UNIVERSIDAD DE GRANADA

Sistemas Inteligentes para la Gestión en la Empresa

Pre-procesamiento de datos y clasificación binaria

Práctica 1

Víctor Vázquez Rodríguez
victorvazrod@correo.ugr.es

Máster Universitario en Ingeniería Informática

Curso 2019/20

Índice

1 Introducción

En esta práctica, se debe abordar un problema de clasificación binaria relativo a la detección de fraudes en transacciones por internet. Para ello, se usan los conjuntos de datos proporcionados en una competición de kaggle sobre este tema organizada por el IEEE-CIS[1].

A lo largo de este documento, se analizará el conjunto de datos inicial y se preparará el mismo para aplicar los distintos clasificadores. Al final, se comentarán los resultados obtenidos.

La explicación de las distintas tareas realizadas sobre el conjunto de datos se apoyará con fragmentos del código escrito en R para tal fin. El código completo se puede ver en el repositorio de GitHub creado para la práctica[2].

[1] Competición en kaggle
[2] Repositorio de GitHub del proyecto

2 Exploración

Los datos proporcionados vienen en dos tablas separadas:

- La tabla *identity*, que recoge información sobre los usuarios que realizan las transacciones (tipo de dispositivo, dirección, ...).
- La tabla *transaction*, que es la que tiene las transacciones como tales (tiempo, cantidad de dinero, producto comprado, tarjeta usada, ...), además de la clase objetivo *isFraud*.

Ambas tablas tienen en común un atributo *TransactionID*, que usamos para unirlas. Hay que tener en cuenta que no todas las transacciones de la tabla *transaction* tienen una identidad asociada, por lo que la unión que realizamos es de tipo *innerjoin*, quedándonos solo con los registros de transacciones que tienen identidad.

```
library(tidyverse)

if (!file.exists("data/train_innerjoin.csv") ||
    !file.exists("data/test_innerjoin.csv")) {
  # Load train and test source datasets
  train_transaction <- read_csv("data/train_transaction.csv")
  train_identity <- read_csv("data/train_identity.csv")
  test_transaction <- read_csv("data/test_transaction.csv")
  test_identity <- read_csv("data/test_identity.csv")

  # Merge tables
  train_dataset <- merge(train_transaction, train_identity, by =
      "TransactionID")
  test_dataset <- merge(test_transaction, test_identity, by =
      "TransactionID")

  # Write merged datasets
  write_csv(train_dataset, "data/train_innerjoin.csv")
  write_csv(test_dataset, "data/test_innerjoin.csv")
} else {
  # Load train and test datasets
  train_dataset <- read_csv("data/train_innerjoin.csv")
  test_dataset <- read_csv("data/test_innerjoin.csv")
}
```

Como se puede apreciar en el código encargado de leer los datos y unir las tablas, tenemos datos tanto de *train* como de *test*. No obstante, los datos de *test* no contienen la clase objetivo, ya que están pensados para usarse en la competición de kaggle enviando los resultados obtenidos de las predicciones realizadas sobre este conjunto y obteniendo las métricas de calidad de la predicción. Como la competición ya ha finalizado, este conjunto no nos es realmente útil, por lo que a partir de ahora trabajaremos solo sobre el de *train*.

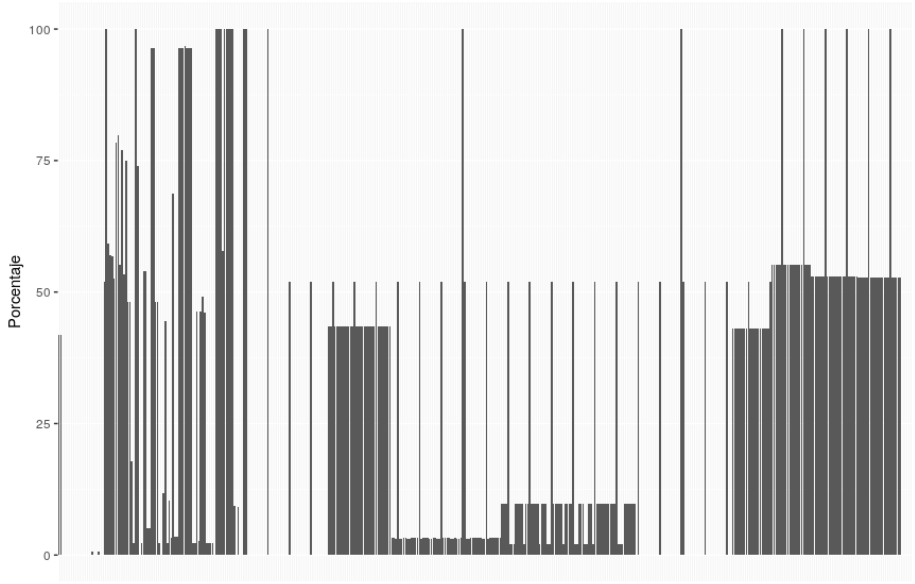

Figura 1: Porcentaje de valores NA por variable.

En este conjunto de datos tenemos un total de 434 variables y 144233 registros tras la unión de las tablas. Son muchísimos datos, por lo que tendremos que realizar una extracción de variables para reducir su número. Si no hiciéramos esto, los tiempos de ejecución y requisitos de memoria serían demasiado altos. Entre estas variables tenemos la clase objetivo *isFraud*, dos variables continuas (*TransactionDT* y *TransactionAmt*) que tendremos que convertir a intervalos y una serie de variables que sabemos que son categóricas gracias a que así lo indica la competición en kaggle, ya que observando los datos podrían parecer numéricas. Cabe destacar que, debido al carácter sensible de los datos, el significado de muchas de las variables está enmascarado y, por lo tanto, no podemos conocerlo.

Para conocer más información sobre los datos, usamos la función `df_status()` del paquete *funModeling*, con la que podemos ver el porcentaje de valores NA y la dispersión de las variables del conjunto de datos. En la figura 1, podemos ver que hay una gran cantidad de NAs, habiendo variables con más de un 50% de estos valores. Todas estas variables que tienen muchos valores vacíos podremos eliminarlas en favor de otras que tienen más información. También podemos ver, en la figura 2, el porcentaje de dispersión de los valores de las distintas variables. Se puede apreciar como hay una variable con una dispersión extremadamente alta, que llega hasta el 80%. Si quitamos esta variable, en la figura 3 vemos que también tenemos variables con una dispersión muy baja, incluso llegando al 0%. Todas estas variables de dispersión elevada o muy baja las podremos eliminar del conjunto de datos.

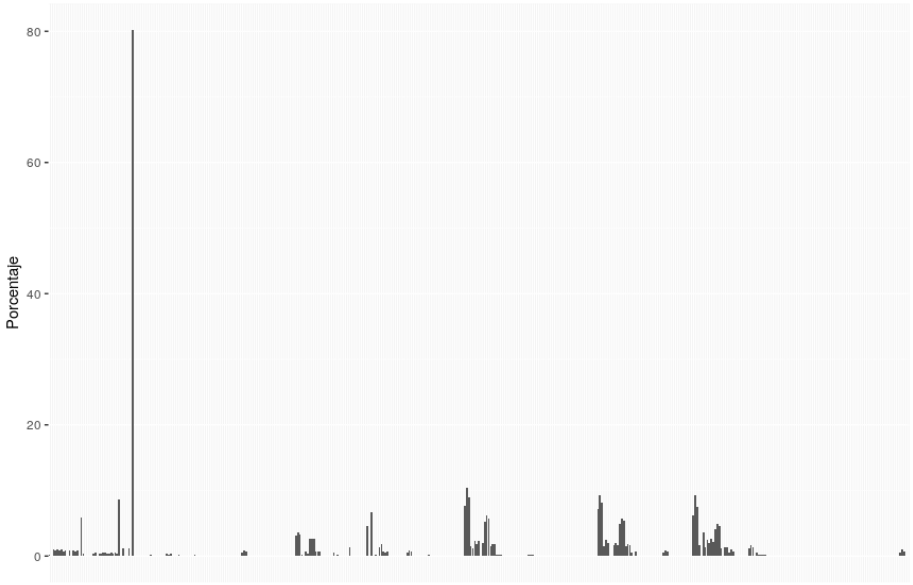

Figura 2: Porcentaje de dispersión de valores.

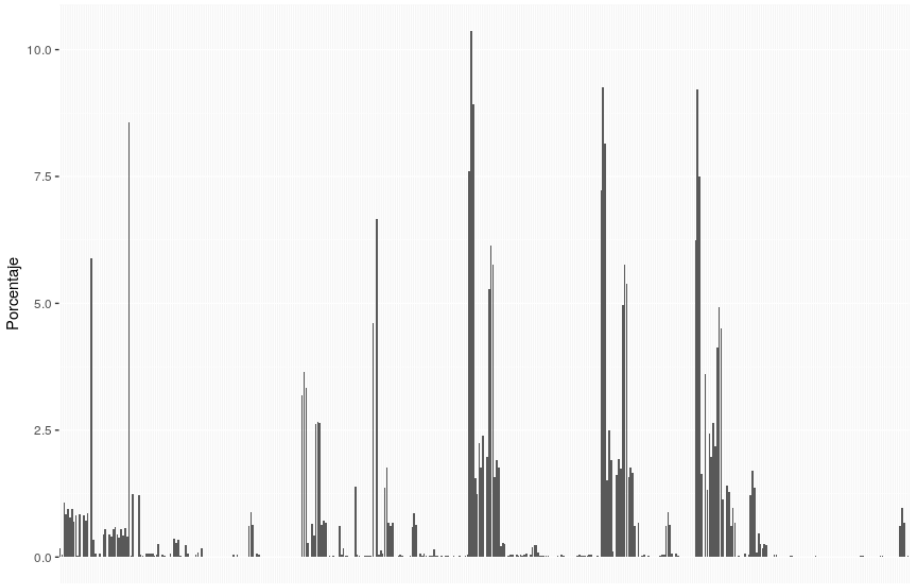

Figura 3: Porcentaje de dispersión de valores para variables con dispersión inferior al 60%.

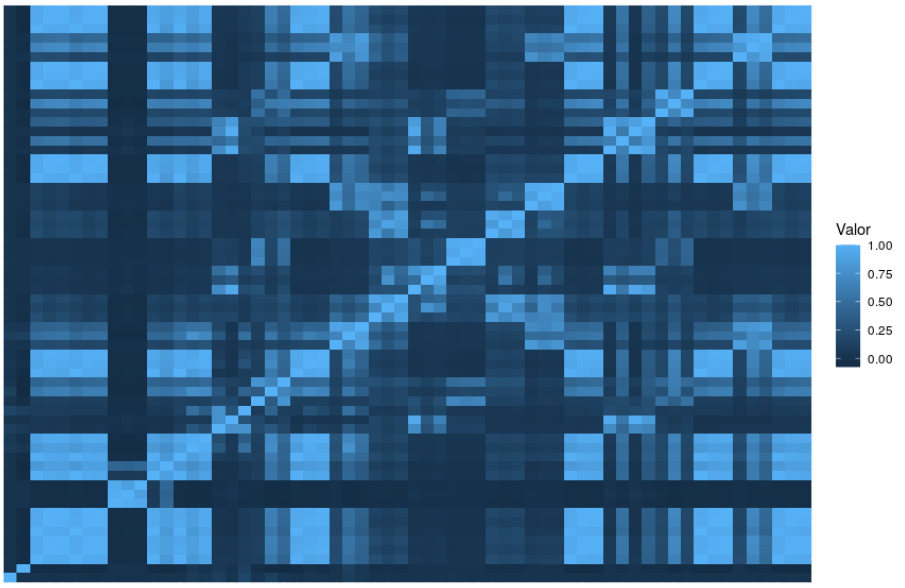

Figura 4: Correlación entre variables numéricas.

También podemos fijarnos en la correlación entre variables para ver si hay algunas que podamos eliminar. Si dos variables están muy correladas, podremos eliminar una de ellas, ya que ambas nos aportarían la misma información. En la figura 4 tenemos un mapa de calor que refleja las correlaciones entre las variables de tipo numérico del conjunto de datos. Las zonas con un tono más claro indican una alta correlación, por lo que serían pares de variables a considerar.

Para terminar con la exploración, podemos observar el número de registros que tenemos para cada valor de la clase objetivo *isFraud*, lo cuál podemos ver en la figura 5. Como era de esperar, hay muchos más ejemplos de transacciones que no son fraude que de las que sí lo son, ya que estas últimas son menos frecuentes. Para entrenar los clasificadores, necesitaremos balancear el conjunto de datos primero.

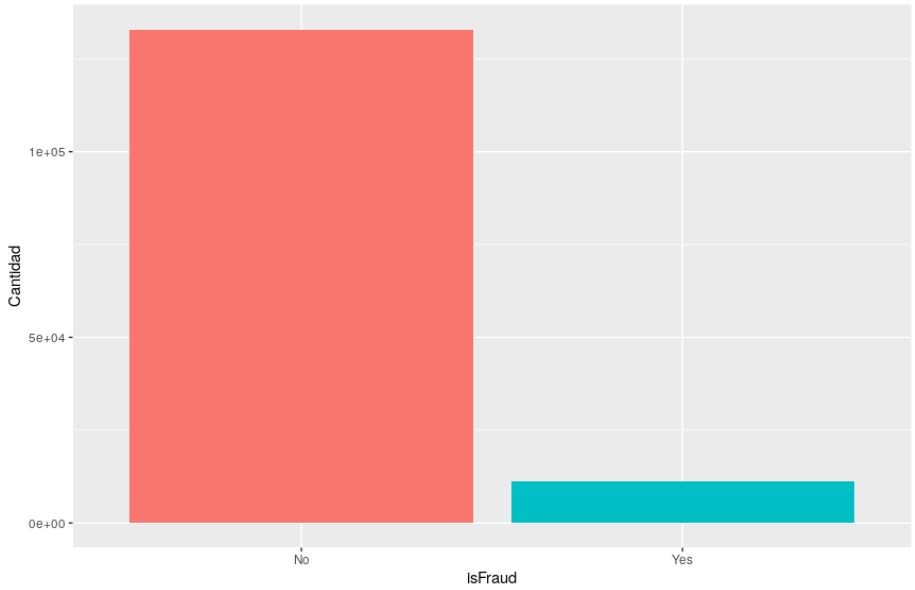

Figura 5: Cantidad de ejemplos de cada valor de la clase objetivo presentes en el conjunto de datos.

3 Pre-procesamiento

Para empezar con el pre-procesamiento, vamos a transformar a factores todas las variables que se especifican como categóricas en la competición de kaggle. Además, pasaremos las variables *TransactionAmt* (cantidad de dinero) y *TransactionDT* (delta de tiempo) a intervalos y eliminaremos el ID de la transacción. La clase objetivo *isFraud* se cambia a valores con un significado más claro (de 1 y 0 a *Yes* y *No*).

```
data_discretized <-
    train_dataset %>%
    mutate(isFraud = as.factor(ifelse(isFraud == 1, "Yes", "No"))) %>%
    mutate_at(c("ProductCD", "P_emaildomain", "R_emaildomain",
        "DeviceType", "DeviceInfo"), factor) %>%
    mutate_at(vars(starts_with("addr")), factor) %>%
    mutate_at(vars(starts_with("card")), factor) %>%
    mutate_at(vars(starts_with("M")), factor) %>%
    mutate_at(vars(matches("id_(1[2-9]|2[0-9]|3[0-8])")), factor) %>%
    mutate(
      TransactionAmt = cut_number(TransactionAmt, n = 10),
      TransactionDT = cut_number(TransactionDT, n = 10)
      ) %>%
    select(-TransactionID)
```

A partir de la información aportada por `df_status()`, podemos eliminar directamente las variables con gran porcentaje de valores NA y dispersión muy alta o muy baja. En mi caso, elimino todas aquellas variables con más de un 50% de valores vacíos, más de un 80% de dispersión y las variables numéricas con menos de un 1% de dispersión.

```
library(funModeling)

status <- df_status(data_discretized)

na_cols <-
  status %>%
  filter(p_na > 50) %>%
  select(variable)

high_dif_cols <-
  status %>%
  filter(unique > 0.8 * nrow(data_discretized)) %>%
  select(variable)

low_dif_cols <-
  status %>%
  filter(type == "numeric" & unique < 0.01 * nrow(data_discretized)) %>%
  select(variable)

data_reduced <-
  data_discretized %>%
  select(-one_of(
    bind_rows(list(na_cols, high_dif_cols, low_dif_cols))$variable
    ))
```

Una vez eliminadas estas variables, todavía quedan en el conjunto de datos valores vacíos, los cuáles tenemos que reemplazar. Si se trata de variables numéricas, sustituiremos con el valor medio de la variable usando la función `na.aggregate` del paquete *zoo*, mientras que si son categóricas, usaremos una nueva categoría *Unknown*.

```
library(zoo)

data_replaced <-
  data_reduced %>%
  mutate_if(is.numeric, na.aggregate) %>%
  mutate_if(is.factor, fct_explicit_na, na_level = "Unknown")
```

Calculando la matriz de correlación y usando la función `findCorrelation()` del paquete *caret*, podemos obtener una lista de variables que se pueden eliminar del conjunto de datos debido a que poseen alta correlación con otras variables. En mi caso, he buscado las que tengan una correlación superior a 0,8.

```
library(caret)
library(reshape2)

corr_matrix <-
    data_replaced %>%
    select_if(is.numeric) %>%
    cor(.)

data_final <-
    data_replaced %>%
    select(-one_of(findCorrelation(corr_matrix, cutoff = 0.8, names =
        TRUE)))
```

Después de todo este tratamiento, nos quedan todavía 54 variables. Vamos a reducir aún más este número usando una técnica especializada de extracción de variables conocida como *Boruta*. Esta función seleccionará y rechazará variables en base a su importancia calculada. En mi caso, la función seleccionaba todas las variables, por lo que decidí quedarme con las 10 que más importancia media tenían.

```
library(Boruta)

boruta_output <- Boruta(isFraud ~ ., data = data_final, doTrace = 2)

imp_cols <-
  attStats(boruta_output) %>%
  rownames_to_column(var = "variable") %>%
  top_n(10, meanImp) %>%
  select(variable)

data_boruta <-
  data_final %>%
  select(isFraud, one_of(imp_cols$variable))
```

Ya tenemos nuestro conjunto de datos final con tan solo 10 variables, tamaño que podemos gestionar con facilidad. Sin embargo, seguimos teniendo el problema del desbalanceo de la clase objetivo que vimos en la exploración de los datos. Para solucionar esto, vamos a aplicar técnicas de generación sintética de datos (combinando *undersampling* y *oversampling*). En concreto, usaremos ROSE y SMOTE, obteniendo dos conjuntos de datos balanceados diferentes sobre los que entrenaremos los mismos clasificadores para comprobar el impacto de ambas técnicas en la calidad de las predicciones. En las figuras 7 y 8 se puede ver el nuevo reparto de registros tras aplicar ROSE y SMOTE, respectivamente.

```
library(ROSE)
library(DMwR)

data_balanced_rose <- ROSE(isFraud ~ ., data = data_boruta, seed =
```

```
    1)$data
data_balanced_smote <- SMOTE(isFraud ~ ., as.data.frame(data_boruta))
```

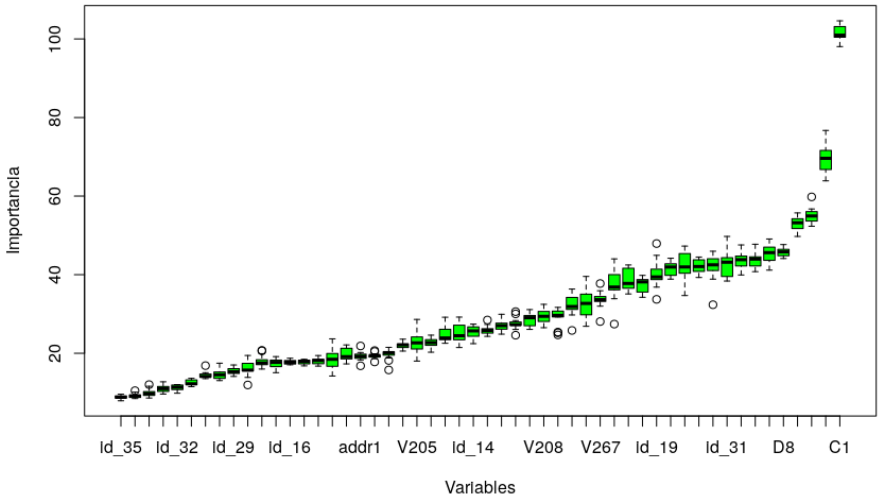

Figura 6: Importancia de cada variable calculada por *Boruta*.

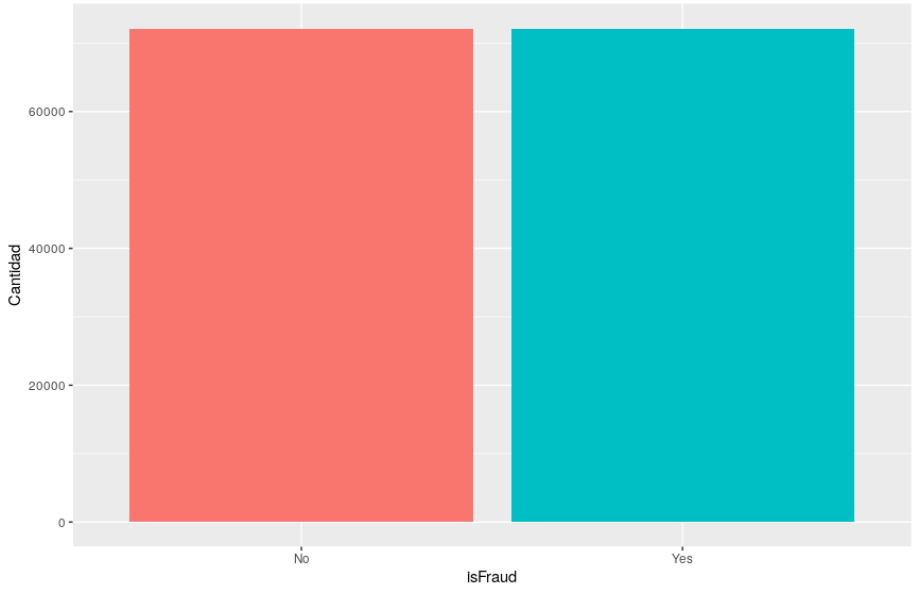

Figura 7: Cantidad de ejemplos de cada valor de la clase objetivo presentes en el conjunto de datos balanceado con ROSE.

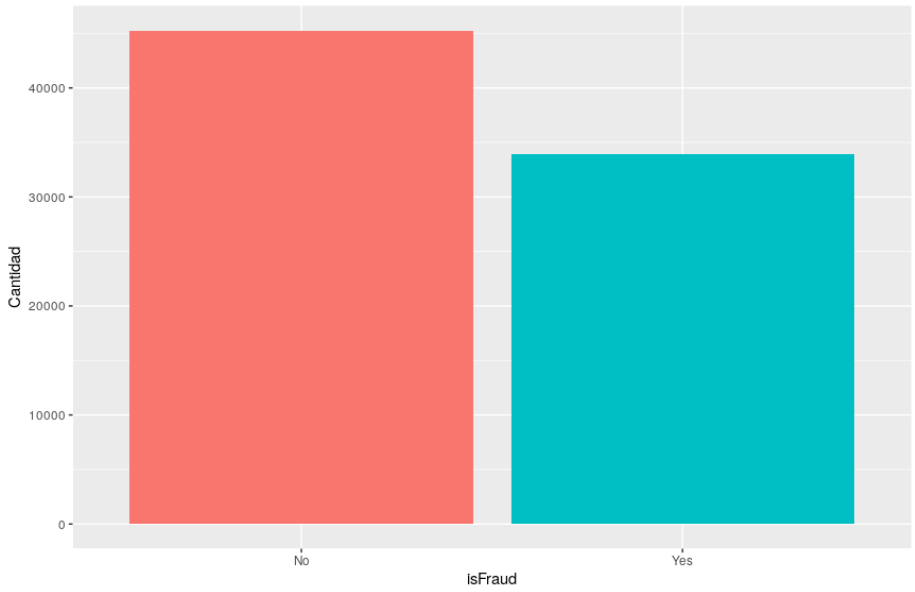

Figura 8: Cantidad de ejemplos de cada valor de la clase objetivo presentes en el conjunto de datos balanceado con SMOTE.

4 Clasificadores

Como clasificadores, he decidido entrenar modelos de regresión logística y perceptrón multicapa. Cabe destacar que he intentando entrenar otros modelos como *Random Forest* o SVM, pero era incapaz de ello ya que RStudio se encontraba siempre con algún error fatal que hacía abortar la sesión.

En las siguientes secciones, se va a explicar cómo se ha llevado a cabo el entrenamiento de los modelos usando el paquete *caret* y, también, se compararán los resultados de cada uno para cada conjunto de datos de los dos que hemos preparado.

Antes de entrenar cualquier modelo, los datos se dividen en dos subconjuntos para entrenamiento y validación usando la función `createDataPartition()` de *caret*. En mi caso, un 70% se destina a entrenamiento y, el resto, a validación.

```
library(caret)

set.seed(0)

# ROSE
train_index_rose <- createDataPartition(data_balanced_rose$isFraud, p =
    .7, list = FALSE)
train_rose <- data_balanced_rose[train_index_rose, ]
val_rose <- data_balanced_rose[-train_index_rose, ]

# SMOTE
train_index_smote <- createDataPartition(data_balanced_smote$isFraud, p
    = .7, list = FALSE)
train_smote <- data_balanced_smote[train_index_smote, ]
val_smote <- data_balanced_smote[-train_index_smote, ]
```

4.1 Regresión logística

A pesar de su nombre, la regresión logística es una técnica muy utilizada en problemas de clasificación binaria, como es el caso del nuestro. Esta técnica se apoya en la función sigmoide, la cuál nos proporciona un valor entre 0 y 1, que interpretamos como la probabilidad de que se pertenezca a la clase principal.

He decidido usar este clasificador debido a que es muy simple y no requiere de un gran consumo de recursos, lo cuál favorece su entrenamiento en mi ordenador personal, donde se ha realizado la práctica.

```
# ROSE
lr_grid_rose <- expand.grid(.nIter = 5)
lr_control_rose <- trainControl(method = "repeatedcv", number = 10,
    repeats = 5)
lr_model_rose <- train(
  isFraud ~ .,
  data = train_rose,
```

```
  method = "LogitBoost",
  trControl = lr_control_rose,
  tuneGrid = lr_grid_rose
  )

lr_prediction_rose <- predict(lr_model_rose, val_rose, type = "raw")

# SMOTE
lr_grid_smote <- expand.grid(.nIter = 5)
lr_control_smote <- trainControl(method = "repeatedcv", number = 10,
    repeats = 5)
lr_model_smote <- train(
  isFraud ~ .,
  data = train_smote,
  method = "LogitBoost",
  trControl = lr_control_smote,
  tuneGrid = lr_grid_smote
)

lr_prediction_smote <- predict(lr_model_smote, val_smote, type = "raw")
```

Como se puede ver en el código, los clasificadores de regresión logística se han entrenado usando 5 iteraciones. Además, se ha aplicado validación cruzada con 10 árboles y otras 5 repeticiones. Las medidas de calidad obtenidas para las predicciones realizadas sobre los conjuntos de validación se pueden ver en la figura 9.

En el caso concreto de nuestro problema, el *recall* o exhaustividad es una medida muy importante, ya que un falso negativo (una transacción fraudulenta no identificada como tal) es muy costoso. Como se puede ver, esta medida no es especialmente buena para la predicción realizada por estos modelos.

Apreciamos que las medidas son prácticamente idénticas para ambos conjuntos de datos, siendo las del clasificador aplicado al conjunto de SMOTE ligeramente mejores. Podemos asumir entonces que la aplicación de las técnicas no ha tenido un impacto relevante en los resultados.

4.2 Perceptrón multicapa

El segundo clasificador utilizado es el perceptrón multicapa. Estas redes neuronales son ampliamente utilizadas en problemas de clasificación debido a su capacidad para extraer características ocultas de los patrones del conjunto de entrenamiento.

```
# ROSE
mlp_control_rose <- trainControl(method = "repeatedcv", number = 10,
    repeats = 5)
mlp_model_rose <- train(
  isFraud ~ .,
  data = train_rose,
```

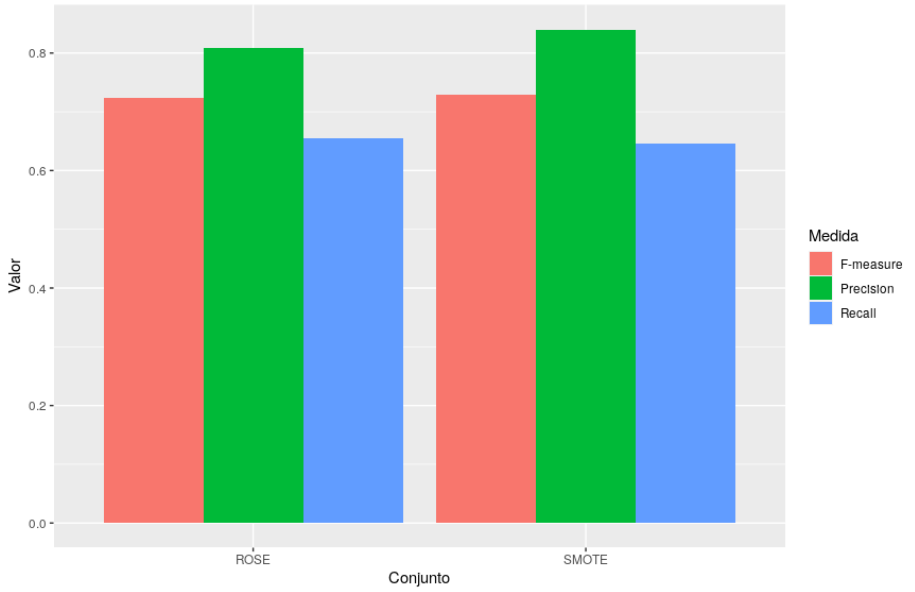

Figura 9: Medidas de calidad de la predicción realizada mediante regresión logística.

```
  method = "mlp",
  trControl = mlp_control_rose
)

mlp_prediction_rose <- predict(mlp_model_rose, val_rose, type = "raw")

# SMOTE
mlp_control_smote <- trainControl(method = "repeatedcv", number = 10,
    repeats = 5)
mlp_model_smote <- train(
  isFraud ~ .,
  data = train_smote,
  method = "mlp",
  trControl = mlp_control_smote
)

mlp_prediction_smote <- predict(mlp_model_smote, val_smote, type = "raw")
```

Es cierto que consumen más recursos que el otro clasificador usado, pero también se podría esperar que se consiguieran resultados más precisos. En la figura 10 se pueden ver estos resultados, que son incluso peores que los obtenidos por el clasificador basado en regresión logística. Sí que podemos destacar los resultados obtenidos para el conjunto de datos balanceado con SMOTE. En este

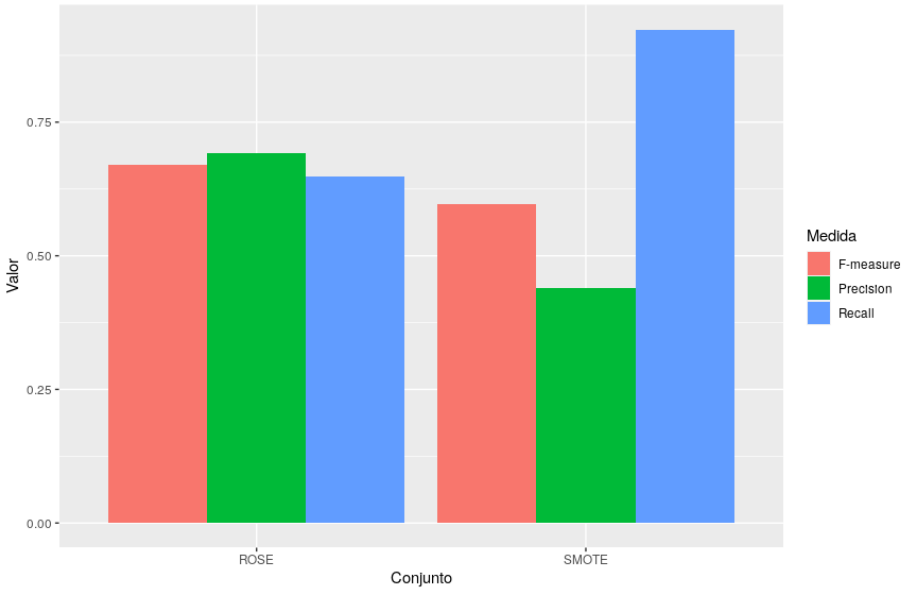

Figura 10: Medidas de calidad de la predicción realizada por el perceptrón multicapa.

caso, vemos como la precisión es muy baja y la exhaustividad es muy elevada. El clasificador es capaz de identificar muy bien los verdaderos positivos del conjunto, pero también cuenta con un gran número de falsos positivos. En resumen, sacamos de aquí que este clasificador tiende a clasificar como fraude la mayoría de las transacciones.

4.3 Comparación

Para comparar los clasificadores entre ellos, vamos a separar los resultados obtenidos para el conjunto de datos al que se ha aplicado ROSE y al que se ha aplicado SMOTE.

En la figura 11, tenemos los resultados de ambos clasificadores sobre el conjunto de validación de ROSE. Apreciamos claramente como la precisión ha sido menor con el perceptrón multicapa, mientras que la exhaustividad se ha mantenido practicamente igual. En definitiva, los resultados dados por el perceptrón han sido peores que los obtenidos con la regresión logística.

En cuanto al conjunto de SMOTE, podemos ver los resultados de ambos clasificadores en la figura 12. Aquí es donde tenemos los resultados más dispares. Ya hemos explicado en el apartado anterior que los resultados del perceptrón nos hacen ver que clasifica la mayoría de transacciones como fraudulentas, lo que hace que no sea un clasificador fiable. La regresión logística vuelve a ser mejor que el perceptrón, aunque sus resultados tampoco son demasiado buenos.

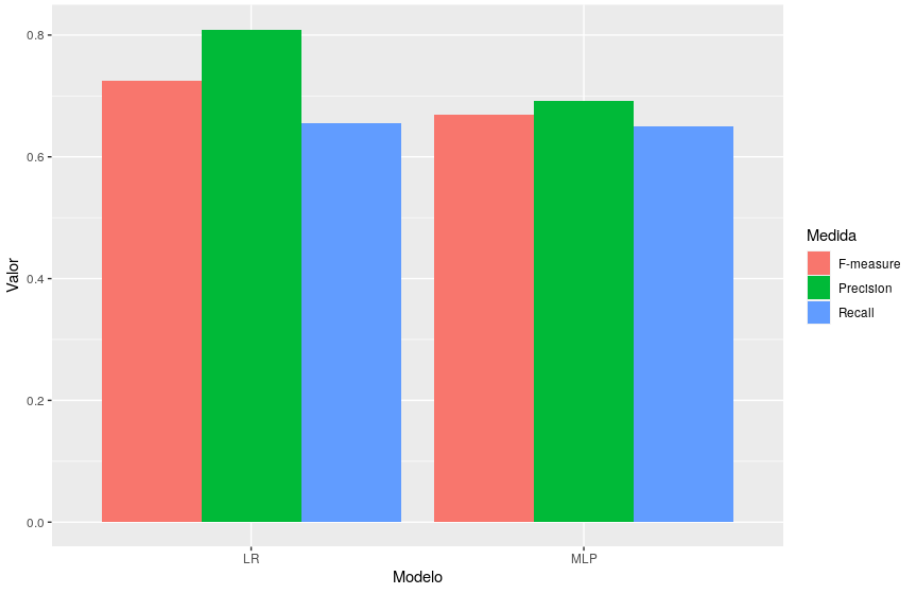

Figura 11: Medidas de calidad de las predicciones realizadas sobre el conjunto de datos ROSE.

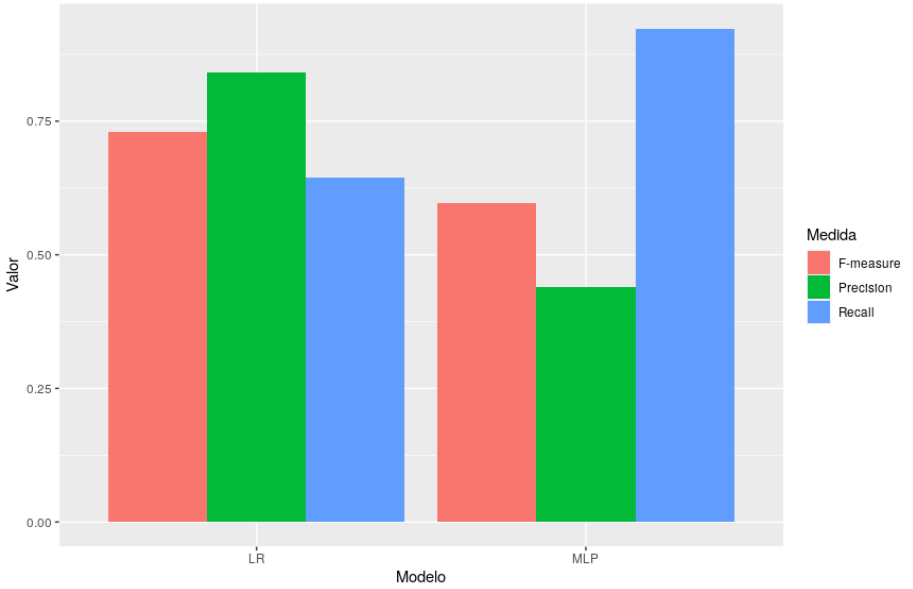

Figura 12: Medidas de calidad de las predicciones realizadas sobre el conjunto de datos SMOTE.

5 Conclusiones

Viendo los resultados obtenidos en la sección anterior y, basándonos en el *F-measure* como una medida general de la bondad del clasificador, podemos decir que la regresión logística es una mejor opción que el perceptrón multicapa. Además, los resultados obtenidos para el conjunto balanceado usando ROSE han sido mejores que los obtenidos para el de SMOTE.

El clasificador basado en regresión logística, a pesar de resultar ser el mejor de los experimentos que he realizado, tampoco me ha dado resultados que pueda considerar particularmente buenos, ya que la medida del *recall* es demasiado baja para un problema donde es tan importante como es este. Otros clasificadores, como *Random Forest*, seguro que son capaces de obtener resultados mejores, pero, como ya he comentado, me han sido imposibles aplicar debido, supongo, a los limitados recursos de mi ordenador.

Sin duda, la parte más importante de este problema de clasificación se encuentra en la selección de las variables con las que entrenar los modelos, aunque también hemos visto que las técnicas usadas para el balanceo de los ejemplos de la clase objetivo tiene un impacto considerable. Es posible que se puedan obtener mejores resultados aplicando otra técnica de extracción de variables diferente a *Boruta* o, incluso, usando más variables de las que he usado yo, una vez más, por mis limitados recursos.