posibilidad de extraer algún beneficio o información valiosa de estos algoritmos para enriquecer nuestra comprensión del éxito en nuestros datos Sin embargo hemos decidido aplicar el algoritmo para ver si podemos sacar aunque sea un poco de provecho de las mismas y nos puedan dar un poco de información que nos interesa.

Lo que hacemos se divide en tres fase, las cuales son las siguientes :

- En la primera fase, el código inicia instalando y cargando paquetes esenciales, como "arules" y "arulesViz", para llevar a cabo el análisis de reglas de asociación. Luego, se carga un conjunto de datos desde el archivo CSV "datosPrepTerrorismo.csv". Este conjunto de datos está estructurado en forma de transacciones, donde cada fila representa un incidente de terrorismo, y las columnas contienen diversos atributos relacionados con esos incidentes.
- La segunda etapa se centra en la generación de reglas de asociación mediante el algoritmo Apriori y el método ECLAT. El código emplea bucles para explorar diferentes combinaciones de niveles de soporte y confianza al aplicar el algoritmo Apriori.
- Posteriormente, en la tercera etapa, se visualiza la cantidad de reglas encontradas en función de la confianza para distintos niveles de soporte mediante gráficos, proporcionando una perspectiva clara de las asociaciones descubiertas en los datos. En la última fase, se evalúan las reglas generadas utilizando métricas como el Test Exacto de Fisher. El código realiza un filtrado de las reglas, seleccionando aquellas que contienen el consecuente "success". El resultado final es un resumen de estas reglas filtradas, presentadas en orden descendente según la confianza. Este análisis ofrece

información valiosa sobre las asociaciones entre los diferentes atributos presentes en los incidentes de terrorismo, contribuyendo a la comprensión más profunda de los patrones y relaciones en los datos.

```
A tibble: 3,107,628 × 6
  rules          support confidence coverage lift count
  <chr>        <dbl>    <dbl>    <dbl> <dbl> <int>
1 {} => {success=[0,1]}      1       1  1   1 18169
2 {} => {extended=[0,1]}     1       1  1   1 18169
3 {attacktype1=[1,2]} => {ishostkid=[0... 0.106     1 0.106 1.00 1922
4 {attacktype1=[1,2]} => {divert=Nepal} 0.106     1 0.106 1.00 1922
5 {attacktype1=[1,2]} => {gsubname2=Ez... 0.106     1 0.106 1.00 1922
6 {attacktype1=[1,2]} => {success=[0,... 0.106     1 0.106 1 1922
7 {attacktype1=[1,2]} => {extended=[0,... 0.106     1 0.106 1 1922
8 {property=[-9,0)} => {success=[0,1]} 0.120     1 0.120 1 2176
9 {property=[-9,0)} => {extended=[0,1]} 0.120     1 0.120 1 2176
10 {targtype1=[1,3)} => {success=[0,1]} 0.224    1 0.224 1 4073
# i 3,107,618 more rows
# i Use `print(n = ...)` to see more row
```

```
# A tibble: 3,107,628 × 7
  rules          support confidence coverage lift count metricas
  <chr>        <dbl>    <dbl>    <dbl> <dbl> <int> <dbl>
1 {} => {success=[0,1]}      1       1  1   1 18169 1
2 {} => {extended=[0,1]}     1       1  1   1 18169 1
3 {attacktype1=[1,2]} => {isho... 0.106     1 0.106 1.00 1922 0.0390
4 {attacktype1=[1,2]} => {dive... 0.106     1 0.106 1.00 1922 0.0545
5 {attacktype1=[1,2]} => {gsub... 0.106     1 0.106 1.00 1922 0.167
6 {attacktype1=[1,2]} => {succ... 0.106     1 0.106 1 1922 1
```

Al final, la parte de ECLAT la pusimos en el proyecto pero al darnos cuenta de que no sería en el primer metodo de apriori decidimos , omitir los resultados de ECLAT.

7 Conclusiones Reglas de Asociación

Al final podemos notar que realmente no tenemos alguna regla relevante que nos ayude a nuestro objetivo, lo cual era algo previsto.

No nos dan información relevante, dado que :

- Reglas con success=[0,1] en el consecuente: Estas reglas indican asociaciones con la variable "success". La primera regla sugiere que siempre que ocurra el evento (indicado por la presencia de "success" en el consecuente), la probabilidad de éxito es del 100%. Este tipo de regla no proporciona información significativa, ya que simplemente indica que "success" siempre está presente.
- Reglas con otras variables en el antecedente y "success" en el consecuente: Las reglas que involucran otras variables en el antecedente junto con "success" en el consecuente podrían ser más informativas. Por ejemplo, la regla attacktype1=[1,2) → success=[0,1] indica que cuando el "attacktype1" es [1,2], el evento tiene un

100% de probabilidad de éxito. Estas reglas pueden ayudar a identificar condiciones específicas asociadas con el éxito.

Realmente, podemos obtener mejor información en la parte del árbol, que lo que nos puede llegar ayudar está parte pero es bueno denotarlo.

- **Agrupación**

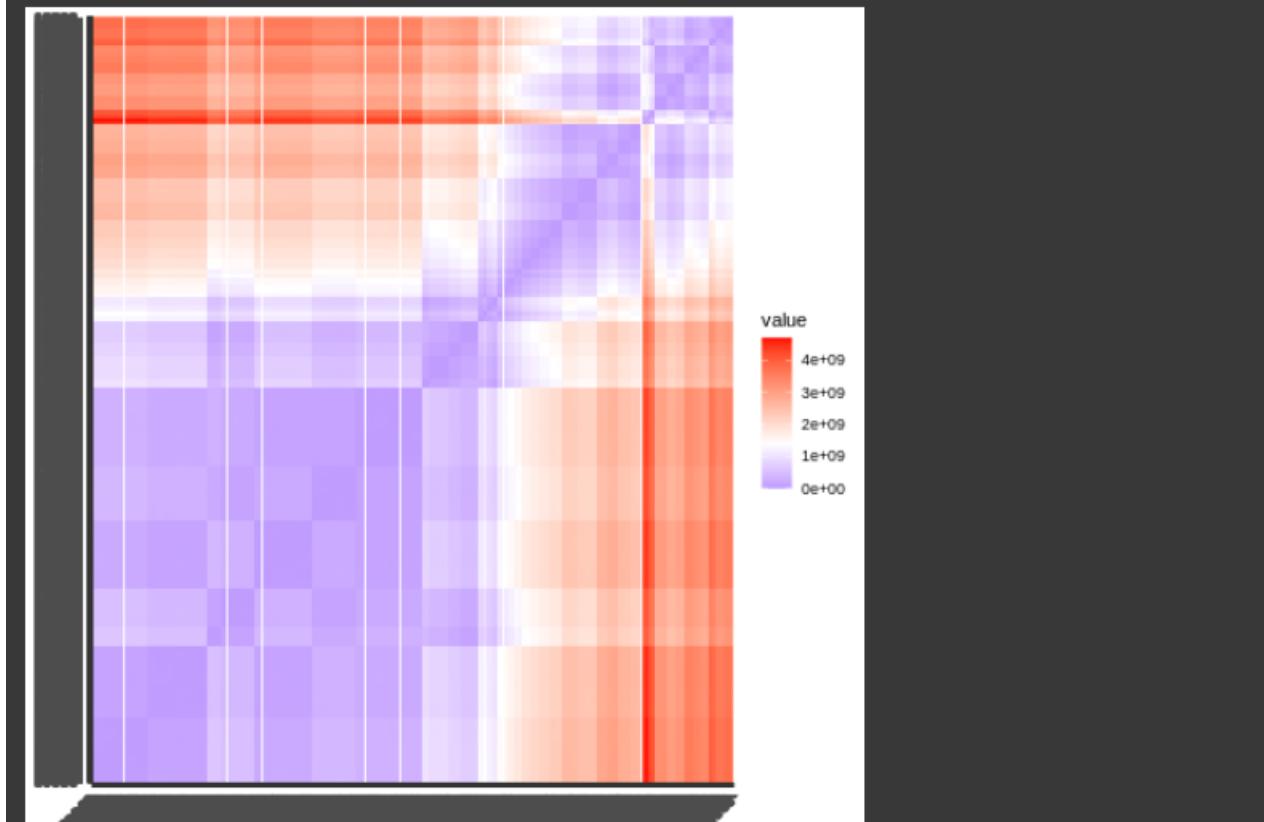
8 Conclusiones Agrupación

Debido a la falta de capacidad de nuestros equipos, solo fue posible trabajar con una muestra del 10% de nuestros datos para llevar a cabo la agrupación.

Empezamos calculando la matriz de distancias para representar las distancias euclidianas entre pares de observaciones. En la imagen siguiente se puede apreciar en color azul la intersección de observaciones cercanas y en rojo las no cercanas.

Notamos que existe una alta cercanía en varios valores en la parte inferior izquierda, pero estos están alineados a la diagonal de identidad, por lo que puede que realmente no se deba a una relación existente entre nuestras variables.

```
# calcular la matriz de distancias
m.distancia <- get_dist(gtd_data, method = "euclidean")
fviz_dist(m.distancia, gradient = list(low = "blue", mid = "white", high = "red"))
```



Posteriormente estimamos el número óptimo de clústers en nuestro conjunto de datos, esto a través de el análisis de factores y diferentes métodos de agrupamiento: Within-Cluster Sum of Squares (WSS), Silhouette y Gap Statistic.

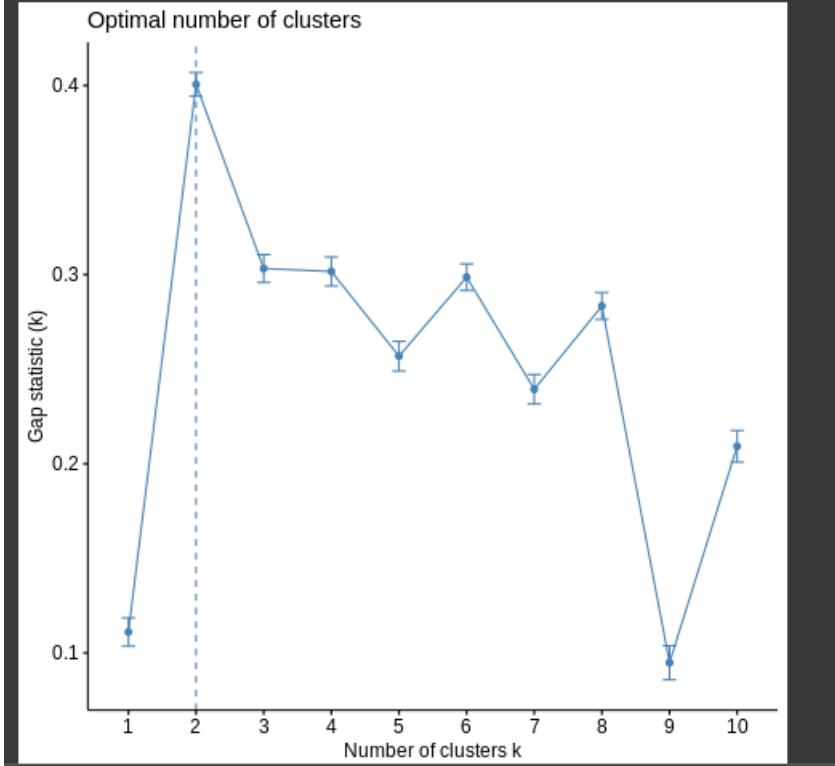
En esta última obtenemos el siguiente gráfico, donde se nos indica, y es fácil visualizar que el número óptimo de clústers es 2.

```

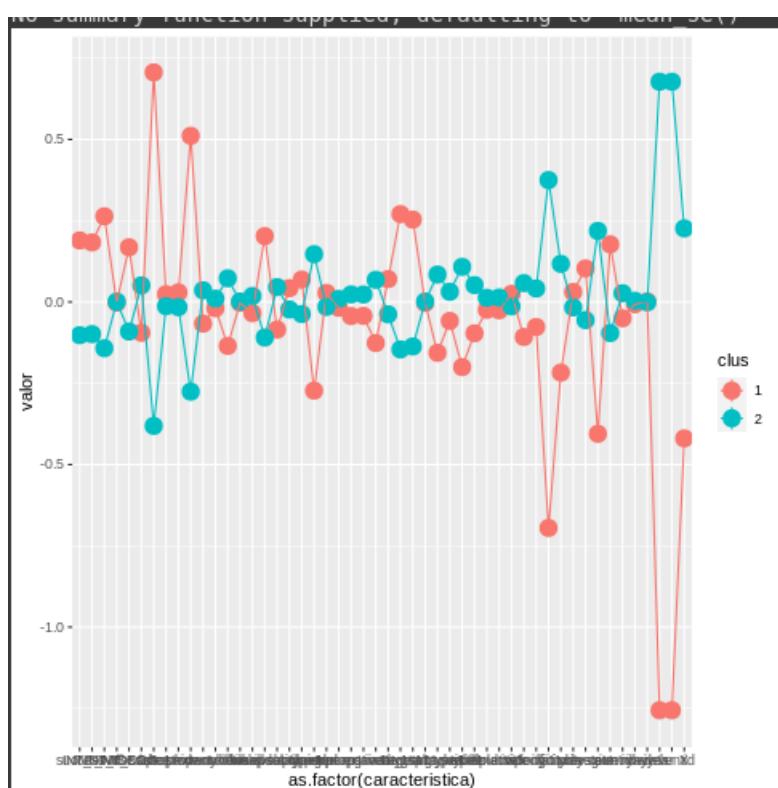
fviz_nbclust(gtd_data, kmeans, method = "wss")
fviz_nbclust(gtd_data, kmeans, method = "silhouette")
fviz_nbclust(gtd_data, kmeans, method = "gap_stat")

Clustering k = 1,2,..., K.max (= 10): ... done
Bootstrapping, b = 1,2,..., B (= 100) [one "." per sample]:
..... 50
..... 100

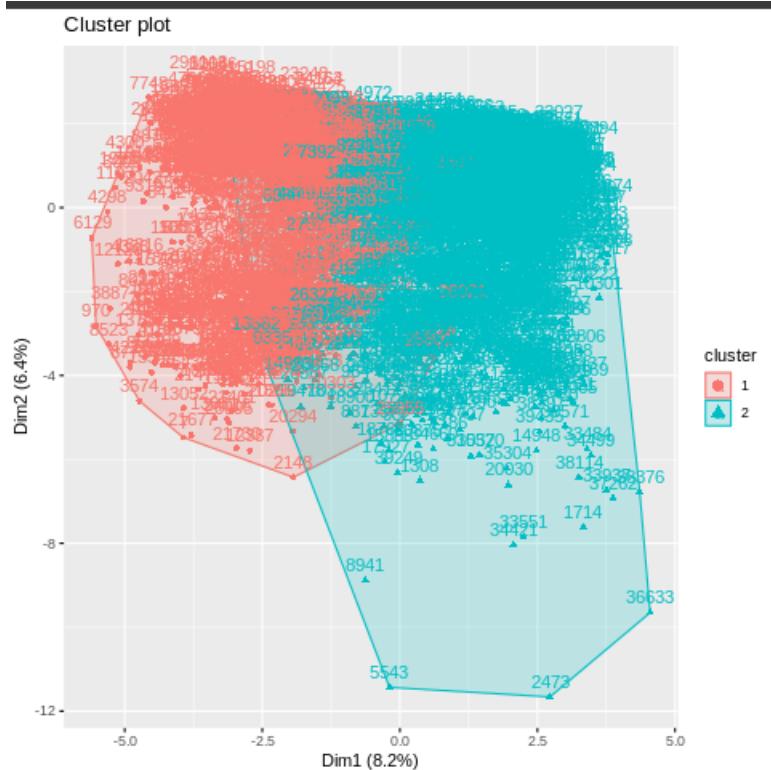
```



Con el resultado anterior, podemos calcular los 2 clústers con k-means, y observamos el siguiente resultado:



Finalmente creamos un gráfico que muestra la relación entre las características y sus valores, diferenciando por el factor de clúster.



9 Conclusiones Finales

El dataset Global Terrorism, puede ser de gran utilidad a la hora de tratar de buscar evitar la realización de ataques terroristas. Por un lado ahora tenemos presente que el éxito de un ataque de esta clase se ve sumamente relacionado con –variable con alta correlación–. De manera que las estrategias a tomar deberán ser principalmente relacionadas con dichos factores y no tanto con otros como los mostrados en la sección de preprocesamiento. Para realizar las tareas predictivas fue fundamental la limpieza del conjunto de datos y eliminación de ciertos atributos que no tenían ninguna influencia sobre la predicción el éxito del ataque. De igual forma, fue muy importante preparar los datos para que las tareas de clasificación, reglas de asociación y agrupación funcionaran de la mejor forma. Para ello, realizamos la imputación de valores perdidos, lo que nos permitió obtener mejores conclusiones; realizamos una discretización para facilitar las tareas de clasificación dejando todos los datos numéricos clasificados en intervalos, y generamos nuevos conjuntos de datos utilizando las estrategias anteriores para ser utilizados dependiendo del objetivo específico.

Las variables más significativas del conjunto de datos para realizar la predicción sobre el éxito de los ataques son:

- 'iyear'
- 'imonth'
- 'iday'
- 'extended'
- 'country'
- 'region'
- 'provstate'
- 'city'
- 'latitude'
- 'longitude'
- 'specificity'
- 'vicinity'
- 'crit3'
- 'doubtterr'
- 'alternative'
- 'multiple'
- 'suicide'
- 'attacktype1'
- 'targtype1'
- 'targsubtype1'
- 'corp1'
- 'target1'
- 'natlty1'
- 'gname'
- 'motive'
- 'guncertain1'
- 'nperps'
- 'nperpcap'
- 'claimed'
- 'claimmode'
- 'weaptype1'
- 'weapsubtype1'
- 'weapdetail'
- 'nkill'
- 'nkillus'
- 'nkillter'
- 'nwoundus'
- 'nwoundte'
- 'property'
- 'propextent'
- 'ishostkid'
- 'scite1'
- 'dbsource'
- 'INT_LOG'
- 'INT_IDEO'
- 'INT_MISC'
- 'INT_ANY'

Como **recomendaciones**, en primer lugar, basándonos en los atributos con mayor correlación, se destaca la importancia de enfocar recursos y estrategias preventivas en situaciones específicas, como fechas, ubicaciones y tipos de eventos, identificados como factores cruciales para predecir el éxito de un ataque. Establecer medidas de seguridad reforzadas en estos contextos podría ser fundamental. Para la tarea de clasificación, probamos árboles CART y redes neuronales, ambos modificando los parámetros para obtener las mejores conclusiones posibles. Gracias a esto, obtuvimos un clasificador bastante bueno, un árbol CART con una exactitud de 0.99, teniendo que dicha medida es confiable puesto que se hizo uso de datos no sesgados: existían igual número de ejemplares con *succes* = 0 como *success* = 1. Lo anterior puede resultar sorprendente sobre todo si consideramos que se realizó también una red neuronal, la cual no solo ocupó mayores recursos respecto a tiempo y espacio, sino que terminó teniendo resultados considerablemente menos favorables, con una exactitud de apenas 0.52. Así sería posible hacer uso del árbol para dada la presencia de circunstancias específicas ver si se encuentra en una situación de riesgo. De igual manera se encontraron dos agrupaciones significativas de nuestros datos, y que nos ayudan a visualizar e interpretar nuestra variable objetivo.. Esto es de utilidad para identificar patrones o relaciones ocultas dentro de nuestros datos.

Por ultimo podemos ver que las reglas de asociación realmente no fueron de tanta utilidad para nuestro propósito, que es el predecir si un ataque tendrá éxito o no.