Estudio de transacciones con tarjetas de crédito fraudulentas

Carolina Armella, Ágatha del Olmo y María Yu García

2024-10-04

Índice

1.1: Introducción

El fraude en transacciones con tarjetas de crédito se ha convertido en uno de los problemas más preocupantes tanto para los consumidores como para las instituciones financieras a nivel mundial. Este proyecto tiene como objetivo desarrollar un sistema de detección de fraudes eficaz utilizando técnicas avanzadas de aprendizaje automático.

Nuestro objetivo principal es crear un modelo predictivo altamente preciso, capaz de identificar transacciones fraudulentas minimizando tanto los falsos positivos como los falsos negativos, pero especialmente los falsos negativos. La importancia de este proyecto radica en su potencial para:

• Proteger a los consumidores frente a pérdidas financieras. • Mejorar la eficiencia en la detección de fraudes, permitiendo respuestas más rápidass y precisas. • Reducir las pérdidas de las instituciones financieras. • Mantener la confianza de los clientes en los sistemas de pagos electrónicos.

La base de datos utilizada en este proyecto contiene transacciones reales con tarjetas de crédito realizadas por europeos durante un período de dos días en septiembre de 2013. Incluye un total de *284807 transacciones*, de las cuales solo *492* fueron clasificadas como fraudulentas, lo que representa apenas un *0.172%* del total. Esta baja proporción de fraudes refleja un *desequilibrio significativo* en los datos.

Por otra parte, el conjunto de datos incluye variables numéricas que han sido transformadas mediante un *Análisis de Componentes Principales (PCA)*, lo que significa que las variables originales relacionadas con las transacciones no son directamente accesibles por motivos de confidencialidad. Estas variables transformadas están etiquetadas como *V1 a V28* y representan los principales componentes del PCA que explican la mayor parte de la variabilidad en los datos originales.

Además, hay dos características que no han sido transformadas:

• *Time*: Indica el tiempo en segundos transcurrido entre cada transacción y la primera transacción registrada.

• *Amount*: Representa el valor total de la transacción.

como el muestreo de la base de datos se realizó durante un período de dos días sin especificar qué días son, la variable pierde algo de relevancia en términos de su capacidad para identificar patrones relacionados con el tiempo del día, la semana, o eventos específicos. Sería interesante contar con una variable temporal más precisa que indique el día de la semana, la hora o incluso el mes, para explorar si hay patrones específicos relacionados con el día o la hora en que se realizan las transacciones, lo cual podría ayudar a identificar diferencias en el comportamiento entre los casos fraudulentos y no fraudulentos. Por esta razón, la gran parte del estudio ignoraremos esta variable.

Finalmente, la variable de respuesta, *Class*, toma valores binarios:

• *1* para las transacciones fraudulentas

• *0* para las transacciones legítimas

En este estudio, utilizaremos modelos de clasificación con el objetivo de identificar las transacciones fraudulentas (clase 1). Compararemos el rendimiento de tres tipos de modelos: redes neuronales, KNN (k-vecinos más cercanos) y árboles de decisión. Elegimos estos modelos debido a sus características: las redes neuronales son muy eficaces para captar patrones complejos, KNN es simple y efectivo con datos no lineales, y los árboles de decisión se desempeñan bien con datos desbalanceados.

Aunque las redes neuronales son modelos de “caja negra” y su interpretación es difícil, en este caso no nos preocupa, ya que estamos trabajando con variables resultantes de un PCA, y desconocemos las originales, lo que implica que ya no son directamente interpretables. Por lo tanto, nuestro enfoque se centra principalmente en la capacidad predictiva del modelo, sin necesidad de una interpretación explícita de las características subyacentes.

En este trabajo, se evaluarán los modelos de detección de fraude usando tres métricas principales: precisión, recall y F1-score. La precisión se enfocará en minimizar los falsos positivos, mientras que el recall priorizará la detección de la mayor cantidad de fraudes posible, aunque con algunos falsos positivos. El F1-score combinará ambos aspectos, proporcionando un equilibrio entre precisión y recall. Además, se utilizará la

matriz de confusión para analizar los errores y ajustar los modelos según sea necesario. Estas métricas permitirán medir de forma efectiva el rendimiento de los modelos en un contexto de datos desbalanceados.

Este proyecto no solo busca crear un sistema funcional de detección de fraudes, sino también explorar las mejores prácticas en el manejo de datos desequilibrados y la interpretación de modelos en un contexto de alta sensibilidad como es la detección de fraudes financieros.

1.2: Descripción del Problema

El objetivo de este proyecto es predecir si una operación es fraudulenta (clase 1) o no (clase 0), lo cual es crucial para reducir pérdidas económicas y mejorar la seguridad de los sistemas de pago.

Comenzaremos con un exhaustivo análisis exploratorio de datos (EDA). Primero, cargaremos nuestra muestra seleccionada y analizaremos cada variable en detalle, revisando sus tipos y realizando los ajustes que sean necesarios. Este paso es esencial para comprender la estructura de nuestros datos, identificar patrones, detectar anomalías y explorar las relaciones entre las diferentes características.

Durante el EDA, nos centraremos en el análisis de las variables, así como en la identificación y tratamiento de valores atípicos. También analizaremos cómo varían las transacciones con el paso del tiempo. Para ello, utilizaremos el apoyo de gráficos que nos permitirán representar los patrones complejos de los datos de una forma más visual y sencilla, lo que nos ayudará a obtener la información valiosa sobre la naturaleza de las transacciones, tanto fraudulentas como legítimas.

En cuanto a la implementación, ésta incluirá la carga y preprocesamiento de los datos, seguido por una exploración y análisis inicial. Posteriormente, se procederá a la construcción de los modelos predictivos, su evaluación y la selección del mejor modelo en función de su rendimiento.

En un entorno real, anticipamos varios desafíos, como el desequilibrio extremo de clases, la rápida evolución de los patrones de fraude, la gestión de falsos positivos y el consiguiente descontento de los clientes y la necesidad de detección en tiempo real, así como las consideraciones de privacidad y seguridad de datos. Para enfrentar estos retos, pondríamos en marcha la implementación de sistemas de reentrenamiento periódico, el establecimiento procesos de revisión humana para casos dudosos y la optimización de nuestros modelos para la predicción en tiempo real, asegurándonos de que cumplimos con las regulaciones estrictas de privacidad.

1.3: Pre-proceso

Carga de los datos

Tras cargar las librerías necesarias, cargamos el conjunto de datos de transacciones con tarjetas de crédito.

```r
fraud <- read_csv("creditcard.csv")
dim(fraud)
```

```
[1] 284807     31
```

Muestreo estratificado

Para empezar, buscamos reducir la complejidad computacional, ya que es una base de datos con muchas observaciones (284807). Seleccionamos una muestra aleatoria de 10000 observaciones usando un muestreo estratificado, es decir, dividimos el conjunto de observaciones en subconjuntos según si son de clase 0 o 1, garantizándonos el mismo porcentaje de casos de cada tipo que en la base de datos original (fijación proporcional). Esto es esencial porque aunque sea mucho menos habitual que un movimiento sea fraudulento, necesitamos que el modelo aprenda y evalúe teniendo en cuenta que estos casos existen, pues son precisamente los que nos interesa encontrar. Dentro de cada estrato escogemos al azar, usando set.seed(1313) para garantizar la reproducibilidad, lo que permite que futuros análisis con el mismo código generen la misma muestra aleatoria. Como comprobamos al final, la muestra y la base de datos original guarda la misma estructura según la Clase.

```r
prop_original <- sum(fraud$Class==1)/nrow(fraud)

prop_0 <- sum(fraud$Class == 0) / nrow(fraud)
prop_1 <- sum(fraud$Class == 1) / nrow(fraud)
prop_0+prop_1==1
```

```
[1] TRUE
```

```r
muestra_clase_1 <- round(prop_1 * 10000)
muestra_clase_0 <- round(prop_0 * 10000)
muestra_clase_1+muestra_clase_0==10000
```

```
[1] TRUE
```

```r
set.seed(1313)
muestra_clase_1 <- fraud %>%
  filter(Class == 1) %>%
  sample_n(muestra_clase_1, replace = TRUE)

set.seed(1313)
muestra_clase_0 <- fraud %>%
  filter(Class == 0) %>%
  sample_n(muestra_clase_0, replace = FALSE)

fraud <- bind_rows(muestra_clase_0, muestra_clase_1)

table(fraud$Class)
```

```
   0    1
9983   17

prop_nueva <- sum(fraud$Class==1)/nrow(fraud)
round(prop_original,4)==prop_nueva

[1] TRUE

dim(fraud)

[1] 10000    31
```

Después, verificamos si existen valores faltantes en el conjunto de datos mediante la función anyNA(fraud), en este caso no los hay, así que no hace falta tratarlos. Guardamos la muestra del conjunto de datos en un archivo .RData para usos posteriores.

```
anyNA(fraud)

[1] FALSE

save(fraud, file = "fraud.RData")
```

Variable factor

La variable Class es la variable predictora en este conjunto de datos, indicando si una transacción es fraudulenta o no. Originalmente es numérica, pero como representa categorías (0 transacción normal, 1 fraude), la convertimos en factor para su uso correcto en modelos de clasificación.

```
load("fraud.RData")

class(fraud$Class)

[1] "numeric"

fraud$Class <- as.factor(fraud$Class)

class(fraud$Class)

[1] "factor"
```

Correlaciones altas

Ahora empezamos a analizar las relaciones entre los predictores. Primero, analizamos la correlación, una medida estadística que mide la fuerza y dirección de la relación entre dos variables. Es útil detectar correlaciones altas entre las variables, ya que las variables fuertemente correlacionadas pueden causar problemas como multicolinealidad. De todas formas, esperamos que sean bajas ya que las variables provienen de un análisis de Componentes Principales y esperamos que se haya eliminado toda la redundancia.

```
corr <- abs(cor(fraud[,1:30])[upper.tri(cor(fraud[,1:30]))]) > .9
corr_altos <- sum(corr)
corr_altos

[1] 0
```

Como esperábamos, no encontramos correlaciones mayores a 0.9, lo que significa que no hay variables con relaciones lineales extremadamente fuertes. Esto es positivo, ya que el riesgo de multicolinealidad y de sobreajuste en el modelo es bajo.

Dependencias lineales

Ahora evaluamos si alguna de las variables puede ser una combinación lineal exacta de otras variables, lo cual puede causar problemas en ciertos modelos. Si una variable es una combinación lineal de otras, podríamos estar introduciendo redundancia en el modelo, lo que afectaría la interpretabilidad y podría hacer que algunos algoritmos no funcionen correctamente, pues la información se duplicaría.

```
findLinearCombos(fraud)

$linearCombos
list()

$remove
NULL
```

No hemos encontrado combinaciones lineales exactas (multicolinealidad extrema), lo que respalda los resultados anteriores de que no hay correlaciones excesivamente altas entre las variables, ya que las variables provienen de un proceso PCA. Podemos concluir que cada una de las variables aporta información única, es decir, maximizan la variabilidad.

Tipificación

Nos puede interesar llevar a cabo una tipificación o estandarización, es decir, el proceso de transformar las variables para que tengan una media de 0 y una desviación estándar de 1. Este procedimiento es particularmente útil para modelos que asumen que las variables tienen una distribución normal, como es el caso de Análisis Discriminante Lineal (LDA) y otros modelos lineales.

Para verificar que la tipificación se haya realizado correctamente, comprobamos que cada variable tenga una media cercana a 0 y una desviación estándar de 1. Esto asegura que las variables estén en una escala similar y que ninguna variable tenga un impacto desproporcionado debido a su escala original.

```
fraud_tip <- fraud

for (i in 1:30) {
  fraud_tip[[i]] <- (fraud[[i]] - mean(fraud[[i]], na.rm = TRUE)) /
sd(fraud[[i]], na.rm = TRUE)
```

```
}

head(fraud_tip,2)
```

```
# A tibble: 2 × 31
    Time     V1    V2     V3     V4    V5     V6    V7     V8     V9
V10
   <dbl>  <dbl> <dbl>  <dbl>  <dbl> <dbl>  <dbl> <dbl>  <dbl>  <dbl>
<dbl>
1  0.269 -0.246 0.529 -0.190 -0.131 0.168 -0.636 1.18  -0.507  1.39  -
0.285
2 -1.16  -0.270 0.502  0.574  0.166 0.768 -0.106 0.937 -0.258 -0.713
0.158
# ℹ 20 more variables: V11 <dbl>, V12 <dbl>, V13 <dbl>, V14 <dbl>, V15
<dbl>,
#   V16 <dbl>, V17 <dbl>, V18 <dbl>, V19 <dbl>, V20 <dbl>, V21 <dbl>,
#   V22 <dbl>, V23 <dbl>, V24 <dbl>, V25 <dbl>, V26 <dbl>, V27 <dbl>,
#   V28 <dbl>, Amount <dbl>, Class <fct>
```

```
comp_media <- numeric(30)
for (i in 1:30) {
  comp_media[i] <- mean(fraud_tip[[i]])
}
round(comp_media,2)
```

```
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
comp_sd <- numeric(30)
for (i in 1:30) {
  comp_sd[i] <- sd(fraud_tip[[i]])
}
comp_sd
```

```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Normalización a [0,1]

En algunos casos, también puede ser conveniente normalizar las variables a un rango de [0, 1], lo que implica una transformación diferente de las variables. Este tipo de ajuste es particularmente útil para modelos que dependen de las distancias, como las Redes Neuronales o K Vecinos más Cercanos (KNN).

La normalización a [0, 1] es especialmente relevante cuando las variables tienen unidades muy diferentes entre sí, como ocurre con la variable Amount (medida en euros) en comparación con las demás. La desviación estándar de esta variable es considerablemente mayor que la de otras variables, lo que podría dar lugar a que domine el modelo si no se ajusta.

En este proceso de normalización, transformamos las variables de manera que todas estén en un rango común de [0, 1], evitando que variables con escalas más grandes

(como Amount) influyan más de lo debido. Este paso ayuda a que todas las variables tengan un peso similar al momento de ser introducidas en el modelo.

```
round(apply(fraud[,2:30], 2, sd),2)
```

```
    V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
V11
  2.05   1.87   1.54   1.44   1.41   1.35   1.23   1.08   1.10   1.10
1.01
   V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
V22
  0.98   1.00   0.91   0.92   0.86   0.79   0.84   0.82   0.92   0.68
0.72
   V23    V24    V25    V26    V27    V28 Amount
  0.59   0.60   0.53   0.48   0.42   0.38 302.61
```

```
fraud_nor <- fraud

for (i in 1:30) {
    fraud_nor[[i]] <- (fraud[[i]] - min(fraud[[i]], na.rm = TRUE)) /
(max(fraud[[i]], na.rm = TRUE) - min(fraud[[i]], na.rm = TRUE))
  }

head(fraud_nor, 2)
```

```
# A tibble: 2 × 31
   Time    V1    V2    V3    V4    V5    V6    V7    V8    V9   V10   V11
V12
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
<dbl>
1 0.628 0.923 0.798 0.869 0.218 0.373 0.568 0.584 0.681 0.635 0.634 0.323
0.599
2 0.236 0.922 0.798 0.903 0.238 0.389 0.586 0.579 0.690 0.502 0.648 0.423
0.793
# ℹ 18 more variables: V13 <dbl>, V14 <dbl>, V15 <dbl>, V16 <dbl>, V17
<dbl>,
#   V18 <dbl>, V19 <dbl>, V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>,
#   V24 <dbl>, V25 <dbl>, V26 <dbl>, V27 <dbl>, V28 <dbl>, Amount <dbl>,
#   Class <fct>
```

```
round(apply(fraud_nor[,2:30], 2, sd),2)
```

```
    V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
V11
  0.05   0.02   0.04   0.07   0.03   0.03   0.02   0.04   0.06   0.03
0.09
   V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
V22
  0.06   0.12   0.06   0.12   0.07   0.04   0.08   0.07   0.02   0.03
0.06
```

```
   V23    V24    V25    V26    V27    V28 Amount
  0.02   0.10   0.04   0.09   0.02   0.02   0.02
```

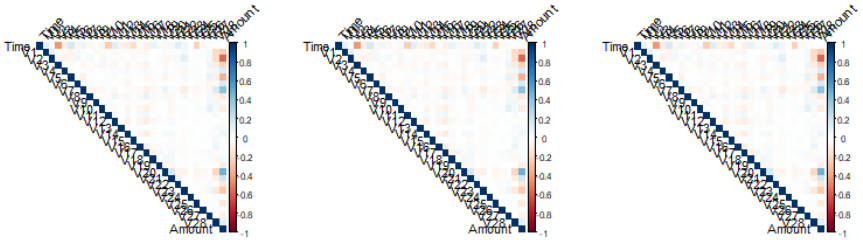
Al finalizar la normalización, todas las variables presentan desviaciones estándar más similares, lo que asegura que ninguna variable esté desbalanceada en términos de su influencia en el modelo.

Comprobación

Finalmente, utilizamos gráficos de correlación para comparar las relaciones entre las variables en las distintas versiones de los datos:

• Datos originales. • Datos normalizados a [0,1]. • Datos tipificados (media 0, desviación estándar 1).

Los gráficos de correlación nos permiten ver visualmente cómo las relaciones entre las variables se mantienen consistentes incluso después del preprocesamiento. Esto es importante para asegurarse de que el escalado y la estandarización no han distorsionado las relaciones entre las variables.



1.4: Initial Exploratory Data Analysis

Hemos decidido tipificar las variables para este estudio, pues los modelos KNN y Redes Neuronales dependen de distancias y pesos. Si las variables tienen escalas diferentes, algunas pueden dominar el modelo, sesgando los resultados. La tipificación asegura que todas las variables tengan el mismo peso. En el caso de los árboles de Decisión, no se requiere tipificación, ya que dividen las observaciones en función de umbrales de las variables, sin importar su escala. Sin embargo, tipificar puede ayudar a mejorar la coherencia de las divisiones en los nodos.

División en training set y test set

Ahora dividimos los datos en training set y test set, usamos createDataPartition del paquete caret para hacer la división asegurándonos de que ambos conjuntos mantengan la misma proporción de Clase, pues necesitamos que se tenga en cuenta en la construcción del modelo y de su análisis de capacidad predictiva.

```r
set.seed(1313)
trainIndex <- createDataPartition(fraud_tip$Class, p = .8,
                                  list = FALSE,
                                  times = 1)

fraud_Train <- fraud_tip[ trainIndex,]
fraud_Test  <- fraud_tip[-trainIndex,]

prop_train <- sum(fraud_Train$Class==1)/nrow(fraud_Train)
prop_test <- sum(fraud_Test$Class==1)/nrow(fraud_Test)

round(prop_train,3)==round(prop_test,3)
```

```
[1] TRUE
```

```r
round(prop_test,3)==round(prop_original,3)
```

```
[1] TRUE
```

A partir de aquí nos olvidamos completamente del conjunto de test, y trabajamos íntegramente con el training set.

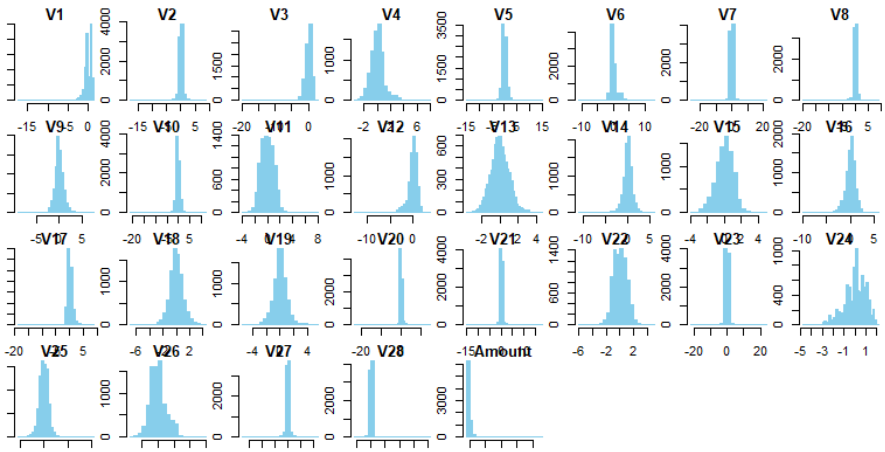
Distribuciones de las variables

Habiendo dividido los datos, analizamos las distribuciones de las variables en el conjunto de datos para comprender mejor sus características.

```r
par(mfrow = c(4, 8), mar = c(1, 1, 1, 1))

for (var in names(fraud_Train)[2:30]) {
  hist(fraud_Train[[var]], main = paste(var),
       xlab = var, col = "skyblue", border = "skyblue", breaks = 30)
}

par(mfrow = c(1, 1))
```



Parece que la mayoría de las variables tienen distribuciones bastante simétricas, aunque hay algunas con sesgo claro a la izquierda o derecha. Varias variables muestran picos en valores específicos, estas concentraciones podrían ser indicativas de comportamientos recurrentes en transacciones normales o fraudulentas. La variabilidad entre las distribuciones es alta, lo que sugiere que cada variable nos aporta información única al modelo predictivo, lo cual tiene sentido teniendo en cuenta el proceso de PCA previo.

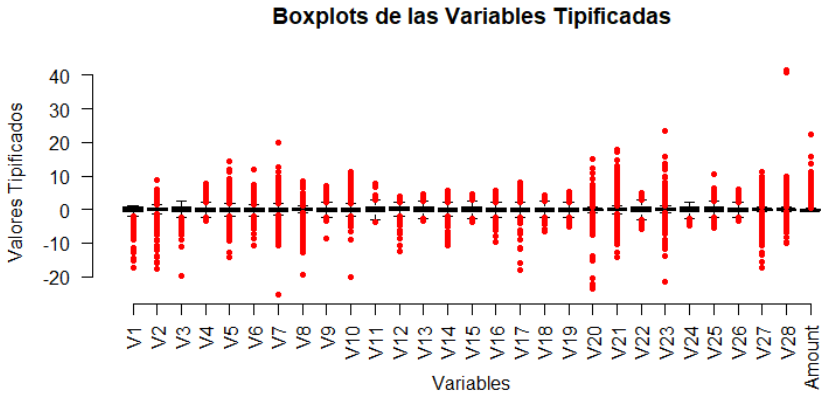
Es relevante observar que algunos histogramas revelan la presencia de posibles outliers, puntos que se encuentran muy alejados de la distribución general, los cuales hemos decidido estudiar más a fondo. La identificación de estos valores atípicos es crucial, ya que pueden tener un gran impacto en la precisión del modelo de detección de fraudes.

Detección de outliers

Los outliers Los outliers (o valores atípicos) son datos que se desvían de manera significativa del resto de las observaciones en un conjunto de datos, y pueden representar anomalías o comportamientos inusuales, esencial para la detección de fraudes. Estadísticamente son las observaciones tres veces mayores en valores absolutos a la desviación típica. Los vemos primero en un plot.

```
boxplot(fraud_Train[ , 2:30],
        col = "skyblue",
        outcol = "red",
        pch = 20,
        frame.plot = FALSE,
        main = "Boxplots de las Variables Tipificadas",
        ylab = "Valores Tipificados",
        xlab = "Variables",
        las = 2)

outliers_count <- sapply(fraud_Train[, 2:30],
                         function(x) sum(x < (mean(x, na.rm = TRUE) - 3 *
sd(x, na.rm = TRUE)) |
                                         x > (mean(x, na.rm = TRUE) + 3
* sd(x, na.rm = TRUE))))
```



Vemos ahora el número total de outliers por variable, y qué porcentaje representan sobre el total de observaciones.

```
outliers_count
```

```
    V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
V11
   102    117     56     85     80    120     93    154     61     90
19
   V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
V22
   109     39    104     33     58     85     56    110    120    129
33
   V23    V24    V25    V26    V27    V28 Amount
   116     13     79     28    141     59    123
```

```
round(outliers_count/nrow(fraud_Train)*100,2)
```

```
    V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
V11
  1.27   1.46   0.70   1.06   1.00   1.50   1.16   1.92   0.76   1.12
0.24
   V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
V22
  1.36   0.49   1.30   0.41   0.72   1.06   0.70   1.37   1.50   1.61
0.41
   V23    V24    V25    V26    V27    V28 Amount
  1.45   0.16   0.99   0.35   1.76   0.74   1.54
```

La mayoría de las variables tienen un porcentaje de 0-2% de outliers, con V8 y V27 mostrando el mayor número con 1.92% (154) y 1.76% (141) respectivamente. Este porcentaje es muy pequeño, y generalmente no debería tener un gran impacto en los modelos predictivos, especialmente con una cantidad de datos alta. De todas formas, en el ejemplo específico que estamos estudiando, el efecto que los outliers puedan tener sobre el modelo es especialmente interesante porque los fraudes tienden a tener características diferentes o atípicas respecto a las transacciones normales. En este sentido, como ya hemos mencionado, los outliers podrían representar precisamente

esos comportamientos inusuales que estamos intentando identificar. Por lo tanto, estudiamos de estos outliers cuáles se relacionan con fraudes.

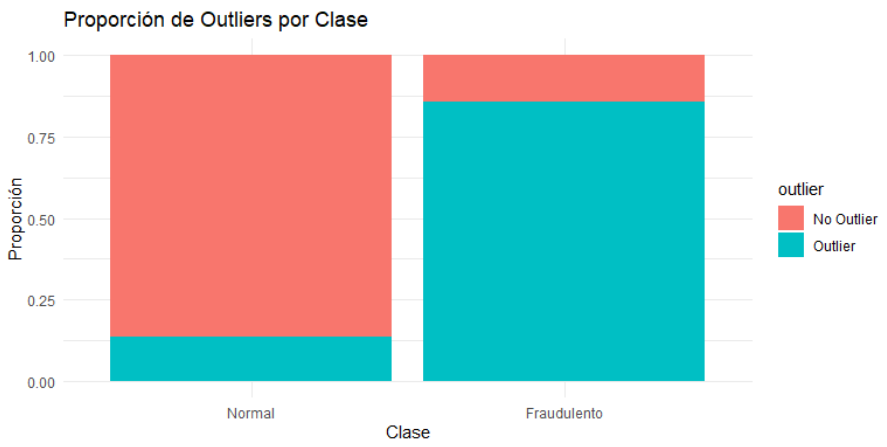
```
outlier_class <- apply(fraud_Train[ , 2:30], 1, function(row) {
  any(row < -3 | row > 3)
})

fraud_Train %>%
  mutate(outlier_class = outlier_class) %>%
  group_by(Class, outlier_class) %>%
  summarise(outliers = n()) %>%
  mutate(pct = outliers / sum(outliers))

# A tibble: 4 × 4
# Groups:   Class [2]
  Class outlier_class outliers   pct
  <fct> <lgl>            <int> <dbl>
1 0     FALSE             6911 0.865
2 0     TRUE              1076 0.135
3 1     FALSE                2 0.143
4 1     TRUE                12 0.857
```

Concluimos que, en efecto, los outliers juegan un papel crucial en la detección de casos fraudulentos. Un 86% de los outliers se corresponden con transacciones fraudulentas, lo que resalta su importancia como posibles indicativos de fraude. Este hallazgo refuerza la necesidad de prestar especial atención a los outliers, ya que no solo representan datos atípicos, sino que también pueden ser una señal clave para identificar comportamientos fraudulentos dentro del conjunto de datos. Podemos verlo de forma muy clara en el siguiente plot:

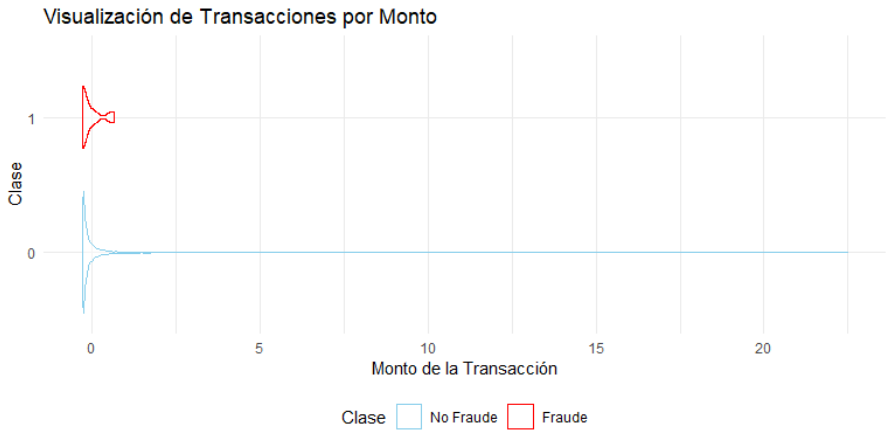
```
fraud_Train %>%
  mutate(outlier = ifelse(rowSums(fraud_Train[,2:30] > 3 |
fraud_Train[,2:30] < -3) > 0, "Outlier", "No Outlier"),  Class =
recode(Class, `0` = "Normal", `1` = "Fraudulento")) %>%
  ggplot(aes(x = Class, fill = outlier)) +
  geom_bar(position = "fill") +
  labs(title = "Proporción de Outliers por Clase", y = "Proporción", x =
"Clase") +
  theme_minimal()
```



Relación con la variable objetivo

En este análisis, exploramos la relación entre el monto de la transacción (Amount) y la clase de la transacción (Class) usando un gráfico de violín, muy útil para estudiar la distribución de datos numéricos categorizados por clases.

```
ggplot(fraud_Train, aes(x = Amount, y = Class, color = as.factor(Class)))
+
  geom_violin(alpha = 0.5) +
  scale_color_manual(values = c("skyblue", "red"),
                     labels = c("No Fraude", "Fraude")) +
  labs(title = "Visualización de Transacciones por Monto",
       x = "Monto de la Transacción",
       y = "Clase",
       color = "Clase") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



Observamos que las transacciones fraudulentas (en rojo) tienden a concentrarse en montos bajos, mientras que las transacciones no fraudulentas (en azul) están distribuidas a lo largo de un rango mucho más amplio de montos, aunque también se concentra bastante en montos bajos. Esto sugiere que el monto de la transacción podría ser una

característica importante para la detección de fraudes, ya que los fraudes parecen estar más relacionados con transacciones de menor valor.

Dado que la relación entre el monto y la clase no es completamente lineal, vemos si discretizar la variable Amount mejora la claridad de la relación entre ambas clases según el monto.

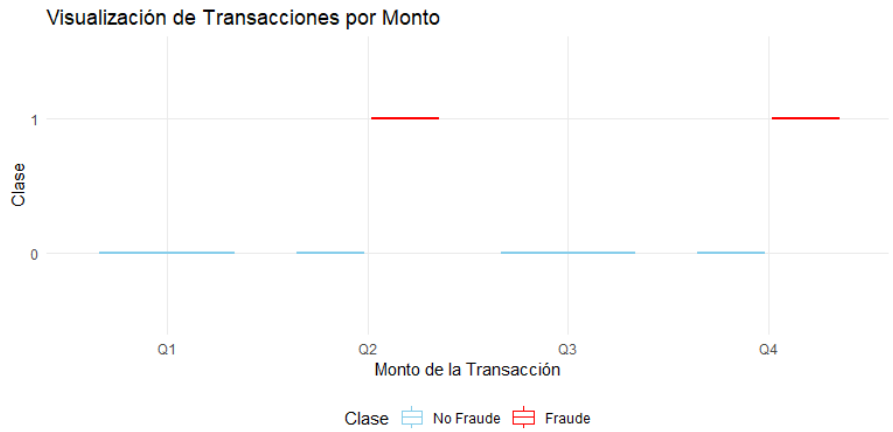
```
fraud_Train2 <- fraud_Test

Amount_discreta <-quantile(fraud_Train2$Amount, probs = seq(0, 1, by =
0.25))

fraud_Train2$Amount <- cut(fraud_Train2$Amount,
                           breaks = Amount_discreta,
                           include.lowest = TRUE,
                           labels = c("Q1", "Q2", "Q3", "Q4"))


ggplot(fraud_Train2, aes(x = Amount, y = Class, color =
as.factor(Class))) +
  geom_boxplot(alpha = 0.5) +
  scale_color_manual(values = c("skyblue", "red"),
                     labels = c("No Fraude", "Fraude")) +
  labs(title = "Visualización de Transacciones por Monto",
       x = "Monto de la Transacción",
       y = "Clase",
       color = "Clase") +
  theme_minimal() +
  theme(legend.position = "bottom")

rm(fraud_Train2)
```



Tras discretizar la variable "Amount" en cuartiles, observamos que las transacciones fraudulentas (en rojo) tienden a concentrarse en montos medianos y altos. Esto sugiere que los fraudes están más asociados con valores elevados. En contraste, las transacciones normales (en azul) se distribuyen más uniformemente a través de todos

los rangos de monto, especialmente en los valores bajos y medianos-altos. La discretización de la variable "Amount" facilita la visualización de estas diferencias y resalta la importancia de esta variable para detectar fraudes.

Correlaciones con la variable objetivo

Estudiamos las correlaciones entre las variables y la variable objetivo Clase, que nos permiten identificar cuáles son más relevantes para predecir el fraude. Si una variable tiene una alta correlación (positiva o negativa) con la clase, es un indicio de que esa variable puede ser un buen predictor.

```
correlations<-numeric(29)

for (i in 2:30) {
  correlations[i-1] <- cor(fraud_Train[[i]],
as.numeric(fraud_Train$Class), method = "pearson")
}
round(correlations,2)
```

```
 [1] -0.07  0.07 -0.13  0.09 -0.06 -0.04 -0.13 -0.03 -0.07 -0.16  0.11 -
0.20
[13] -0.01 -0.24  0.00 -0.11 -0.21 -0.06  0.01  0.01 -0.01  0.01  0.00
0.00
[25] -0.01 -0.01 -0.06 -0.01  0.00
```

Aunque algunas variables parecen tener cierta relación con la clase como V7 (0.16), V11 (0.24) y V14 (0.21), la correlación no es lo suficientemente fuerte como para hacer predicciones claras solo a partir de ellas. Esto resalta la importancia de utilizar métodos de modelado más avanzados (como los que estamos considerando, KNN, árboles de decisión y redes neuronales), que no dependen exclusivamente de la correlación lineal para hacer predicciones.

Distribuciones según la clase

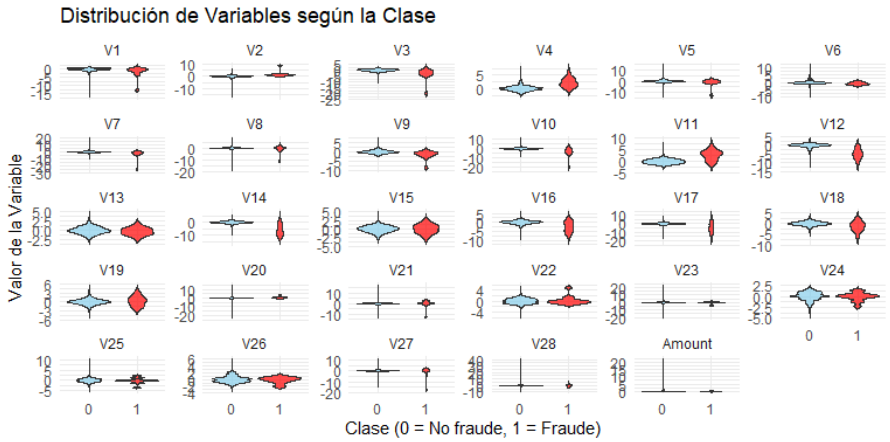
Por último, nos interesa analizar cómo se distribuyen las variables dentro de cada clase. Si observamos que una variable tiene una distribución muy diferente entre las transacciones fraudulentas y no fraudulentas, podemos deducir que esa variable es interesante para la clasificación.

```
fraud_long <- fraud_Train %>%
  pivot_longer(cols = 2:30, names_to = "Variable", values_to = "Value")

order_of_variables <- names(fraud_Train)[2:31]
fraud_long$Variable <- factor(fraud_long$Variable, levels =
order_of_variables)

ggplot(fraud_long, aes(x = Class, y = Value, fill = Class)) +
  geom_violin(trim = FALSE, alpha = 0.7) +
  facet_wrap(~ Variable, scales = "free_y") +
  theme_minimal() +
```

```
  labs(title = "Distribución de Variables según la Clase",
       x = "Clase (0 = No fraude, 1 = Fraude)",
       y = "Valor de la Variable") +
  scale_fill_manual(values = c("skyblue", "red")) +
  theme(legend.position = "none")
```



Al analizar los gráficos de violín, se observan diferencias significativas en la distribución de las transacciones fraudulentas y no fraudulentas. Variables como V3, V4, V12, V14, V16 y V17 muestran claras disparidades en términos de densidad y rango entre las clases, lo que sugiere que podrían ser relevantes para la detección de fraudes. En contraste, variables como V2, V13, V15 y V26 no muestran diferencias evidentes entre las transacciones fraudulentas y no fraudulentas.

1.5: Conclusión

En conclusión, nuestro análisis de las transacciones financieras para la detección de fraudes revela que, a pesar de la desbalanceada distribución de clases, se pueden identificar patrones relevantes que permiten distinguir entre transacciones normales y fraudulentas. La estandarización y normalización de las variables, junto con el análisis de outliers y correlaciones, han sido cruciales para preparar los datos y minimizar el impacto de posibles sesgos. Las variables como V7, V11 y V14, junto con el monto de la transacción, emergieron como factores clave en la clasificación de fraudes. Este proceso de limpieza y transformación de datos, junto con el enfoque en la relación entre variables y la clase objetivo, establece una base sólida para el desarrollo de modelos predictivos más precisos y eficientes en la detección de fraudes.

1.6: Referencias

Credit card fraud Detection. (2018, March 23). https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data

Kuhn, M. (2019, March 27). 4 Data Splitting | The caret Package. https://topepo.github.io/caret/data-splitting.html#simple-splitting-with-important-groups

Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015

Dal Pozzolo, Andrea; Caelen, Olivier; Le Borgne, Yann-Ael; Waterschoot, Serge; Bontempi, Gianluca. Learned lessons in credit card fraud detection from a practitioner perspective, Expert systems with applications,41,10,4915-4928,2014, Pergamon

Dal Pozzolo, Andrea; Boracchi, Giacomo; Caelen, Olivier; Alippi, Cesare; Bontempi, Gianluca. Credit card fraud detection: a realistic modeling and a novel learning strategy, IEEE transactions on neural networks and learning systems,29,8,3784-3797,2018,IEEE

Dal Pozzolo, Andrea Adaptive Machine learning for credit card fraud detection ULB MLG PhD thesis (supervised by G. Bontempi)

Carcillo, Fabrizio; Dal Pozzolo, Andrea; Le Borgne, Yann-Aël; Caelen, Olivier; Mazzer, Yannis; Bontempi, Gianluca. Scarff: a scalable framework for streaming credit card fraud detection with Spark, Information fusion,41, 182-194,2018,Elsevier

Carcillo, Fabrizio; Le Borgne, Yann-Aël; Caelen, Olivier; Bontempi, Gianluca. Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization, International Journal of Data Science and Analytics, 5,4,285-300,2018,Springer International Publishing

Bertrand Lebichot, Yann-Aël Le Borgne, Liyun He, Frederic Oblé, Gianluca Bontempi Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection, INNSBDDL 2019: Recent Advances in Big Data and Deep Learning, pp 78-88, 2019

Fabrizio Carcillo, Yann-Aël Le Borgne, Olivier Caelen, Frederic Oblé, Gianluca Bontempi Combining Unsupervised and Supervised Learning in Credit Card Fraud Detection Information Sciences, 2019

Yann-Aël Le Borgne, Gianluca Bontempi Reproducible machine Learning for Credit Card Fraud Detection - Practical Handbook

Bertrand Lebichot, Gianmarco Paldino, Wissam Siblini, Liyun He, Frederic Oblé, Gianluca Bontempi Incremental learning strategies for credit cards fraud detection, IInternational Journal of Data Science and Analytics