ugr | Universidad de Granada

Máster en Ingeniería Informática

SISTEMAS INTELIGENTES PARA LA GESTIÓN EN LA EMPRESA

Práctica 1. Pre-procesamiento de datos y clasificación binaria

---

Lidia Sánchez Mérida

Granada, Abril de 2020

Contents

Conjuntos de datos

En esta práctica se pretende analizar el conjunto de datos procedente de *Kaggle* denominado **IEE-CIS Fraud Detection** [1]. En él se distinguen dos tipos de documentos. Aquellos que contienen el término `identity` contienen los datos personales de los individuos que han realizado transacciones bancarias. Mientras que los ficheros que contienen la palabra `transaction` disponen de la información asociada a cada una de las transacciones realizadas. Ambos ficheros son relacionables a través de un campo denominado `TransactionID`, el cual nos permite conocer más detalles acerca de las personas que han realizado las transferencias. Sin embargo, de muchas de ellas no se conoce esta información.

Para cada uno de estos tipos de ficheros existe su correspondiente conjunto de entrenamiento y validación ya separados, pero como los conjuntos de datos son tan amplios voy a usar directamente los ficheros `train_innerjoin.csv` y `test_innerjoin.csv` proporcionados en el repositorio de GitHub de la asignatura.

```
# Establecemos una semilla para que los resultados sean reproducibles.
set.seed(32)
# Leemos los datos desde los ficheros
train<-read.csv(file="./ficheros/train_innerjoin.csv", header=TRUE, sep=",")
test<-read.csv(file="./ficheros/test_innerjoin.csv", header=TRUE, sep=",")
# Dimensiones de los conjuntos
dim(train)
```

```
## [1] 144233    434
```

```
dim(test)
```

```
## [1] 141907    433
```

Tras cargar los datos podemos observar, como anteriormente he puntualizado, que ambos cuentan con un número considerablemente amplio tanto de registros como de variables. Para conocer más información acerca de los conjuntos de datos vamos a realizar un análisis exploratorio que nos permita identificar los aspectos más relevantes.

Análisis exploratorio

Estado de los datos

Primeramente vamos a conocer el estado de los datos. Mediante la función `df_status` [2] podremos conocer los valores de todos los campos de un conjunto con el objetivo de conocer la cantidad y el porcentaje de ceros, valores perdidos o infinitos. Asimismo, en la última columna también nos indica la cantidad de valores únicos que existen para cada campo. De este modo podemos saber, por ejemplo, si una variable es categórica. Asimismo, Kaggle dispone de una sección en la que se describen brévemente los campos [3].

```
# Obtenemos el estado de los dos conjuntos de transacciones
library(funModeling)
train_st<-df_status(train)
test_st<-df_status(test)
```

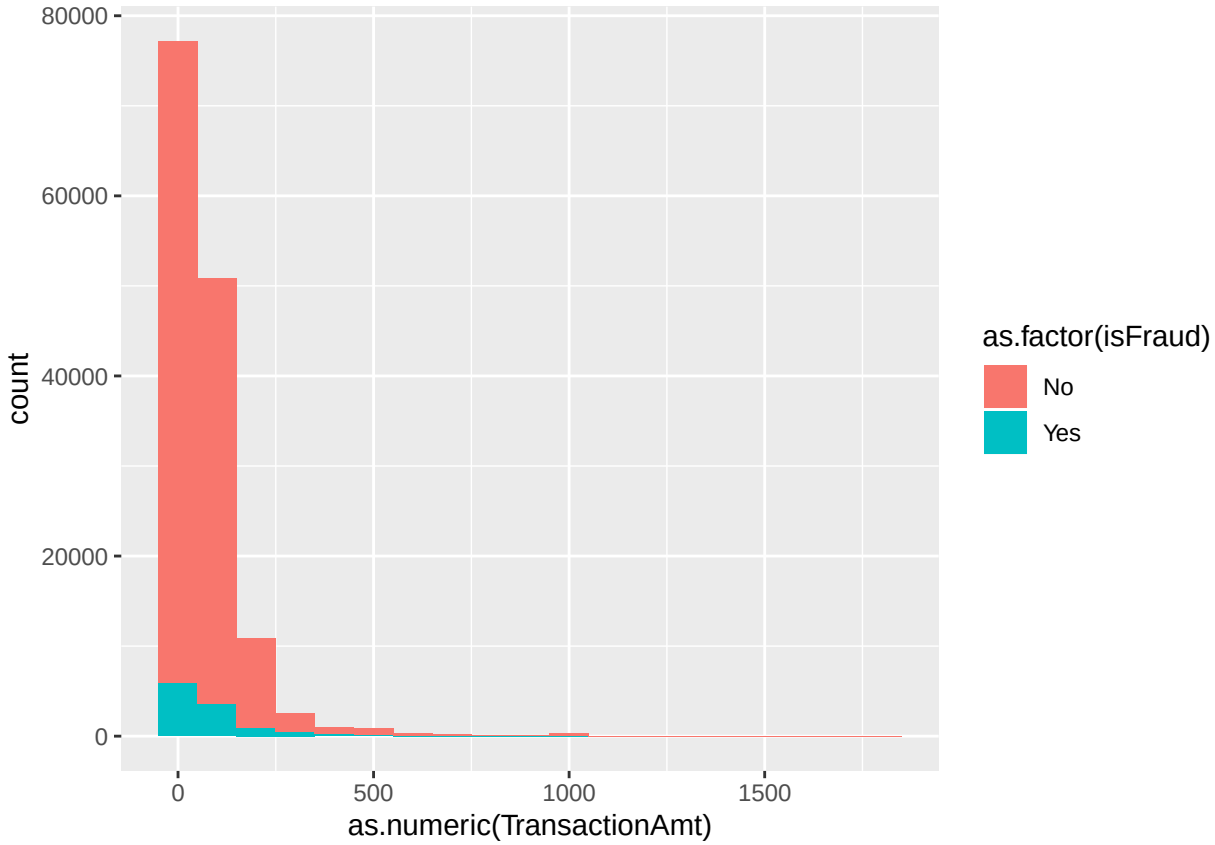
Si bien el resultado proporcionado por estas funciones es súmamente extenso como para mostrarlo, he realizado un análisis acerca de las variables más relevantes obteniendo las siguientes conclusiones:

- En primer lugar destacamos que la variable categórica `isFraud` que intentamos predecir dispone de un altísimo número de ejemplos de transacciones no fraudulentas en el conjunto de entrenamiento. Tal es así que apenas existe un 8% de transacciones clasificadas como fraudulentas. Si bien este tipo de fenómenos es bastante común, clases tan súmamente desbalanceadas suelen dificultar el proceso de entrenamiento y obtención de buenos clasificadores.
- Casi la mitad de las variables disponen de aproximádamente un 50% de valores perdidos. Mayoritariamente se encuentran relacionadas con medidas tales como la distancia, características como la dirección, entre otras.
- Existen bastantes variables categóricas, además de `isFraud` que indican los dominios origen y destino, el tipo de dispositivo desde el que se hizo la transacción, el producto de la misma, entre otras.

Relación dinero-fraude

A continuación procedo a representar gráficamente otro tipo de estadísticas que pueden también ser interesantes para conocer algo más las características de las transacciones. En primer lugar vamos a averiguar si existe algún tipo de relación entre la **cantidad de la transacción y su clasificación como fraudulenta o no**. El objetivo es conocer si existe un patrón para detectar las transacciones fraudulentas, como por ejemplo si en muchas de ellas se han traspasado grandes cantidades de dinero. Para ello vamos a dibujar un histograma que represente la variable `TransactionAmt`, que es la que contiene las sumas de dinero, y las diferenciaremos en función de si son o no fraudulentas. Para este último paso deberemos de transformar los valores de la columna `isFraud` a factores.

```
library(ggplot2)
library(magrittr)
library(dplyr)
# Transformamos 0->No, 1->Yes para diferenciar las transacciones
# fraudulentas de las que no lo son.
train_categ<-train %>% mutate(isFraud = as.factor(ifelse(isFraud == 1, 'Yes', 'No')))
# Representamos las cantidades y si son de transacciones fraudulentas o no.
ggplot(train_categ) +
  geom_histogram(aes(x = as.numeric(TransactionAmt),
     fill = as.factor(isFraud)), binwidth = 100)
```

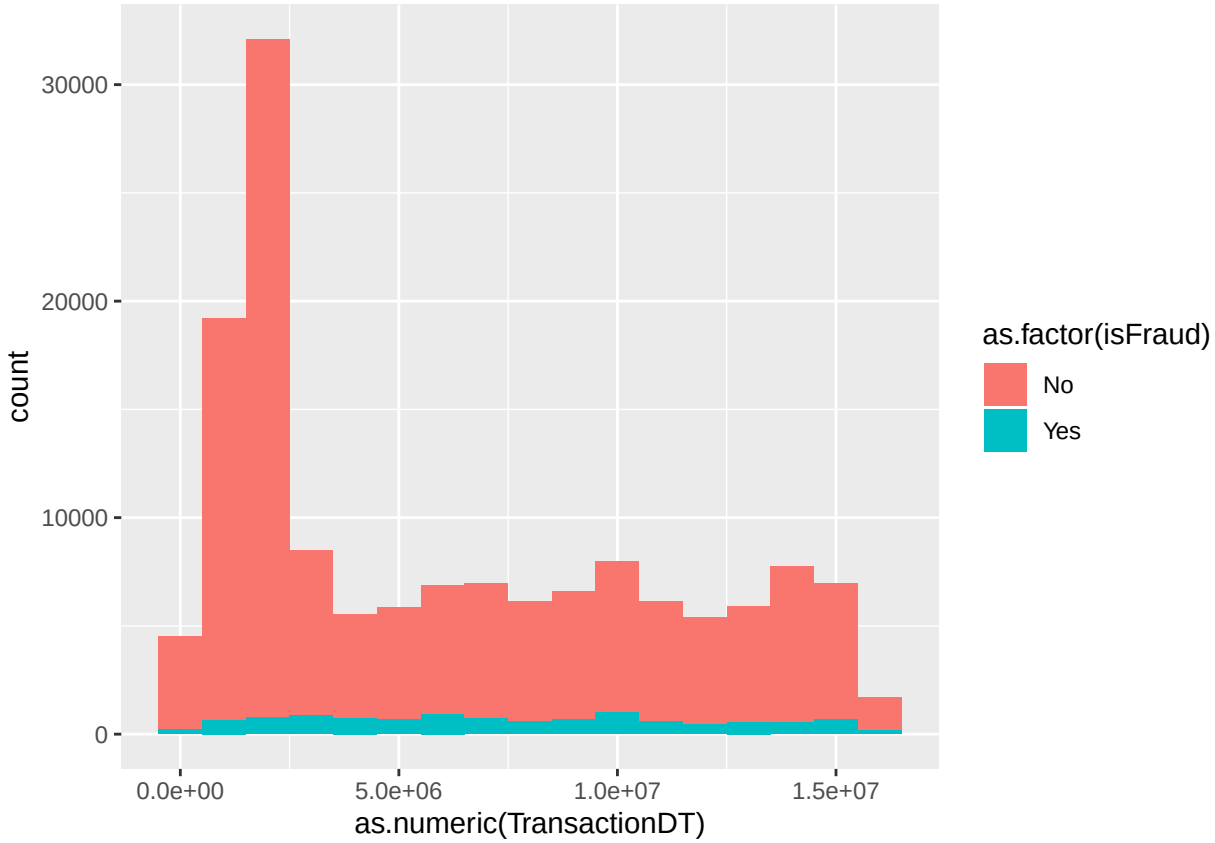


Como se puede observar no existen grandes diferencias entre las cantidades monetarias de las transacciones clasificadas como fraudulentas de las que no lo son. Sin embargo, este histograma nos permite comprobar que la **mayoría de transacciones se encuentran agrupadas** dentro de un rango aproximado [0-700], lo cual nos posibilita realizar un **corte horizontal para reducir el número de registros** a los contenidos en dicho rango.

Relación tiempo-fraude

A continuación vamos a estudiar como segundo caso la relación entre el **tipo de transacción y la medida de tiempo asociada** en el campo `TransactionDT`. Esta representa la diferencia entre dos valores temporales, y aunque si bien en la descripción del *dataset* no especifican las medidas utilizadas, sus valores se basan en los de los campos `DX` donde X es un número entre 1 y 15. De estos conocemos por la descripción en Kaggle que se consideran, por ejemplos, los días transcurridos entre una transacción y otra. El objetivo de este análisis consiste en averiguar si el rango temporal entre dos transacciones puede ser un factor a considerar para averiguar si son o no fraudulentas.

```
ggplot(train_categ) +
  geom_histogram(aes(x = as.numeric(TransactionDT),
    fill = as.factor(isFraud)), binwidth = 1000000)
```



Tal y como se puede observar, aquellas transacciones marcadas como **fraudulentas disponen de un valor menor que las no fraudulentas**, por lo que podemos considerar que quizás este factor de tiempo puede estar relacionado con la naturaleza legal de la transacción. Una posible teoría que explique este suceso puede basarse en que cuando, por ejemplo hacemos algo que está mal, intentamos que sea lo más rápido posible. Al entrenar los modelos comprobaremos si efectívamente existe algún tipo de relación entre esta variable y la columna a predecir.

Preprocesamiento de datos

En esta sección procedemos a aplicar diversas técnicas para mejorar la calidad del conjunto de datos de modo que podamos obtener un subconjunto más sencillo con el que entrenar los modelos predictivos. Uno de los principales problemas de este conjunto reside en su **gran dimensionalidad** tanto del número de registros como de columnas. El hecho de disponer de un enorme grupo de muestras puede provocar problemas de rendimiento a la hora de intentar entrenar un modelo con tal volumen de datos. Mientras que disponer de tal cantidad de variables también puede producir grandes tiempos de espera en entrenamiento al considerar tantas columnas para obtener el clasificador. Estos son los dos principales problemas que se abordan a continuación.

Reducción del número de muestras aleatoriamente

En primer lugar debo reducir el número de registros del conjunto de datos para poder trabajar con él. Para ello, en primer lugar, vamos a realizar el corte horizontal referente al **análisis exploratorio de la cantidades monetarias** de las transacciones. De este modo obtendremos un conjunto de datos más reducido

y orientado hacia el grupo mayoritario de transacciones. Para ello vamos a eliminar aquellos registros con un cantidad superior a 700 USD.

```
library(dplyr)
library(magrittr)
minitrain_700<-train %>% filter(TransactionAmt < 700)
cat("Dimensiones al quitar transacciones > 700 USD\n")
```

```
## Dimensiones al quitar transacciones > 700 USD
```

```
dim(minitrain_700)
```

```
## [1] 143560    434
```

Tal y como podemos comprobar, hemos conseguido reducir casi 1.000 registros. Al sufrir tan poca variación realmente **no creo que merezca la pena eliminar 1.000 muestras** del conjunto puesto que este hecho no supone un cambio significativo en el entrenamiento de los modelos. Asimismo, dependiendo de las combinaciones de preprocesamientos que se relicen al conjunto, se evaluará la posibilidad de reducir el número de registros en caso de que las técnicas utilizadas inviertan demasiado tiempo.

Variables con demasiados valores perdidos

Continuamos abordando otro de los principales problemas de este conjunto de datos: los valores perdidos o NAs. Si bien uno de los métodos posibles para eliminarlos es predecirlos utilizando alguna técnica que nos permita obtener un valor aproximado a partir de la información que existe en el conjunto, si el número de NAs es demasiado predominante ($> 50\%$) es prácticamente imposible aplicar este método. Por lo tanto, a continuación se desarrolla una función en la que se eliminarán aquellas columnas del dataset que dispongan de un número de valores perdidos **superior al 50% de los datos** totales. Para ello me he basado en los ejemplos de los *scripts* proporcionados en la asignatura.

```
eliminar_columnas_nas<-function(datos) {
  library(magrittr)
  library(dplyr)
  library(funModeling)
  estado<-df_status(datos)
  # Obtenemos aquellas columnas con más de un 50%
  # de valores perdidos (NAs) a partir del estado
  na_cols <- estado %>%
    filter(p_na > 50) %>%
    select(variable)
  # Eliminamos las columnas obtenidas
  remove_cols <- bind_rows(
    list(na_cols)
  )
  # Obtenemos el conjunto de datos reducido
  datos_finales <- datos %>%
    select(-one_of(remove_cols$variable))
  # Devolvemos el conjunto resultante
  datos_finales
}
```

Desviación estándar de las variables

Se consideran variables irrelevantes aquellas que no aportan información útil debido a que sus **valores son muy similares**. Este tipo de columnas no nos ayudan a predecir si una transacción es o no fraudulenta puesto que independientemente de las muestras su comportamiento no varía. Para ello vamos a calcular la **desviación estándar**. Si esta medida estadística es 0 entonces nos indica que no existe variabilidad en los valores de la columna, y por tanto, esta variable no será útil.

Para realizar este preprocesamiento vamos a hacer uso de la función `nearZeroVar` [4] que calcula el número de valores únicos y su frecuencia de aparición. De este modo obtendremos aquellas variables cuya desviación estándar sea igual o cercana a 0. Previo a su aplicación, es necesario transformar todas las variables a numéricas para poder calcular la desviación estándar. Para las columnas categóricas se asignará a cada etiqueta un único valor que la represente.

```
eliminar_columnas_sd<-function(datos) {
  # Convertimos todas las variables a variables numéricas
  # excepto isFraud porque cambia 0->1 y 1->2
  for(i in c(1:ncol(datos))) {
    if (names(datos[i]) != "isFraud") {
      datos[,i] <- as.numeric(datos[,i])
    }
  }
  # Calculamos el número de variables con desviación estándar cercana o igual a 0.
  library(caret)
  nz<-nearZeroVar(datos, names = TRUE, saveMetrics = TRUE)
  # Desviación estándar = 0
  nz0<-nz[nz,"zeroVar"] > 0, ]
  # Eliminamos las variables con desviación estándar = 0
  datos_finales<-datos[, -which(colnames(datos) %in% rownames(nz0))]
  # Devolvemos el conjunto resultante
  datos_finales
}
```

Tratamiento de valores perdidos

En este apartado procedemos a tratar los numerosos valores perdidos que contiene el conjunto de datos. Si bien existen multitud de técnicas para aplicar, yo voy a considerar las siguientes:

- **Eliminación de NAs**. Esta puede ser la técnica más drástica para tratar los valores perdidos. Para llevarla a cabo vamos a aplicar la función `na.omit()` que eliminará todos los registros que contentan este tipo de valores.
- **Imputación de NAs**. Al contrario que la anterior, esta técnica más sofisticada es una de las más utilizadas según he estado investigando. Para aplicarla existen varias librerías pero al parecer la más popular es `mice` debido a las múltiples operaciones que puede llevar a cabo para asignar un valor a cada NA [5]. Dependiendo de la naturaleza de los datos, existen métodos específicos aunque también dispone de métodos generales aplicables a cualquier tipo de dato.

```
eliminar_nas<-function(datos) {
  datos_finales<-na.omit(datos)
  # Devolvemos el conjunto resultante
  datos_finales
}
```

```r
imputar_nas<-function(datos, imputs, iters, metodo) {
  library(mice)
  # Especificamos el número de imputaciones, el número de iteraciones
  # por imputación y el método a utilizar para imputar los NAs
  modelo_mice<-mice(datos, m=imputs, maxit=iters, method=metodo)
  # Obtenemos el conjunto de datos imputado
  datos_imputados<-complete(modelo_mice)
  # Devolvemos el conjunto resultante
  datos_imputados
}
```

Correlación

En esta sección vamos a estudiar la **correlación entre las variables y la columna a predecir**, que en el caso de este *dataset* es `isFraud`, y posteriormente analizaremos cuán **relacionadas se encuentran las variables entre sí**. El objetivo es, por un lado, conocer si existe alguna variable o grupo de variables que expliquen el comportamiento de la variable categórica a predecir. Mientras que por otro lado queremos conocer si existen variables muy correladas entre sí, de manera que podamos reducir la dimensionalidad eliminando aquellas cuyo coeficiente de correlación sea muy alto, ya que este sería un indicio de que dichas columnas aportan la misma información.

Para el primer estudio vamos a aplicar la función `correlation_table` tal y como se ejemplifica en uno de los *scripts* proporcionados en la asignatura. Como precondición para calcular los coeficientes de correlación, es necesario que las columnas sean numéricas. De igual modo ocurre en la correlación entre variables para la que utilizaremos, en primer lugar, la función `cor` que se encargará de calcular los coeficientes entre todas las variables del conjunto. Posteriormente, aplicaremos la función `findCorrelation` [6] que nos ayudará a identificar aquellas variables que tienen un determinado coeficiente de correlación. De este modo se puede personalizar el nivel de exigencia con el que eliminar las columnas correladas.

```r
correlacion_fraude<-function(datos) {
  library(funModeling)
  # Calculamos los coeficientes de correlación en torno a la variable clasificatoria
  tabla_corr<-correlation_table(datos, target='isFraud')
  tabla_corr
}

correlacion_variables<-function(datos, max_corr) {
  library(caret)
  # Calculamos los coeficientes de correlación todas-todas las variables.
  coefs<-cor(datos)
  # Obtenemos aquellas variables con una correlación superior
  cols_corr<-findCorrelation(na.omit(coefs), cutoff=max_corr)
  # Eliminamos las columnas resultantes del dataset
  datos_finales<-datos[-c(cols_corr)]
  # Devolvemos el conjunto resultante
  datos_finales
}
```

Outliers

Los *outliers* son datos cuyos valores están caracterizados por encontrarse fuera del rango normal para el atributo. En otras palabras, este tipo de datos disponen de valores muy diferentes a los del resto del

conjunto y por ende pueden afectar a la calidad de los modelos entrenados. Por este motivo, se va a realizar un estudio y tratamiento de este tipo de valores. Para el análisis me he inspirado en este ejemplo [7] en el que se hace uso de la función `boxplot` [8], con la que se calculan los *outliers* de cada columna.

```
estudio_outliers<-function(datos) {
  library(magrittr)
  library(dplyr)
  library(tidyr)
  library(purrr)
  # Calculamos los outliers de todas las variables del conjunto de datos.
  outliers<-datos %>%
        map(~ boxplot.stats(.x)$out)
  # Devolvemos los resultados
  outliers
}
```

Una vez hemos encontrado los *outliers*, procedemos a estudiar los diferentes tratamientos que se les pueden aplicar. Dentro del ejemplo mecionado anteriormente se presentan varias técnicas, de las cuales he escogido las dos más interesantes a mi parecer: **la imputación mediante la fórmula de los cuantiles y la predicción de los valores**. La primera técnica consiste en reemplazar aquellos valores que se sitúen fuera del rango de los cuantiles 25 y 75, por el cuantil 5 o el 95, respectivamente. El objetivo es transformar los *outliers* que se encuentran por debajo del rango en el valor más pequeño que se encuentra dentro de dicho intervalo, y realizar el mismo procedimiento con los que se encuentran por encima del rango pero, en este caso, asignando el mayor valor posible.

La segunda técnica consiste en convertir todos los *outliers* detectados en valores NA para posteriormente imputarlos. Como ya disponemos de una función que se encarga de tratar los valores perdidos, será esa la que usaremos para predecirlos.

```
outliers_formula<-function(datos) {
  # Eliminamos la columna isFraud para que no se le aplique este procedimiento.
  datos_finales<-datos[, -which(names(datos) %in% c("isFraud"))]
  i <- 1
  for(col in datos_finales) {
    # Calculamos los cuantiles 25 y 75 para comprobar es un outlier
    quantiles<-quantile(col, probs=c(0.25, 0.75))
    # Calculamos los cuantiles 5 y 95 para asociar como nuevos valores a los outliers
    nuevos_valores<-quantile(col, probs=c(0.05, 0.95))
    # Calculamos la varianza máxima que puede sufrir un valor
    H<-1.5*IQR(col)
    # Sustitución de outliers
    col[col < (quantiles[1] - H)] <- nuevos_valores[1]
    col[col > (quantiles[2] + H)] <- nuevos_valores[2]
    # Actualizamos el dataset
    datos_finales[i]<-col
    i <- i + 1
  }
  # Volvemos a añadir la columna `isFraud`
  datos_finales<-cbind(datos_finales, isFraud=datos$isFraud)
  # Devolvemos los datos resultantes
  datos_finales
}

outliers_prediccion<-function(datos, imputs, iters, metodo) {
```

```
  library(magrittr)
  library(dplyr)
  library(tidyr)
  library(purrr)
  # Eliminamos la columna isFraud para que no se le aplique este procedimiento.
  datos_sin_fraud<-datos[, -which(names(datos) %in% c("isFraud"))]
  # Transformamos los valores outliers en valores perdidos (NAs)
  datos_nas<-datos_sin_fraud %>%
     map_if(is.numeric, ~ replace(.x, .x %in% boxplot.stats(.x)$out, NA)) %>%
     bind_cols
  # Imputamos los valores perdidos
  datos_finales<-imputar_nas(datos_nas, imputs, iters, metodo)
  # Añadimos de nuevo la columna isFraud
  datos_finales<-cbind(datos_finales, isFraud=datos$isFraud)
  # Devolvemos el dataset resultante
  datos_finales
}
```

Selección de variables

El objetivo de este apartado consiste en reducir el número de variables para dejar aquellas que mejor pueden predecir la variable `isFraud`. Si bien existen diversas técnicas para realizar este proceso, en mi caso se exponen dos de las más comunes: **Análisis de Componentes Principales (PCA)** y **análisis de la importancia de las variables**. En el primer caso se trata de un algoritmo de aprendizaje no supervisado que realiza combinaciones lineales de las columnas. Esta técnica solo es recomendable aplicarla cuando se dispone de muchas variables puesto que perdemos las columnas originales. Además, es muy sensible a la variación en los datos y por ello se recomienda realizar un **centrado y escalado** de los datos previo al PCA. Para llevar a cabo el procesamiento descrito voy a utilizar la función `preProcess` [9] para luego obtener el *dataset* resultante con la función `predict`.

Para el cálculo de la importancia de las características existen también diversos métodos pero tras probar varios, he optado por la función `boruta` [10]. Esta técnica se basa en *Random Forest* para calcular la relevancia de cada columna y establecer un ranking. Se puede parametrizar tanto el nivel de confianza con el que elige las variables como el número de ejecuciones que realiza.

```
pca<-function(datos, umbral) {
  # Eliminamos la columna isFraud para que no se le aplique este procedimiento.
  datos_sin_fraud<-datos[, -which(names(datos) %in% c("isFraud"))]
  library(caret)
  # Centrado, escalado y PCA
  pca<-preProcess(datos_sin_fraud, method = c("center", "scale", "pca"), thres=umbral)
  # Obtenemos el dataset con las combinaciones de las variables obtenidas del PCA
  datos_pca<-predict(pca, datos_sin_fraud)
  # Volvemos a añadir la columna `isFraud`
  datos_pca<-cbind(datos_pca, isFraud=datos$isFraud)
  # Devolvemos el conjunto de datos resultante
  datos_pca
}

importancia_variables<-function(datos, ejecs) {
  library(Boruta)
  boruta_output<-Boruta(isFraud ~ ., data=datos, doTrace=0, maxRuns=ejecs)
  # Estudia más aún las variables que no están claras
```

```
  roughFixMod<-TentativeRoughFix(boruta_output)
  # Seleccionamos los atributos calculados
  boruta_signif<-getSelectedAttributes(roughFixMod)
  # Obtenemos la importancia de cada variable
  imps<-attStats(roughFixMod)
  # Devolvemos las variables ordenadas de mayor a menor importancia
  imps2<-imps[imps$decision != 'Rejected', c('meanImp', 'decision')]
  resultado<-imps2[order(-imps2$meanImp), ]
  resultado
}
```

Balanceo de la clase a predecir

En esta sección se trata de balancear las clases de la variable a predecir. Como hemos podido observar en el análisis exploratorio, casi la totalidad de las muestras son transacciones no fraudulentas por lo que la clase minoritaria dispone de muy pocos ejemplos de transacciones fraudulentas. Si bien este tipo de problemas es muy común, en el caso de este *dataset* **es súmamente desproporcionado**. Para intentar paliar este inconveniente se van a aplicar dos de las técnicas más populares: **undersampling** y **oversampling**. En el primer caso se obtendrá un subconjunto de datos en el que se iguale el número de muestras de ambas clases de modo que exista el mismo número de transacciones fraudulentas y no fraudulentas. Mientras que con la segunda técnica el objetivo es producir más muestras de transacciones fraudulentas mientras se restan algunas de la clase mayoritaria para intentar balancear ambas.

Para aplicar *undersampling* se hará uso de la función `downSample` [11], la cual realizará el procedimiento descrito anteriormente. Con esta técnica se prevee una reducción muy drástica del conjunto de datos, que posteriormente comprobaremos cómo afecta a los modelos entrenados.

Por otro lado, para aplicar la técnica *oversampling* existen multitud de funciones pero yo he optado por la más popular denominada `SMOTE` [12]. Esta genera nuevas muestras de la clase minoritaria mientras que reduce el de la clase mayoritaria para intentar equilibrar ambas clases. Para ello se debe especificar el porcentaje de generación y eliminación, respectivamente.

```
under_sampling<-function(datos) {
  library(caret)
  # Obtenemos las etiquetas a predecir
  etiquetas<-factor(datos$isFraud)
  # Under-sampling para igualar ambas clases
  set.seed(32)
  datos_finales<-downSample(datos[, -which(names(datos) %in% c("isFraud"))],
      etiquetas, yname='isFraud')
  # Devolvemos los datos resultantes
  datos_finales
}

over_sampling<-function(datos, p_over, p_under, k_vecinos) {
  library(DMwR)
  # Oversampling con el algoritmo SMOTE para generar más muestras
  datos$isFraud<-as.factor(datos$isFraud)
  datos_finales<-SMOTE(isFraud ~., datos, perc.over=p_over, perc.under=p_under, k=k_vecinos)
  # Devolvemos los datos resultantes
  datos_finales
}
```

División del conjunto de datos

Por último vamos a introducir una función capaz de dividir el conjunto de datos en **entrenamiento y validación**. De este modo podremos calcular un mayor número de medidas de calidad tras validar los modelos, más que considerar solamente la tasa de error que nos devolvería Kaggle si subimos las predicciones. Para ello vamos a hacer uso de la función `createDataPartition` [13], a la cual se le proporciona el conjunto de datos a dividir especificando la variable a predecir, el número de muestras para el conjunto de entrenamiento y el número de particiones.

```
get_train_test<-function(datos, porcentaje_train) {
  # Mezclamos los datos aleatoriamente
  set.seed(32)
  datos<-datos[sample(1:nrow(datos)), ]
  # Dividimos el conjunto en train y test
  library(caret)
  train<-createDataPartition(datos$isFraud, p=porcentaje_train, list=FALSE, times=1)
  datos_train<-datos[train, ]
  datos_test<-datos[-train, ]
  # Devolvemos ambos dataframes en una lista
  df<-list()
  df$train<-datos_train
  df$test<-datos_test
  df
}
```

Modelos

En esta sección se definen las funciones tanto para entrenar los modelos predictivos como para evaluar su rendimiento. El objetivo es estudiar si la capacidad de generalización de los clasificadores mejora o empeora en función de los cambios que se realicen en el preprocesamiento de los datos. Por un lado voy a utilizar la librería `e1071` para entrenar modelos utilizando la técnica **Naive Bayes**, la cual se caracteriza por su rapidez y por estar orientada a la clasificación binaria. Mientras que como segunda técnica utilizaré **Random Forest** por ser más robusta y sofisticada que la anterior. Para ello haré uso de la librería de igual nombre `randomForest`.

Para medir la calidad de todos los modelos realizamos el mismo proceso: en primer lugar obtenemos las predicciones a partir del conjunto de validación y mediante la función `confusionMatrix` [14] se calculan varias medidas de calidad, como la matriz de confusión, la precisión, entre otros. Asimismo, se incluye una segunda función para dibujar la **curva ROC** y calcular el área bajo ella de modo que podamos visualizar otra medida de calidad de manera gráfica.

```
calidad_modelo<-function(modelo, test, etiquetas_test) {
  # Obtenemos las predicciones del modelo para el conjunto de test
  predicciones<-predict(modelo, test)
  # Obtenemos la matriz de confusión y la información
  confusion<-table(etiquetas_test, predicciones, dnn = c("Real", "Predicha"))
  resultado<-list()
  resultado$preds<-predicciones
  resultado$conf<-confusion
  resultado
}

curva_roc<-function(predicciones, etiquetas) {
```

```
  library("ROCR")
  # Calculamos el porcentaje de acierto entre las etiquetas predichas y las reales
  preds<-prediction(as.numeric(predicciones), as.numeric(etiquetas))
  # Comprobamos la eficacia del modelo considerando los falsos positivos y los aciertos
  curva<-performance(preds, "tpr", "fpr")
  # Dibujamos la curva.
  plot(curva, col="green", add=FALSE, main="Curva ROC", lwd = 2)
  segments(0, 0, 1, 1, col='black')
  grid()
  # Calculamos el área debajo de la curva
  curva.area = performance(preds, "auc")
  cat("\nEl área bajo la curva ROC es", curva.area@y.values[[1]]*100,"%\n")
}

entrenar_modelo<-function(tecnica, train, test, clase_positiva) {
  set.seed(32)
  # Entrenamos el modelo utilizando todas las características
  if (tecnica == "NB") {
    library(e1071)
    modelo<-naiveBayes(isFraud~., data = train)
  }
  else if (tecnica == "RF") {
    library(randomForest)
    modelo<-randomForest(formula=isFraud~., data = datos$train, ntree=100)
  }
  # Obtenemos las predicciones y matriz de confusión
  calidad<-calidad_modelo(modelo, test, test$isFraud)
  info_modelo<-confusionMatrix(calidad$conf, positive=clase_positiva)
  # Devolvemos la información del modelo y la calidad
  resultado<-list()
  resultado$calidad<-calidad
  resultado$info<-info_modelo
  resultado
}
```

Modelos del primer preprocesamiento

En este primer modelo vamos a comenzar eliminando aquellas columnas con más de un **50% de valores perdidos**. De este modo conseguimos reducir el número de variables de **434 a 305**, que si bien siguen siendo muchas hemos realizado una considerable reducción. A continuación, procedemos a eliminar las columnas con **desviación estándar=0** para continuar reduciendo la dimensionalidad del conjunto de modo que, añadiendo este segundo procesamiento, hemos conseguido reducir hasta las **301 columnas**.

```
# Eliminamos columnas con > 50% NA
train_m1<-eliminar_columnas_nas(train)
# Eliminamos columnas con sd=0
train_m1<-eliminar_columnas_sd(train_m1)
```

A continuación vamos a **tratar los valores perdidos** aplicando la primera técnica explicada anteriormente que consiste en **eliminar todos los registros con NA**. De este modo podemos comprobar cómo el conjunto de datos se ve grávemente afectado puesto que pasa de más de 100.000 muestras a disponer solo de aproximadamente **21.984 ejemplos**. Como ya anticipamos este método es muy agresivo pero cuando entrenemos los modelos comprobaremos si este hecho influye en su calidad.

Aplicando algunas técnicas para conocer la importancia de las 301 variables que disponemos me he dado cuenta de que al eliminar los valores perdidos algunas variables **vuelven a tener una desviación estándar igual a 0**. La razón de ser puede residir en que estas columnas solo disponían de un valor y de muchos NAs por lo que al eliminarlos ya no disponen de variabilidad en sus datos. Por lo tanto vamos a realizar aplicar por segunda vez la misma función para eliminarlas. El objetivo consiste en estudiar la correlación entre variables y con la variable a predecir para continuar reduciendo la dimensionalidad asociada a las columnas. Tras esta segunda pasada se han eliminado cuatro columnas más disponiendo ahora de **297 variables**.

```
# Eliminar registros con valores NA
train_m1<-eliminar_nas(train_m1)
# Volvemos a eliminar las nuevas columnas que han surgido con desviación estándar=0
# tras quitar valores perdidos
train_m1<-eliminar_columnas_sd(train_m1)
# Dimensión del conjunto resultante
cat("\nSegunda tanda para eliminar variables con sd=0\n")
```

```
##
## Segunda tanda para eliminar variables con sd=0
```

```
dim(train_m1)
```

```
## [1] 21984   297
```

```
# Correlación entre variables
train_m1<-correlacion_variables(train_m1, 0.75)
# Dimensión del conjunto resultante
cat("\nTras eliminar variables correladas > 0.75\n")
```

```
##
## Tras eliminar variables correladas > 0.75
```

```
dim(train_m1)
```

```
## [1] 21984    84
```

```
# Correlación entre las variables y la categórica `isFraud`
tabla_corr<-correlacion_fraude(train_m1)
head(tabla_corr, 10)
```

```
##    Variable isFraud
## 1   isFraud    1.00
## 2      V152    0.51
## 3      V247    0.47
## 4      V244    0.45
## 5      V248    0.41
## 6      V194    0.38
## 7      V253    0.35
## 8      V153    0.31
## 9      V250    0.24
## 10     V245    0.23
```

```r
# Obtenemos las variables que tienen un coeficiente de correlación muy bajo con respecto a isFraud
tabla_corr_baja<-tabla_corr %>% filter(isFraud > -0.1 & isFraud < 0.1)
# Las eliminamos del conjunto de datos haciendo una copia en otro para no perder el original
train_m1_corr<-train_m1[, -which(colnames(train_m1) %in% tabla_corr_baja$Variable)]
# Dimensión del conjunto resultante
cat("\nTras eliminar variables con correlación en [-0.1, 0.1] con respecto a isFraud\n")
```

```
##
## Tras eliminar variables con correlación en [-0.1, 0.1] con respecto a isFraud
```

```r
dim(train_m1_corr)
```

```
## [1] 21984    34
```

A continuación vamos a estudiar la **correlación entre variables**. Como se ha explicado anteriormente, nuestro objetivo es eliminar aquellas que proporcionen información redundante para así reducir más el número de columnas del *dataset*. Para ello vamos a establecer un umbral de **0.75** por el cual eliminaremos todas aquellas variables cuyos coeficientes sean mayores. Este procesamiento nos ha hecho bajar de 286 variables a solo **84 columnas**, por lo que podemos concluir que la gran mayoría de variables restantes proporcionaban la misma información.

En relación al análisis de **correlación entre las variables y la categórica `isFraud`** podemos visualizar los diez primeros resultados puesto que la tabla es considerablemente amplia como para mostrarla entera. Sin embargo, estudiándola en profundidad he podido ver que existen muchas columnas con menos de 0.1 de correlación. A priori un valor tan bajo indica que estas columnas no están relacionadas con `isFraud` aunque pueden aportar información útil a la hora de entrenar un clasificador. Para comprobar esta teoría vamos a entrenar estos primeros modelos **eliminando aquellas variables con una correlación en el intervalo [-0.1, 0.1]**. De este modo eliminamos pasamos a tener un *dataset* con solo **34 variables**, una dimensionalidad mucho más gestionable.

A continuación vamos a realizar un estudio de los **outliers** para las columnas restantes. De nuevo solo mostramos los 10 primeros resultados porque la lista es bastante larga. Sin embargo, visualizándola en profunidad he podido observar que la gran mayoría de columnas disponen de este tipo de valores en mayor o menor cantidad. Por ello vamos a aplicar el primero de los tratamientos explicados, consistente en transformarlos en el cuantil 5 o 95 en función de si se encuentran por debajo o por encima del rango de valores, respectivamente.

```r
# Estudio de los outliers
cat("\nOutliers antes del tratamiento\n")
```

```
##
## Outliers antes del tratamiento
```

```r
head(summary(estudio_outliers(train_m1_corr)), 10)
```

```
##         Length Class  Mode
## isFraud 1470   -none- numeric
## D8      1963   -none- numeric
## V110      66   -none- numeric
## V111      34   -none- numeric
## V114     211   -none- numeric
## V117      30   -none- numeric
```

```
## V121      49   -none- numeric
## V139    1968   -none- numeric
## V152     800   -none- numeric
## V153    2560   -none- numeric
```

```
# Tratamiento de outliers mediante la fórmula
train_m1_out<-outliers_formula(train_m1_corr)
# Estudio del conjunto resultante
cat("\nOutliers tras el tratamiento\n")
```

```
##
## Outliers tras el tratamiento
```

```
head(summary(estudio_outliers(train_m1_out)), 10)
```

```
##      Length Class  Mode
## D8   1963   -none- numeric
## V110    0   -none- numeric
## V111    0   -none- numeric
## V114    0   -none- numeric
## V117    0   -none- numeric
## V121    0   -none- numeric
## V139 1968   -none- numeric
## V152    0   -none- numeric
## V153 1989   -none- numeric
## V194 1229   -none- numeric
```

Una vez hemos tratado los *outliers* de todas las columnas excepto de `isFraud` volvemos a realizar un análisis para comprobar si aún existen este tipo de valores. Y tal y como podemos observar así es, algunas variables siguen disponiendo de este tipo de valores. Mi teoría es que si un valor no supera el umbral global a la variable, no se modifica y por lo tanto es ignorado. Por el momento vamos a mantener este conjunto de datos para posteriormente comparar los modelos entrenados con él y con otro conjunto al que le apliquemos la segunda técnica.

Como ya disponemos de un conjunto de variables más reducido, **no vamos a aplicar el PCA** puesto que nos arriesgamos a perder información a la hora de combinar las columnas. Por lo que procedemos directamente a realizar el último paso previo al entrenamiento de los modelos: **balancear la clase a predecir**. Para ello vamos a aplicar la primera técnica explicada denominada **under-sampling**, con el objetivo de disponer del mismo número de transacciones fraudulentas como de no fraudulentas. Tal y como podemos observar el conjunto se ha reducido considerablemente puesto que el número de muestras fraudulentas es súmamente menor que el de no fraudulentas. Y si además partimos de un conjunto con pocos datos pues al final el *dataset* se queda demasiado reducido.

A continuación dividimos el conjunto en entrenamiento y test para validar los modelos que vamos a entrenar con el objetivo de calcular varias medidas de calidad. Para ello vamos a separar el **70% para entrenamiento y el 30% para validación**. Si bien la práctica de aplicar el preprocesamiento sobre el conjunto de datos completo, afectando también al futuro conjunto de validación, no es técnicamente correcta, como facilita súmamente el trabajo es muy utilizada y por lo tanto yo también voy a trabajar de esta forma.

```
# Under-sampling para balancear la clase `isFraud`
train_m1_final<-under_sampling(train_m1_out)
cat("Conjunto de datos tras under-sampling\n")
```

```
## Conjunto de datos tras under-sampling
```

```r
dim(train_m1_final)
```

```
## [1] 2940   34
```

```r
# Dividimos el conjunto final en 70% entrenamiento y 30% test
datos<-get_train_test(train_m1_final, 0.7)
cat("\nConjunto de entrenamiento\n")
```

```
##
## Conjunto de entrenamiento
```

```r
dim(datos$train)
```

```
## [1] 2058   34
```

```r
cat("\nConjunto de test\n")
```

```
##
## Conjunto de test
```

```r
dim(datos$test)
```

```
## [1] 882  34
```

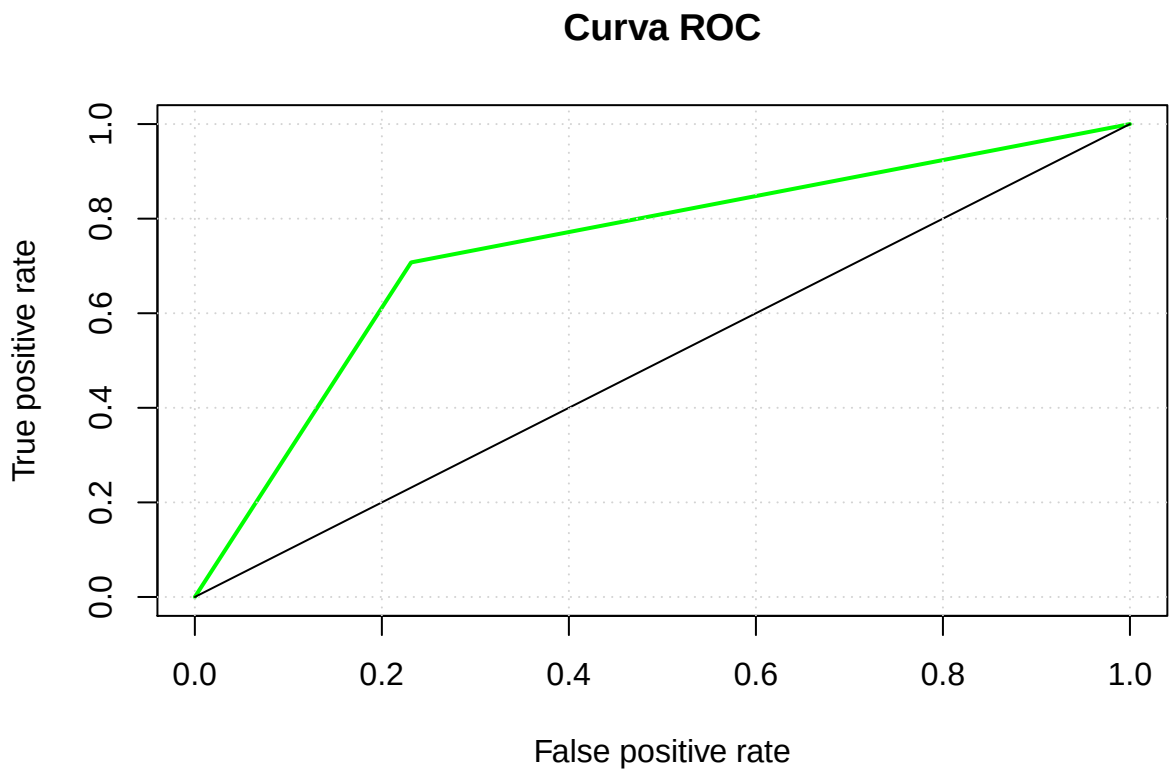
```r
cat("\n")
```

```r
# MODELO NAIVE BAYES
m1_nb<-entrenar_modelo('NB', datos$train, datos$test, '1')
m1_nb$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real   0   1
##    0 339 102
##    1 129 312
##
##                Accuracy : 0.7381
##                  95% CI : (0.7077, 0.7668)
##     No Information Rate : 0.5306
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4762
##
##  Mcnemar's Test P-Value : 0.08714
##
##             Sensitivity : 0.7536
##             Specificity : 0.7244
##          Pos Pred Value : 0.7075
```

```
##          Neg Pred Value : 0.7687
##              Prevalence : 0.4694
##          Detection Rate : 0.3537
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7390
##
##        'Positive' Class : 1
##
```

```r
# Dibujamos la curva ROC del modelo de Naive Bayes
curva_roc(m1_nb$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 73.80952 %
```

Si bien la documentación oficial de la función `confusionMatrix` no aporta información acerca de los atributos mostrados, he encontrado un ejemplo [15] en el que se detallan cada uno de ellos. En primer lugar destacamos que la **precisión del modelo es de casi un 74%**, valor que se encuentra dentro de un intervalo al 95% de confianza lo que nos indica que este valor es verídico. Por otro lado podemos observar en el atributo *No Information Rate* cómo la función ha detectado que ambas clases están balanceadas puesto que no existe una mayor representación de una clase. Este indicio explica que el clasificador no se ve influido por una clase u otra.

Si nos fijamos en un campo denominado *Kappa* podemos conocer cuán bueno es el clasificador entrenado frente a otro que escoge las etiquetas al azar. El objetivo es conocer si el clasificador obtenido, en este caso por

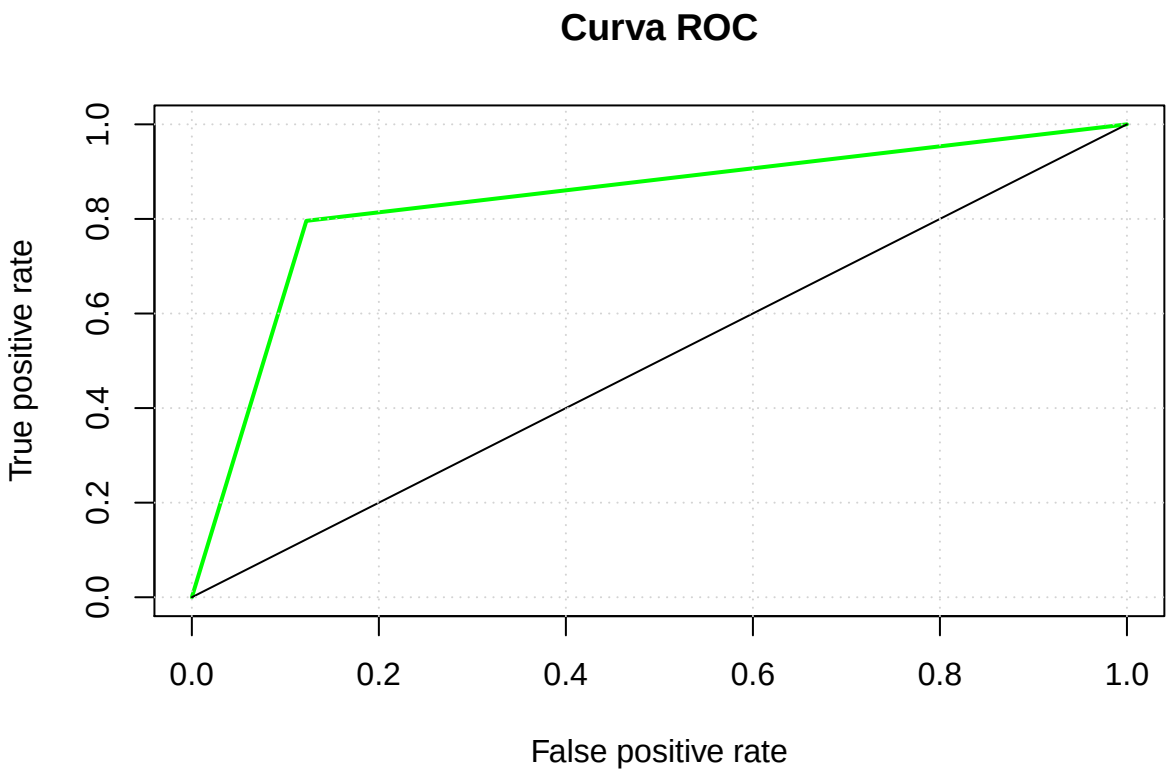
Naive Bayes, realmente está prediciendo las etiquetas o las clasifica aleatoriamente. Investigando más acerca de la repercusión de este valor [16] se considera que un modelo es confiable cuando este valor se encuentra **por encima del 80%**, lo que significa que apenas presenta cierta aleatoriedad en sus clasificaciones. En este caso podemos apreciar que nuestro valor es del 47%, lo que significa que en más de **la mitad de las etiquetas coincide con un clasificador aleatorio**. Tal y como se explica en la fuente [16], la gravedad de estos resultados dependen del ámbito en el que se encuentre y considerando la capacidad de detectar transacciones fraudulentas, podemos determinar que **este clasificador no es para nada competitivo** dentro de un ámbito real.

Visualizando la curva ROC podemos comprobar la teoría anterior puesto que tal y como podemos observar el clasificador no dispone de una gran capacidad de generalización puesto que su área apenas llega a un 74%.

```
# MODELO RANDOM FOREST
m1_rf<-entrenar_modelo('RF', datos$train, datos$test, '1')
m1_rf$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real   0   1
##    0 387  54
##    1  90 351
##
##                Accuracy : 0.8367
##                  95% CI : (0.8107, 0.8605)
##     No Information Rate : 0.5408
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6735
##
##  Mcnemar's Test P-Value : 0.003538
##
##             Sensitivity : 0.8667
##             Specificity : 0.8113
##          Pos Pred Value : 0.7959
##          Neg Pred Value : 0.8776
##              Prevalence : 0.4592
##          Detection Rate : 0.3980
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8390
##
##        'Positive' Class : 1
##
```

```
curva_roc(m1_rf$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 83.67347 %
```

En el caso del clasificador entrenado con **Random Forest** podemos observar que cuenta con una **precisión del 83%**, 9 puntos más que el modelo anterior. Este aumento en la precisión acompaña a un descenso de falsos positivos en la matriz de confusión. Sin embargo, analizando de nuevo el resto de atributos, en este caso el **coeficiente de *Kappa* es de 67%**. Si bien sigue sin llegar al recomendable 80%, podemos afirmar que este modelo demuestra que sus decisiones no son tan aleatorias como en el caso anterior. Aún así, con **este preprocesamiento ninguno de los modelos entrenados son competitivos**.

Si además visualizamos la curva ROC de este modelo podemos apreciar una mayor amplitud, lo que produce un **área mayor bajo la curva**, por lo que bajo el mismo preprocesamiento de los conjuntos esta técnica ha conseguido un clasificador con una mejor capacidad de generalización frente a la técnica Naive Bayes.

Modelos del segundo preprocesamiento

En este segundo preprocesamiento vamos a partir del conjunto de datos de entrenamiento original. El primer paso que vamos a realizar es **balancear la clase `isFraud`** y así también reducir la dimensionalidad del conjunto. El objetivo consiste en preprocesar los datos con las clases ya balanceadas.

```
# Balanceamos las clases tomando todas las transacciones
# positivas y el mismo número de transacciones negativas
train_m2<-under_sampling(train)
# Comprobamos que ambas clases se encuentran balanceadas
cat("Transacciones NO fraudulentas (clase 0)\n")
```

```
## Transacciones NO fraudulentas (clase 0)
```

```
length(train_m2$isFraud[train_m2$isFraud==0])
```

```
## [1] 11318
```

```
cat("\nTransacciones SI fraudulentas (clase 1)\n")
```

```
##
## Transacciones SI fraudulentas (clase 1)
```

```
length(train_m2$isFraud[train_m2$isFraud==1])
```

```
## [1] 11318
```

Como podemos observar hemos pasado de tener más de 100.000 registros a solo disponer de 11.318 en cada clase lo que hace un total de **22.636 muestras**. Una vez disponemos del conjunto de datos balanceado, comenzamos a eliminar las columnas con más de un 50% de NAs y aquellas con desviación estándar igual a 0, como hicimos en el primer procesamiento.

```
# Eliminamos las columnas con > 50% NAs
train_m2<-eliminar_columnas_nas(train_m2)
# Eliminamos las columnas con sd=0
train_m2<-eliminar_columnas_sd(train_m2)
```

Tal y como se puede comprobar hemos pasado de las 434 variables originales a **324 columnas**. A continuación, vamos a introducir una nueva modificación de modo que **no eliminaremos los valores perdidos o NAs, sino que los imputaremos utilizando la función `mice`**. Con ella se puede especificar el método a utilizar para imputar los valores perdidos así como el número de imputaciones e iteraciones por cada una de ellas. En relación al primer parámetro he probado en primer lugar *rf* (*Random Forest*), pero debido a la poca variabilidad de los datos esta técnica no ha producido buenos resultados. Otro método sofisticado es *cart*, el cual es capaz de escoger clasificación o árboles de decisión en función de la naturaleza de la columna. Existen también otras técnicas más rápidas y sencillas como la media de los valores (*mean*) o asignar un valor aleatorio dentro del rango de valores de la columna (*sample*).

```
cat("Dimensiones tras borrar columnas > 50% NAs y con sd=0\n")
```

```
## Dimensiones tras borrar columnas > 50% NAs y con sd=0
```

```
dim(train_m2)
```

```
## [1] 22636   324
```

En primer lugar cabe destacar que este proceso es **muy costoso computacionalmente** por la gran cantidad de columnas que disponemos. Asimismo, cuanto más sofisticado sea el método a aplicar, más tiempo tardará en imputar los valores NA. Según esta fuente [17] a mayor número de imputaciones e iteraciones mejor resultado obtendremos al predecir los valores perdidos. Sin embargo, tras realizar muchas pruebas he comprobado que para ciertas columnas **no existe suficiente información como para imputar todos los NAs** y que independientemente del número de iteraciones/imputaciones, el *dataset* resultante sigue conteniendo valores perdidos. Por lo tanto, a continuación se aplica la configuración que mejor relación calidad~tiempo he conseguido para imputar la mayoría de los valores NAs. Con ella pasamos de tener **988.826 valores perdidos a 11.003**. Como anteriormente he comentado, este proceso es súmamente costoso y por ello solo se ha realizado una vez guardando en un fichero el *dataset* obtenido para sus posteriores usos.

```r
# Primera imputación de NAs utilizando clasificación y arboles de decisión ('cart')
train_m2_cart<-imputar_nas(train_m2, 1, 1, "cart")
# Almacenamos el dataset resultante en un fichero csv
write.csv(train_m2_cart,'./ficheros/train_m2_cart.csv', row.names = FALSE, col.names=TRUE)
```

Una vez disponemos del conjunto imputado mediante la función `mice`, vamos a calcular la media para cada una de las columnas que aún disponen de NAs. Lo he intentado de nuevo con esta función pero incluso con un método más sencillo como la media siguen quedando algunos valores perdidos en ciertas columnas. Por ello lo voy a realizar con una función particular que terminará de limpiar el *dataset* de valores perdidos. A continuación, eliminamos aquellas variables con un coeficiente de **correlación mayor que 0.75**. De este modo pasamos a tener **151 variables**.

En este segundo preprocesamiento, a diferencia del anterior, no vamos a estudiar la correlación variables~`isFraud` sino que vamos a aplicar el **algoritmo PCA** para reducir el número de variables. Sin embargo, previamente vamos a sustituir los *outliers* con la función que implementa la fórmula, la cual también ha sido aplicada en el anterior preprocesamiento.

```r
# Cargamos el dataset desde el fichero
train_m2_cart<-read.csv(file="./ficheros/train_m2_cart.csv", header=TRUE, sep=",")
# Imputación de NAs
cat("Número de NAs tras la primera imputación con 'cart'\n")
```

```
## Número de NAs tras la primera imputación con 'cart'
```

```r
sum(is.na(train_m2_cart))
```

```
## [1] 11003
```

```r
cat("\nVariables con NAs\n")
```

```
##
## Variables con NAs
```

```r
cols_nas<-colnames(train_m2_cart)[!complete.cases(t(train_m2_cart))]
cols_nas
```

```
## [1] "D12"  "V101" "V103" "V132" "V133" "V293" "V295"
```

```r
# Declaramos una función para calcular la media y asignársela a los valores NA de una columna
## El objetivo es poder aplicar el procedimiento a todo el dataset
asignar_media<-function(x) replace(x, is.na(x), mean(x, na.rm=TRUE))
train_m2_mean<-replace(train_m2_cart, TRUE, lapply(train_m2_cart, asignar_media))
cat("\nNúmero de NAs tras asignar la media\n")
```

```
##
## Número de NAs tras asignar la media
```

```r
sum(is.na(train_m2_mean))
```

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

```r
# Correlación entre variables
train_m2_mean<-correlacion_variables(train_m2_mean, 0.75)
# Dimensión del conjunto resultante
cat("\nTras eliminar variables correladas > 0.75\n")
```

```
##
## Tras eliminar variables correladas > 0.75
```

```r
dim(train_m2_mean)
```

```
## [1] 22636   151
```

Como podemos apreciar tras el tratamiento de *outliers* siguen apareciendo algunos en ciertas columnas. De nuevo, los vamos a dejar en el *dataset* resultante. Una segunda consecuencia de haber tratado los *outliers* consiste en la generación de **nuevas columnas con desviación estándar igual a 0**. Este hecho ha sido descubierto al obtener un error aplicando el PCA referente a la imposibilidad de tratar variables con desviación estándar 0. Por lo que volvemos a eliminar este tipo de columnas una vez más. De este modo hemos pasado de disponer de **151 variables a 125**. Como todavía es un número considerable de variables, el siguiente paso es **aplicar el PCA** para realizar combinaciones de las anteriores y así reducir su dimensionalidad.

```r
# Estudio de los outliers
cat("\nOutliers antes del tratamiento\n")
```

```
##
## Outliers antes del tratamiento
```

```r
head(summary(estudio_outliers(train_m2_mean)), 10)
```

```
##                Length Class  Mode
## TransactionDT       0   -none- numeric
## TransactionAmt   1732   -none- numeric
## card1               0   -none- numeric
## card2               0   -none- numeric
## card3               0   -none- numeric
## card4             973   -none- numeric
## card5               0   -none- numeric
## card6               0   -none- numeric
## P_emaildomain    6723   -none- numeric
## R_emaildomain    6471   -none- numeric
```

```r
# Tratamiento de outliers mediante la fórmula
train_m2_out<-outliers_formula(train_m2_mean)
# Estudio del conjunto resultante
cat("\nOutliers tras el tratamiento\n")
```

```
##
## Outliers tras el tratamiento
```

```
head(summary(estudio_outliers(train_m2_out)), 10)
```

```
##                 Length Class  Mode
## TransactionDT      0   -none- numeric
## TransactionAmt  1732   -none- numeric
## card1              0   -none- numeric
## card2              0   -none- numeric
## card3              0   -none- numeric
## card4              0   -none- numeric
## card5              0   -none- numeric
## card6              0   -none- numeric
## P_emaildomain   6723   -none- numeric
## R_emaildomain   6471   -none- numeric
```

```
# Eliminamos las nuevas columnas con sd=0
train_m2_out<-eliminar_columnas_sd(train_m2_out)
cat("\nDimensión tras eliminar de nuevo columnas con sd=0\n")
```

```
##
## Dimensión tras eliminar de nuevo columnas con sd=0
```

```
dim(train_m2_out)
```

```
## [1] 22636   125
```

```
# Aplicamos el PCA con un 80% de máxima varianza
# según el estudio realizado.
train_pca<-pca(train_m2_out, 0.8)
cat("\nDimensión tras aplicar el PCA con 80% de varianza\n")
```

```
##
## Dimensión tras aplicar el PCA con 80% de varianza
```

```
dim(train_pca)
```

```
## [1] 22636    49
```

Si bien se puede especificar la máxima varianza que deben explicar las variables, para determinar el valor con mejor relación *calidad~número de variables* he entrenado modelos con valores 0.7, 0.8 y 0.9. Los resultados obtenidos se pueden visualizar en las siguientes tablas para Naive Bayes y Random Forest, respectivamente.

Atributos	0.7	0.8	0.9
Nº variables	35	49	70
Precisión	70%	71%	71%
Kappa	41%	42%	43%

Atributos	0.7	0.8	0.9
Nº variables	35	49	70
Precisión	80%	81%	81%
Kappa	61%	62%	63%

Tal y como se puede observar, no existen grandes diferencias entre los tres umbrales de máxima varianza. Existe cierta mejoría del 70% al 80% tanto en *accuracy* como en el coeficiente de *Kappa* mientras que el aumento del número de variables no es demasiado significativo. Por el contrario, entre el 80%-90% sí podemos observar que el número de columnas se incrementa bastante en relación con la poca mejoría que conlleva. Por lo tanto para aplicar el algoritmo **PCA se establece un umbral del 80%**.

```
# Transformamos la variable a predecir de numérica a factor para
# que los modelos la detecten y realicen las predicciones
train_pca$isFraud<-as.factor(train_pca$isFraud)
# Dividimos el conjunto final en 70% entrenamiento y 30% test
datos<-get_train_test(train_pca, 0.7)
# Mostramos el resultado
cat("Conjunto de entrenamiento\n")
```

```
## Conjunto de entrenamiento
```

```
dim(datos$train)
```

```
## [1] 15846    49
```

```
cat("\nConjunto de test\n")
```

```
##
## Conjunto de test
```

```
dim(datos$test)
```

```
## [1] 6790   49
```

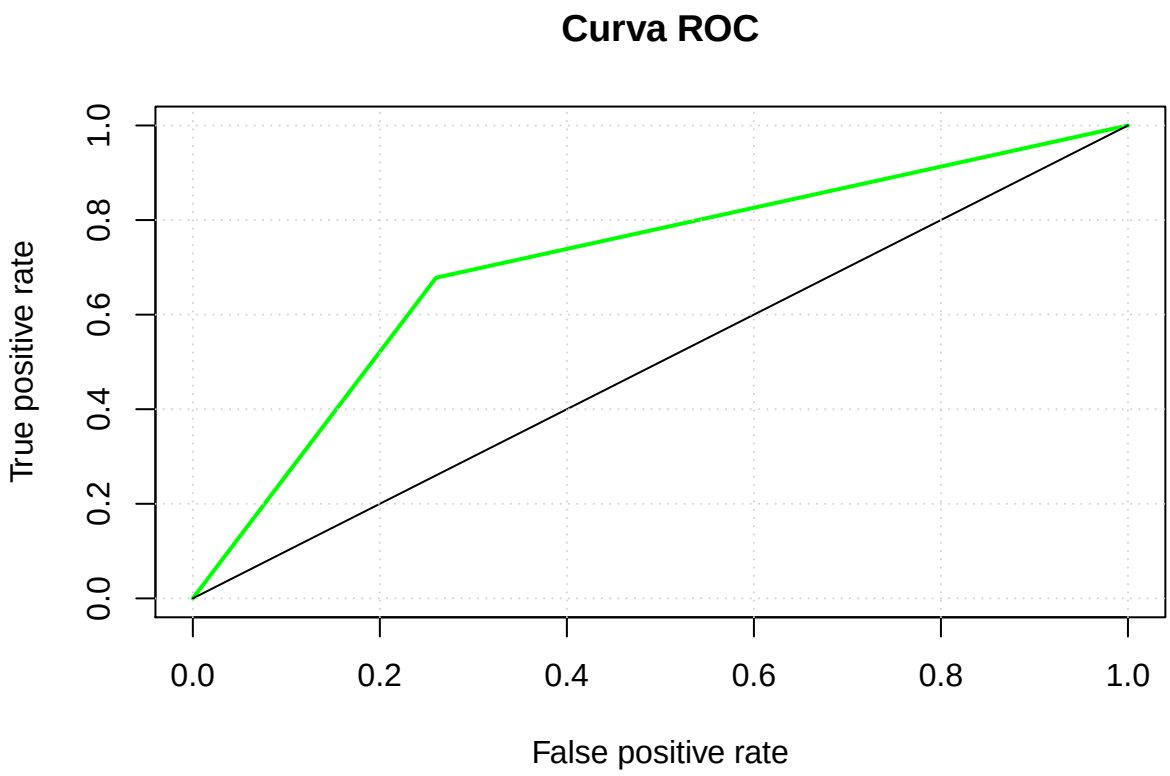
```
cat("\n")
```

```
# MODELO NAIVE BAYES
m2_nb<-entrenar_modelo('NB', datos$train, datos$test, '1')
m2_nb$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 2512  883
##    1 1093 2302
##
##                Accuracy : 0.709
##                  95% CI : (0.698, 0.7198)
```

```
##     No Information Rate : 0.5309
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.418
##
##  Mcnemar's Test P-Value : 2.58e-06
##
##             Sensitivity : 0.7228
##             Specificity : 0.6968
##          Pos Pred Value : 0.6781
##          Neg Pred Value : 0.7399
##              Prevalence : 0.4691
##          Detection Rate : 0.3390
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7098
##
##        'Positive' Class : 1
##
```

```
curva_roc(m2_nb$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 70.89838 %
```

El modelo entrenado con **Naive Bayes** ha obtenido una **precisión del 70%** bajo un coeficiente de **Kappa del 41%**. En comparación con los resultados del anterior modelo podemos observar que este ha empeorado

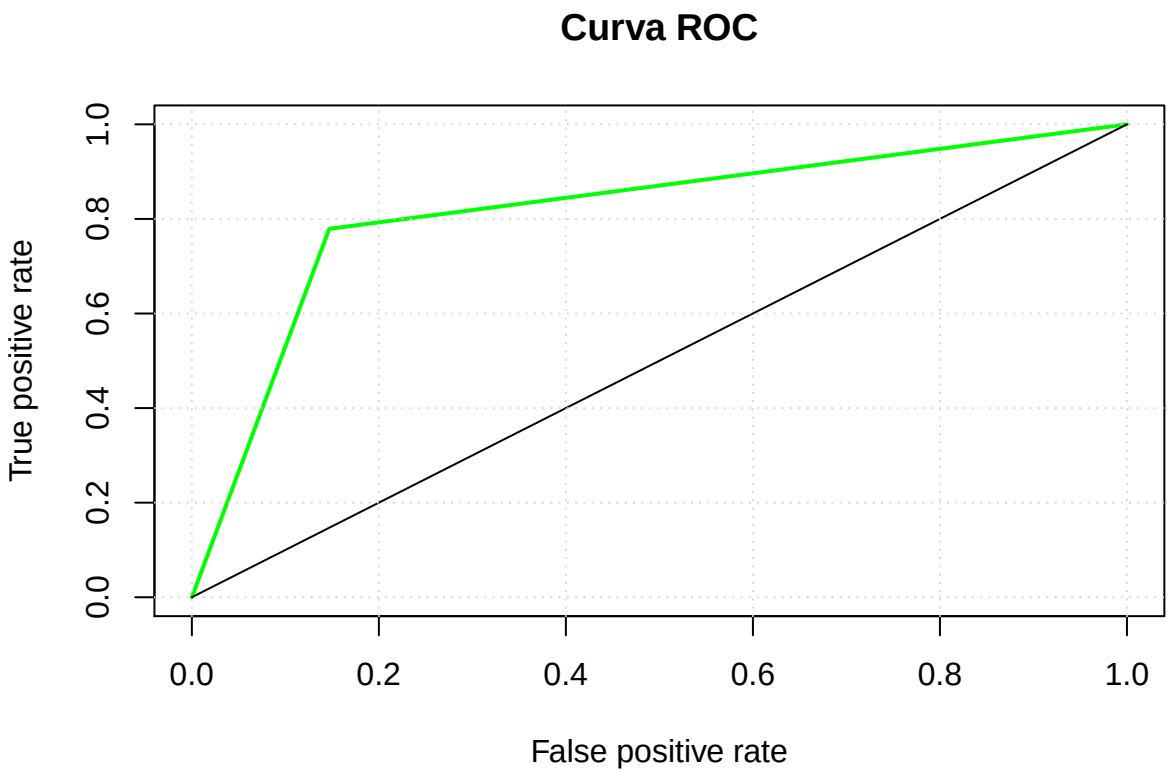
ligeramente puesto que el valor de Kappa es bastante más bajo que el anterior. Acompañando a este fenómeno se encuentra la matriz de confusión en la que podemos visualizar que prácticamente se equivoca en un 30% indicando que una transacción es fraudulenta cuando no lo es, y algo más para el caso contrario.

En la curva ROC podemos observar que la capacidad de generalización de este clasificador es algo más baja que en la del primer preprocesamiento, aunque son bastante similares. Si resumimos los datos podemos determinar que pese a haber introducido tres cambios con respecto al preprocesamiento anterior (no estudiar la correlación variables~isFraud, imputar NAs en lugar de borrarlos y utilizar el PCA para eliminar más columnas) este clasificador presenta un comportamiento muy parecido puesto que **sigue fallando en aproximádamente un 30% de ocasiones**. Tanto el valor de *Kappa* como la precisión han disminuido puesto que el volumen de datos de validación es mayor, y por lo tanto, con un grupo más grande de datos se pueden realizar un mayor número de tests y por ende se computa más fallos.

```
# MODELO RANDOM FOREST
m2_rf<-entrenar_modelo('RF', datos$train, datos$test, '1')
m2_rf$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 2896  499
##    1  750 2645
##
##                Accuracy : 0.8161
##                  95% CI : (0.8066, 0.8252)
##     No Information Rate : 0.537
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6321
##
##  Mcnemar's Test P-Value : 1.506e-12
##
##             Sensitivity : 0.8413
##             Specificity : 0.7943
##          Pos Pred Value : 0.7791
##          Neg Pred Value : 0.8530
##              Prevalence : 0.4630
##          Detection Rate : 0.3895
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8178
##
##        'Positive' Class : 1
##
```

```
curva_roc(m2_rf$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 81.6053 %
```

En el caso del modelo entrenado con **Random Forest** podemos apreciar que los resultados también son peores con respecto al del primer preprocesamiento. En este caso se obtiene una **precisión del 81%** con un coeficiente de **Kappa aproximadamente del 63%**. De nuevo se puede observar un aumento del número de errores al clasificar una transacción fraudulenta como una que no lo es. Por lo que si en ambas técnicas sucede quiere decir que los datos no son lo suficientemente representativos como para terminar de diferenciar ambas clases. Sin embargo, este clasificador no ha empeorado tanto como el de Naive Bayes basándome en el valor de *Kappa*, puesto que si bien no alcanza el valor óptimo del 80%, no ha bajado 20 puntos como el anterior. Esto nos indica que *Random Forest* es una técnica más robusta frente a la variación de los datos. Este hecho acompaña a la visualización de la **curva ROC** puesto que es muy parecida a la del primer preprocesamiento ya que su área solo ha descendido dos puntos.

Conclusiones

Hasta el momento hemos realizado dos preprocesamientos diferentes. Las conclusiones que podemos determinar comparando los modelos entrenados son las siguientes:

- **No es beneficioso submuestrear el conjunto al comienzo** del preprocesamiento puesto que existen muy pocas transacciones positivas y al realizarlo de forma aleatoria podemos perder muchas de ellas, como ha surgido en la primera combinación.
- Aunque es una técnica mucho más costosa computacionalmente y puede presentar ciertas dificultades es **mejor imputar los valores perdidos** que eliminar las filas que los contengan. La razón de ello

es que se pierden muchos registros y se reduce drásticamente el conjunto de datos con el que entrenar y validar.

- La **fórmula** utilizada para tratar los *outliers* tiene la desventaja de que **algunos de ellos son ignorados** si no superan un determinado umbral. Si bien es un método muy sencillo para una primera aproximación, puede que la imputación resulte ser una mejor técnica.
- **Naive Bayes** es una técnica sencilla y rápida con la que entrenar un clasificador binario pero es muy sensible a la variación de los datos por lo que sus clasificadores no tienen buena capacidad de predicción. Mientras que **Random Forest** es un poco más costosa computacionalmente pero mucho más robusta a las cambios de los datos. Por ello sus clasificadores tienen mejor capacidad de predicción y, por el momento, no han presentado diferencias significativas en ambos preprocesamientos.

Modelos del tercer preprocesamiento

En esta tercera combinación de técnicas de preprocesamiento vamos a comprobar la influencia que puede tener eliminar aquellas columnas con un **bajo coeficiente de correlación** (entre -0.1 y 0.1) con la variable a predecir dentro de un **conjunto de datos más amplio**. Asimismo, en esta sección se va a aplicar la segunda técnica para tratar *outliers* mediante **imputación**. De este modo comprobamos la efectividad de este segundo método con respecto a la fórmula.

En este caso partimos del fichero creado en el preprocesamiento anterior puesto que de nuevo las tres técnicas a aplicar son *downsampling*, eliminación de columnas con más de un 50% NAs y con desviación estándar igual a 0. Por tanto reutilizamos el *dataset* creado anteriormente.

```
library(magrittr)
library(dplyr)
# Cargamos el dataset desde el fichero
train_m3<-read.csv(file="./ficheros/train_m2_cart.csv", header=TRUE, sep=",")
# Terminamos de imputar los NAs restantes con la media
train_m3<-replace(train_m3, TRUE, lapply(train_m3, asignar_media))
cat("Dimensión del conjunto de datos inicial.\n")
```

```
## Dimensión del conjunto de datos inicial.
```

```
dim(train_m3)
```

```
## [1] 22636   324
```

```
cat("\nNúmero de NAs tras asignar la media\n")
```

```
##
## Número de NAs tras asignar la media
```

```
sum(is.na(train_m3))
```

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

```
# Correlación entre variables
train_m3<-correlacion_variables(train_m3, 0.75)
# Dimensión del conjunto resultante
cat("\nTras eliminar variables correladas > 0.75\n")
```

```
##
## Tras eliminar variables correladas > 0.75
```

```
dim(train_m3)
```

```
## [1] 22636   151
```

Tras realizar los pasos anteriores además de eliminar aquellas variables con una correlación superior a 0.75, disponemos de un *dataset* de **151 variables**. A continuación eliminaremos aquellas columnas con una correlación entre [-0.1, 0.1] con respecto a la variable a predecir `isFraud`. Con esto conseguimos reducir drásticamente las variables a **53 columnas**. Como disponemos de un número bastante asequible, en este tercer preprocesamiento no vamos a aplicar el algoritmo PCA. Pasamos directamente al tratamiento de *outliers* mediante imputación.

```
# Correlación variables~isFraud
tabla_corr<-correlacion_fraude(train_m3)
head(tabla_corr, 10)
```

```
##    Variable isFraud
## 1   isFraud    1.00
## 2     card3    0.27
## 3      V188    0.25
## 4       V87    0.22
## 5       V52    0.21
## 6       V51    0.20
## 7      V123    0.20
## 8      V186    0.20
## 9       V44    0.18
## 10     V229    0.18
```

```
# Obtenemos las variables que tienen un coeficiente de correlación muy bajo con respecto a isFraud
tabla_corr_baja<-tabla_corr %>% filter(isFraud > -0.1 & isFraud < 0.1)
# Las eliminamos del conjunto de datos haciendo una copia en otro para no perder el original
train_m3_corr<-train_m3[, -which(colnames(train_m3) %in% tabla_corr_baja$Variable)]
# Dimensión del conjunto resultante
cat("\nTras eliminar variables con correlación en [-0.1, 0.1] con respecto a isFraud\n")
```

```
##
## Tras eliminar variables con correlación en [-0.1, 0.1] con respecto a isFraud
```

```
dim(train_m3_corr)
```

```
## [1] 22636    53
```

Como se explicó anteriormente, en este paso vamos a convertir todos los *outliers* detectados en valores perdidos para luego imputarlos utilizando la función `mice`. Para ello vamos a repetir el procedimiento realizado con los valores perdidos originales del *dataset*, es decir, al principio vamos a aplicar el método *cart (classification and regression trees)* que es más sofisticado. Luego comprobaremos si todavía existen valores NAs para imputarlos a continuación con la media.

En este caso, como existe un menor número de columnas que imputando los NAs originales la función invierte mucho menos tiempo. Sin embargo, de nuevo encuentra bastantes dificultades para imputar todos los valores perdidos introducidos por lo que es necesario transformar los restantes con la media de las columnas afectadas.

```
# Imputamos los NAs introducidos en lugar de los outliers
train_m3_out<-outliers_prediccion(train_m3_corr, 1, 1, 'cart')
# Calculamos la media para el resto de los NAs
train_m3_mean<-replace(train_m3_out, TRUE, lapply(train_m3_out, asignar_media))
```

Como podemos apreciar tras el doble tratamiento de *outliers* no quedan valores perdidos residuales pero sí que aparecen nuevos *outliers* tras realizar el preprocesamiento. Si bien he estado probando varias técnicas para aplicar este tratamiento, en todas he obtenido los mismos resultados. Por lo que la conclusión a la que llego es que estos **nuevos *outliers* se pueden generar por alguno de los procesos aplicados**: la imputación o la media.

```
cat("Número de NAs tras asignar la media\n")
```

```
## Número de NAs tras asignar la media
```

```
sum(is.na(train_m3_mean))
```

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

```
# Estudio del conjunto resultante
cat("\nOutliers tras el tratamiento\n")
```

```
##
## Outliers tras el tratamiento
```

```
head(summary(estudio_outliers(train_m3_mean)),10)
```

```
##               Length Class  Mode
## TransactionDT      0   -none- numeric
## card2              0   -none- numeric
## card3              0   -none- numeric
## D1                 0   -none- numeric
## D4              5508   -none- numeric
## D8              1450   -none- numeric
## V14                0   -none- numeric
## V16                0   -none- numeric
## V24                0   -none- numeric
## V31                0   -none- numeric
```

```
# Transformamos la variable a predecir de numérica a factor para
# que los modelos la detecten y realicen las predicciones
train_m3_mean$isFraud<-as.factor(train_m3_mean$isFraud)
# Dividimos el conjunto final en 70% entrenamiento y 30% test
datos<-get_train_test(train_m3_mean, 0.7)
# Mostramos el resultado
cat("Conjunto de entrenamiento\n")
```

```
## Conjunto de entrenamiento
```

```r
dim(datos$train)
```

```
## [1] 15846    53
```

```r
cat("\nConjunto de test\n")
```

```
##
## Conjunto de test
```

```r
dim(datos$test)
```

```
## [1] 6790   53
```

```r
cat("\n")
```

Tal y como podemos observar en el nuevo clasificador entrenado con **Naive Bayes** los resultados son casi iguales a los del segundo preprocesamiento en cuanto al *accuracy* y al coeficiente de *Kappa*. Sin embargo, en relación a la matriz de confusión se puede observar una menor tasa de falsos positivos mientras que por el contrario aumentan el número de transacciones fraudulentas clasificadas como no fraudulentas.

```r
# MODELO NAIVE BAYES
m3_nb<-entrenar_modelo('NB', datos$train, datos$test, '1')
m3_nb$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 2626  769
##    1 1201 2194
##
##                Accuracy : 0.7099
##                  95% CI : (0.6989, 0.7206)
##     No Information Rate : 0.5636
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4197
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7405
##             Specificity : 0.6862
##          Pos Pred Value : 0.6462
##          Neg Pred Value : 0.7735
##              Prevalence : 0.4364
##          Detection Rate : 0.3231
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7133
##
##        'Positive' Class : 1
##
```

En el caso del nuevo clasificador con **Random Forest** podemos apreciar un considerable **empeoramiento de los resultados** puesto que la precisión obtenida se encuentra un 4% por debajo que en el segundo preprocesamiento y lo que es más preocupante aún, el coeficiente de *Kappa* disminuye hasta un 58% cuando en el modelo de la sección anterior se situaba 9 puntos por encima.

```
# MODELO RANDOM FOREST
m3_rf<-entrenar_modelo('RF', datos$train, datos$test, '1')
m3_rf$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 2677  718
##    1  685 2710
##
##                Accuracy : 0.7934
##                  95% CI : (0.7835, 0.8029)
##     No Information Rate : 0.5049
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5867
##
##  Mcnemar's Test P-Value : 0.3929
##
##             Sensitivity : 0.7905
##             Specificity : 0.7963
##          Pos Pred Value : 0.7982
##          Neg Pred Value : 0.7885
##              Prevalence : 0.5049
##          Detection Rate : 0.3991
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7934
##
##        'Positive' Class : 1
##
```

Para comprobar qué factor es el que ha desencadenado este empeoramiento, especialmente en el caso del clasificador entrenado con Random Forest, vamos a realizar pequeñas modificaciones con las que volver a entrenar nuevos modelos y así poder compararlos con los anteriores.

Variación: outliers fórmula vs. imputación

En esta primera modificación del tercer preprocesamiento vamos a repetir el proceso desde el tratamiento de los *outliers* de modo que en lugar de hacerlo mediante la imputación, vamos a aplicar la segunda técnica que emplea la fórmula. A continuación entrenaremos de nuevo los modelos para conocer la influencia que tiene el tratamiento de los *outliers* sobre la capacidad de predicción de los clasificadores.

```
# Tratamiento de outliers con la fórmula
train_m3_modif<-outliers_formula(train_m3_corr)
# Transformamos la variable a predecir en factor
train_m3_modif$isFraud<-as.factor(train_m3_modif$isFraud)
```

```r
# Dividimos el conjunto final en 70% entrenamiento y 30% test
datos<-get_train_test(train_m3_modif, 0.7)

# MODELO NAIVE BAYES
m3_1_nb<-entrenar_modelo('NB', datos$train, datos$test, '1')
cat("MODELO NAIVE BAYES\n")
```

```
## MODELO NAIVE BAYES
```

```r
m3_1_nb$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 2883  512
##    1 1338 2057
##
##                Accuracy : 0.7275
##                  95% CI : (0.7168, 0.7381)
##     No Information Rate : 0.6216
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4551
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8007
##             Specificity : 0.6830
##          Pos Pred Value : 0.6059
##          Neg Pred Value : 0.8492
##              Prevalence : 0.3784
##          Detection Rate : 0.3029
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.7419
##
##        'Positive' Class : 1
##
```

```r
# MODELO RANDOM FOREST
m3_1_rf<-entrenar_modelo('RF', datos$train, datos$test, '1')
cat("\nMODELO RANDOM FOREST\n")
```

```
##
## MODELO RANDOM FOREST
```

```r
m3_1_rf$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
```

```
## Real    0    1
##    0 2955  440
##    1  596 2799
##
##                Accuracy : 0.8474
##                  95% CI : (0.8386, 0.8559)
##     No Information Rate : 0.523
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6948
##
##  Mcnemar's Test P-Value : 1.467e-06
##
##             Sensitivity : 0.8642
##             Specificity : 0.8322
##          Pos Pred Value : 0.8244
##          Neg Pred Value : 0.8704
##              Prevalence : 0.4770
##          Detection Rate : 0.4122
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8482
##
##        'Positive' Class : 1
##
```

Como podemos apreciar ambos clasificadores disponen de **mejores resultados aplicando el tratamiento con la fórmula** que mediante imputación. Estas mejoras se traducen tanto en una precisión mayor como en un coeficiente de *Kappa* más elevado. Por lo tanto podemos determinar que de entre las dos técnicas consideradas para tratar este tipo de valores, la que mejor es capaz de asignar un nuevo valor a los *outliers* es la que utiliza la fórmula de los cuantiles.

Conclusiones

En esta subsección vamos a analizar los efectos producidos por el segundo y el tercer tipo de preprocesamiento para seguir extrayendo conclusiones en cuanto al tratamiento del conjunto de datos y su influencia sobre los clasificadores.

- Pese a determinar en la anterior sección de conclusiones que el tratamiento de *outliers* mediante la **fórmula** podría no ser una buena técnica, en la variación de este tercer preprocesamiento hemos comprobado que **es mejor que la imputación**, puesto que esta última tiene un grave impacto sobre la capacidad de predicción de los clasificadores. Y es que si bien se aplicó en primer lugar la función `mice` utilizando clasificación y árboles de decisión, no se puedieron imputar todos los valores perdidos ni con la combinación que muestro en esta memoria ni con las que he probado y no he incluido. Por ello se debía de calcular la media para cada columna afectada de modo que eliminásemos los NAs restantes. Así que la conclusión que podemos extraer de este caso es que las técnicas más sofisticadas no siempre tienen que dar mejores resultados que las más sencillas.
- En relación a la correlación y el PCA, si comparamos los resultados obtenidos en esta última modificación con los del segundo preprocesamiento podemos observar que los clasificadores entrenados previa aplicación del **PCA proporcionan peores resultados** que estudiando la correlación entre las variables~`isFraud`. Especialmente en el caso de los entrenados con *Random Forest* que llegan a bajar hasta 10 puntos tanto en precisión como en el coeficiente de *Kappa*. Este hecho nos indica que para nuestro problema no es beneficioso aplicar el algoritmo PCA puesto que el *dataset* que obtiene solo dispone de

columnas que son combinaciones lineales de las originales. Por ello, estamos perdiendo información de las estas últimas que al parecer es más útil para obtener clasificadores más competentes.

Modelos del cuarto preprocesamiento

Una vez conocemos la combinación de técnicas de preprocesamiento que mejores resultados proporciona hasta el momento, en esta sección vamos a introducir una nueva modificación. Consiste en generar un mayor número de datos con el objetivo de comprobar si los clasificadores mejoran su capacidad de predicción. Para ello aplicaremos la técnica *oversampling* utilizando el algoritmo `SMOTE` explicado anteriormente. El conjunto de datos inicial se compone de 50.000 muestras escogidas aleatoriamente utilizando la función `sample` [18]. A partir de él se aplicará el preprocesamiento que mejores resultados hemos obtenido y al final aplicaremos *oversampling* para balancear las clases.

```
# Escogemos 50.000 muestras aleatorias del conjunto inicial
train_m4 <- train[sample(nrow(train), 50000, replace = FALSE, prob = NULL),]
# Aplicamos el mejor preprocesamiento hasta el momento.
## Elinamos columnas con + 50% NAs y con sd=0
train_m4<-eliminar_columnas_nas(train_m4)
train_m4<-eliminar_columnas_sd(train_m4)
```

En este punto disponemos de un *dataset* con **50.000 registros y 301 variables** que pasamos a imputar para predecir los valores utilizando, para ello, un primer método *cart* y luego la media para los NAs restantes. Luego almacenaremos el conjunto de datos resultante en un fichero para que en posteriores usos esté disponible sin volver a realizar este proceso tan costoso.

```
## Imputamos los valores perdidos con "cart" y los restantes con la media.
train_m4_cart<-imputar_nas(train_m4, 1, 1, "cart")
train_m4_mean<-replace(train_m4_cart, TRUE, lapply(train_m4_cart, asignar_media))
## Almacenamos el dataset resultante en un fichero porque este proceso es muy costoso
write.csv(train_m4_mean,'./ficheros/train_m4_mean.csv', row.names = FALSE, col.names=TRUE)
```

A continuación eliminamos las variables muy correladas y las columnas con bajo coeficiente de correlación con respecto a `isFraud`. Al realizar este último paso he podido comprobar que el *dataset* resultante solo contaba con 22 columnas. Me han parecido muy pocas y por ello he realizado un estudio acerca del impacto que tiene este preprocesamiento en los modelos predictivos. Este se muestra más adelante. Y finalmente aplicamos la fórmula para realizar un tratamiento sobre los *outliers*.

```
## Cargamos el dataset desde el fichero
train_m4_mean<-read.csv(file="./ficheros/train_m4_mean.csv", header=TRUE, sep=",")

## Correlación entre variables
train_m4_corr<-correlacion_variables(train_m4_mean, 0.75)
cat("Dimensión del conjunto tras la correlación entre variables\n")
```

```
## Dimensión del conjunto tras la correlación entre variables
```

```
dim(train_m4_corr)
```

```
## [1] 50000   105
```

```r
## Correlación entre variables ~ isFraud
tabla_corr<-correlacion_fraude(train_m4_corr)
## Obtenemos las variables con correlación entre -0.03 y 0.03
tabla_corr_baja<-tabla_corr %>% filter(isFraud > -0.03 & isFraud < 0.03)
## Las eliminamos
train_m4_corr<-
  train_m4_corr[, -which(colnames(train_m4_corr) %in% tabla_corr_baja$Variable)]
cat("\nDimensión del conjunto tras la correlación entre variables~isFraud\n")
```

```
##
## Dimensión del conjunto tras la correlación entre variables~isFraud
```

```r
dim(train_m4_corr)
```

```
## [1] 50000    57
```

```r
## Tratamiento de outliers con la fórmula
train_m4_out<-outliers_formula(train_m4_corr)
```

Una vez disponemos del conjunto de datos preprocesado, vamos a balancear la clase `isFraud`. En el subconjunto que hemos escogido existen menos de 4.000 transacciones fraudulentas por lo que es necesario ampliar esta clase. Para ello aplicamos la función considerada anteriormente y tras varios experimentos ajustando el número de muestras para cada clase, la configuración que consigue un mejor balanceado obtiene **28.072 muestras negativas y 27.993 muestras positivas**. De este modo tenemos un *dataset* con un número de registros muy similar al conjunto de partida inicial.

```r
cat("Nº de transacciones NO fraudulentas\n")
```

```
## Nº de transacciones NO fraudulentas
```

```r
dim(train_m4_out[train_m4_out$isFraud==0,])
```

```
## [1] 46001    57
```

```r
cat("\nNº de transacciones SÍ fraudulentas\n")
```

```
##
## Nº de transacciones SÍ fraudulentas
```

```r
dim(train_m4_out[train_m4_out$isFraud==1,])
```

```
## [1] 3999   57
```

```r
train_m4_smote<-over_sampling(train_m4_out, 600, 117, 5)
cat("\nTras balancear nº de transacciones NO fraudulentas\n")
```

```
##
## Tras balancear nº de transacciones NO fraudulentas
```

```r
dim(train_m4_smote[train_m4_smote$isFraud==0,])
```

```
## [1] 28072    57
```

```r
cat("\nTras balancear nº de transacciones SÍ fraudulentas\n")
```

```
##
## Tras balancear nº de transacciones SÍ fraudulentas
```

```r
dim(train_m4_smote[train_m4_smote$isFraud==1,])
```

```
## [1] 27993    57
```

Cuando ya disponemos de nuestro conjunto de datos final para esta cuarta sección, dividimos el conjunto en entrenamiento y validación y procedemos a entrenar los modelos. Como anteriormente he comentado he realizado un **estudio** acerca de la influencia de quitar más o menos variables en función de la correlación con `isFraud`. Los resultados se pueden visualizar en las dos siguientes tablas para los modelos entrenados con Naive Bayes y Random Forest, respectivamente.

Naive Bayes	22	37	57	92	105
Accuracy	78.39%	81.35%	83.93%	83.36%	83.83%
Kappa	56.78%	62.7%	67.86%	66.73%	67.65%

Random Forest	22	37	57	92	105
Accuracy	94.02%	95.28%	95.55%	95.71%	95.74%
Kappa	88.04%	90.57%	91.1%	91.41%	91.47%

Tal y como se puede observar la tendencia general consiste en que cuantas más variables se eliminan, peor es el resultado. Sin embargo mayor tiempo se invierte en entrenar el clasificador, especialmente con Random Forest. Por lo tanto, de entre todos los experimentos el que mejor relación calidad~tiempo muestra es aquel que dispone de **57 variables**, lo que se transforma en eliminar aquellas columnas con un **coeficiente de correlación entre -0.03 y 0.03** con respecto a `isFraud`.

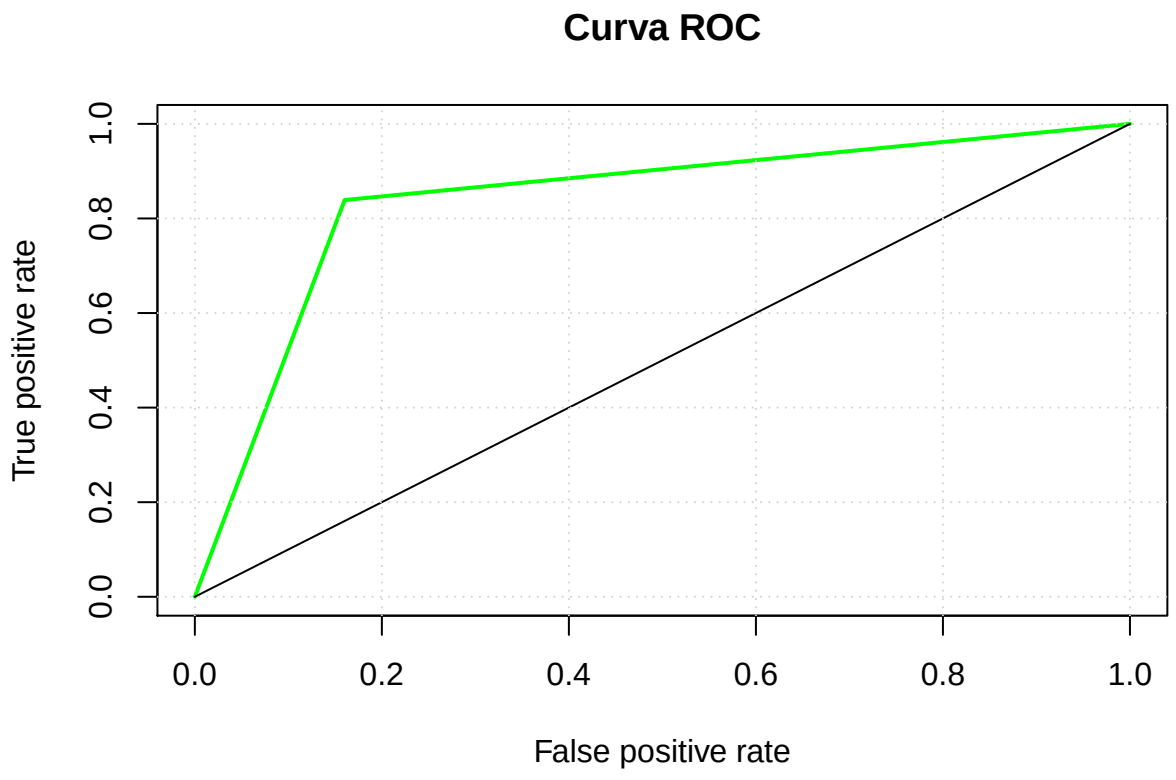
```r
# Transformamos la variable a predecir en factor
train_m4_smote$isFraud<-as.factor(train_m4_smote$isFraud)
# Dividimos el conjunto final en 70% entrenamiento y 30% test
datos<-get_train_test(train_m4_smote, 0.7)

# MODELO NAIVE BAYES
m4_nb<-entrenar_modelo('NB', datos$train, datos$test, '1')
m4_nb$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 7072 1349
```

```
##    1 1352 7045
##
##                Accuracy : 0.8394
##                  95% CI : (0.8338, 0.8449)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6788
##
##  Mcnemar's Test P-Value : 0.9693
##
##             Sensitivity : 0.8393
##             Specificity : 0.8395
##          Pos Pred Value : 0.8390
##          Neg Pred Value : 0.8398
##              Prevalence : 0.4991
##          Detection Rate : 0.4189
##    Detection Prevalence : 0.4993
##       Balanced Accuracy : 0.8394
##
##        'Positive' Class : 1
##
```

```
curva_roc(m4_nb$calidad$preds, datos$test$isFraud)
```




```
##
```

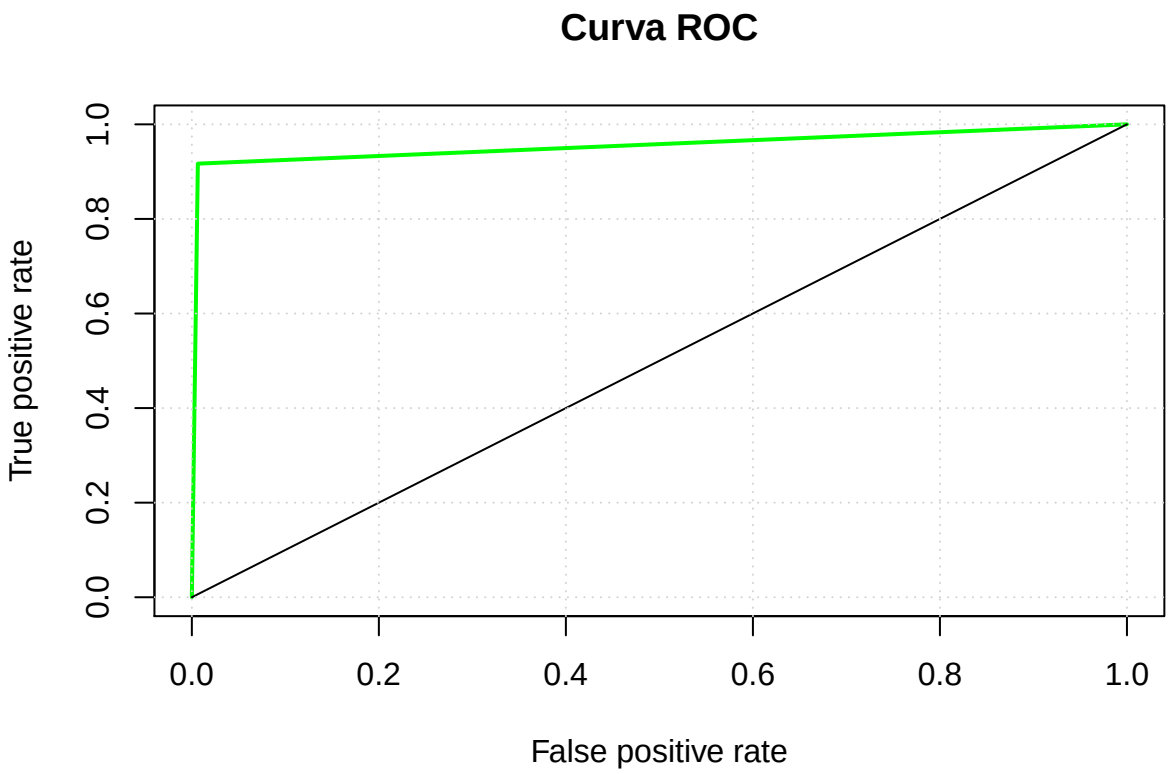
```
## El área bajo la curva ROC es 83.93977 %
```

Como podemos observar, en el clasificador entrenado con **Naive Bayes** los resultados son considerablemente mejores con respecto a los otros preprocesamientos tanto a nivel de precisión como en el coeficiente de *Kappa*. Este hecho también se refleja en su curva ROC cuya área es más amplia que en los anteriores clasificadores. Por lo que podemos concluir que para esta técnica y para nuestro problema, disponer de una **mayor cantidad de datos es beneficioso** para poder mejorar la capacidad de predicción.

```
# MODELO RANDOM FOREST
m4_rf<-entrenar_modelo('RF', datos$train, datos$test, '1')
m4_rf$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 8367   54
##    1  698 7699
##
##                Accuracy : 0.9553
##                  95% CI : (0.9521, 0.9584)
##     No Information Rate : 0.539
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9106
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9930
##             Specificity : 0.9230
##          Pos Pred Value : 0.9169
##          Neg Pred Value : 0.9936
##              Prevalence : 0.4610
##          Detection Rate : 0.4578
##    Detection Prevalence : 0.4993
##       Balanced Accuracy : 0.9580
##
##        'Positive' Class : 1
##
```

```
curva_roc(m4_rf$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 95.52313 %
```

Del mismo modo sucede con el clasificador entrenado mediante **Random Forest** en el que especialmente se nota la mejoría tanto en la precisión pero sobre todo en el coeficiente de *Kappa* que supera por diez puntos ese 80% recomendado para determinar que **es un clasificador competente**. Asimismo, este hecho acompaña la drástica reducción de falsos positivos, por lo que para este clasificador en concreto el hecho de disponer de un mayor número de datos ha sido muy ventajoso para detectar cuándo las transacciones no son fraudulentas. Esta mejoría también se refleja en la **curva ROC** cuya área y amplitud es la mayor hasta ahora conseguida con casi un **95.5%**, lo cual es un gran resultado considerando la calidad de los clasificadores anteriores.

Conclusiones

En este apartado exponemos las conclusiones a las que he llegado de este cuarto preprocesamiento en comparación con los anteriores.

- En primer lugar cabe destacar que si bien un **atributo no está muy correlacionado con la variable a predecir no significa que no aporte información útil** que ayude a clasificar, en nuestro caso una transacción. Por lo tanto, se debe tener cuidado a la hora de eliminar columnas en función del coeficiente de correlación.
- Asimismo se puede intuir que **a mayor número de datos** tanto de entrenamiento como de validación, **más confiable es el clasificador**. Así se ha reflejado en el coeficiente de *Kappa*, que especialmente en el caso del modelo entrenado con Random Forest ha proporcionado un valor superior al óptimo

80%. De este modo podemos asegurar que nuestro clasificador realmente no está etiquetando las muestras al azar, sino que está utilizando lo que ha aprendido aprendido durante el entrenamiento. Lo cual demuestra que esta técnica es súmamente eficaz cuando se dispone de un conjunto de datos suficientemente amplio.

Modelos del quinto preprocesamiento

Como se ha podido comprobar la combinación de las técnicas de preprocesamiento que mejores resultados han obtenido junto con la generación de más datos ha provocado una enorme mejoría en los modelos. Sin embargo, como hemos podido observar elegir las columnas en función de su correlación puede provocar la pérdida de información útil. Por ello, esta sección vamos a realizar un estudio acerca de la **importancia de las variables** para seleccionar las candidatas con las que entrenar. Si bien existen muchas funciones para aplicar esta técnica, he optado por `boruta` [19], un algoritmo que se basa en **Random Forest** para establecer un ranking con la relevancia de las columnas con respecto a la variable a predecir. Se puede especificar tanto el nivel de confianza con el que destaca una variable como importante o no, así como el número de ejecuciones que realiza. Cuanto mayor es este último parámetro, mejor predice la importancia de la columna pero es más costoso computacionalmente.

La idea de aplicar esta función reside en partir de un subconjunto de datos con 50.000 muestras extraidas aleatoriamente del conjunto original. A continuación preprocesamos los datos eliminando columnas con más de un 50% de NAs, con desviación estándar igual a 0 e imputando los valores perdidos. Este conjunto es el mismo que el que se encuentra en el fichero `train_m4_mean.csv` y por ello vamos a reutilizarlo. A continuación eliminamos algunas variables cuya **correlación es mayor que 0.85**, un valor más alto que el que utilizamos anteriormente (0.75) puesto que en este conjunto se eliminan demasiadas variables resultando en solo 105. Por ello he decidido aumentar el máximo coeficiente de correlación para poder estudiar un mayor número de variables, en concreto son 145.

Realizando varias pruebas para aplicar la función he podido comprobar que el **mínimo de ejecuciones necesarias son 11**, y si bien en la documentación se indica que cuantas más mejor, como he comentado anteriormente esto repercute en el tiempo que invierte y más aún si disponemos de un *dataset* tan amplio tanto en registros (50.000) como en columnas (145). Por tanto, para que no se demorase mucho he decidido establecer ese mínimo de ejecuciones para experimentar los resultados con esta técnica. Una vez los he obtenido adjunto una captura de pantalla mostrando las **diez variables más relevantes** para solo ejecutar esta función una sola vez.

```
train_m5<-read.csv(file="./ficheros/train_m4_mean.csv", header=TRUE, sep=",")
# Correlación entre variables -> 145 variables
train_m5<-correlacion_variables(train_m5, 0.85)
# Importancia de las variables
imp<-importancia_variables(train_m5, 11)
# Obtenemos las 60 variables más importantes
tabla_60<-head(imp, 60)
# Formamos un dataset con ellas
train_m5_imp<-train_m5[, which(colnames(train_m5) %in% rownames(tabla_60))]
# Añadimos la variable a predecir
train_m5_imp<-cbind(train_m5_imp, isFraud=as.factor(train_m5$isFraud))
# Guardamos el nuevo dataset en un fichero para disponer de él más adelante
write.csv(train_m5_imp,'./ficheros/train_m5_imp.csv', row.names = FALSE, col.names=TRUE)
```

Tal y como se puede observar las dos más relevantes para explicar la variable a predecir son `TransactionAmt` y `D8`, seguidas de cerca por `id_02`. El resto de columnas parece tener una relevancia similar puesto que sus valores son parecidos. Seguramente con un mayor número de iteraciones el algoritmo podría haber precisado aún más, pero para ello es necesario disponer de un ordenador más potente para que no consuma tantísimo tiempo.

	meanImp <dbl>	decision <fctr>
TransactionAmt	44.52685	Confirmed
D8	40.13327	Confirmed
id_02	33.21351	Confirmed
TransactionID	27.85599	Confirmed
C7	25.90392	Confirmed
V200	25.05175	Confirmed
V257	24.69439	Confirmed
V187	24.66534	Confirmed
id_09	24.60577	Confirmed
V123	23.11484	Confirmed

1-10 of 10 rows

Figure 1: Importancia de las 10 primeras variables.

Tras varias pruebas entrenando los modelos con diferentes números de variables relevantes, el que mejor relación calidad~tiempo he obtenido es un *dataset* con sesenta columnas, por lo que a continuación se muestran los resultados escogiendo las **60 variables más relevantes**. Previo al entrenamiento de los modelos se tratarán los *outliers* con la fórmula de los cuantiles y posteriormente se generarán más muestras con *oversampling* para luego entrenar los clasificadores. De este modo podremos conocer el impacto que supone escoger las columnas que van a predecir `isFraud` utilizando esta nueva técnica.

```
train_m5_imp<-read.csv(file="./ficheros/train_m5_imp.csv", header=TRUE, sep=",")
# Outliers con la fórmula
train_m5_out<-outliers_formula(train_m5_imp)

# Oversampling para balancear la clase isFraud
train_m5_smote<-over_sampling(train_m5_out, 600, 117, 5)
cat("Balanceado: transacciones no fraudulentas\n")
```

```
## Balanceado: transacciones no fraudulentas
```

```
dim(train_m5_smote[train_m5_smote$isFraud==0,])
```

```
## [1] 28072    61
```

```
cat("\nBalanceado: transacciones fraudulentas\n")
```

```
##
## Balanceado: transacciones fraudulentas
```

```
dim(train_m5_smote[train_m5_smote$isFraud==1,])
```

```
## [1] 27993    61
```

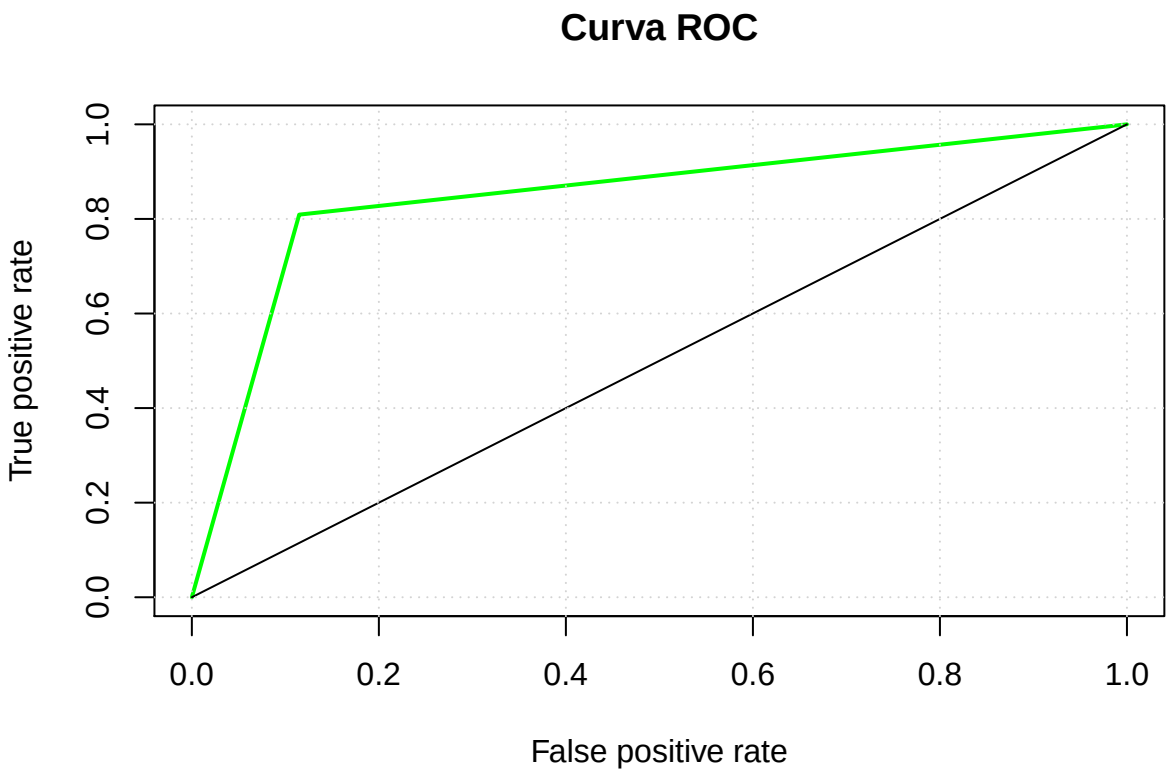
Como podemos observar a continuación, la selección de variables calculando su importancia con la función `boruta` ha sido beneficioso para el clasificador entrenado con **Naive Bayes**, puesto que tanto su precisión como su coeficiente de *Kappa* aumentan ligeramente. Esto nos indica que las características seleccionadas a través de esta técnica explican mejor si una transacción es o no fraudulenta. Este hecho acompaña a un ligero aumento en el área bajo la curva ROC pues se sitúa dos puntos por encima que la del anterior modelo. Asimismo, podemos notar que disminuye considerablemente el número de falsos positivos con respecto al modelo anterior.

```r
# Transformamos la variable a predecir en factor
train_m5_smote$isFraud<-as.factor(train_m5_smote$isFraud)
# Dividimos el conjunto final en 70% entrenamiento y 30% test
datos<-get_train_test(train_m5_smote, 0.7)

# MODELO NAIVE BAYES
m5_nb<-entrenar_modelo('NB', datos$train, datos$test, '1')
m5_nb$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 7454  967
##    1 1603 6794
##
##                Accuracy : 0.8472
##                  95% CI : (0.8417, 0.8526)
##     No Information Rate : 0.5385
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6943
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8754
##             Specificity : 0.8230
##          Pos Pred Value : 0.8091
##          Neg Pred Value : 0.8852
##              Prevalence : 0.4615
##          Detection Rate : 0.4040
##    Detection Prevalence : 0.4993
##       Balanced Accuracy : 0.8492
##
##        'Positive' Class : 1
##
```

```r
curva_roc(m5_nb$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 84.71333 %
```

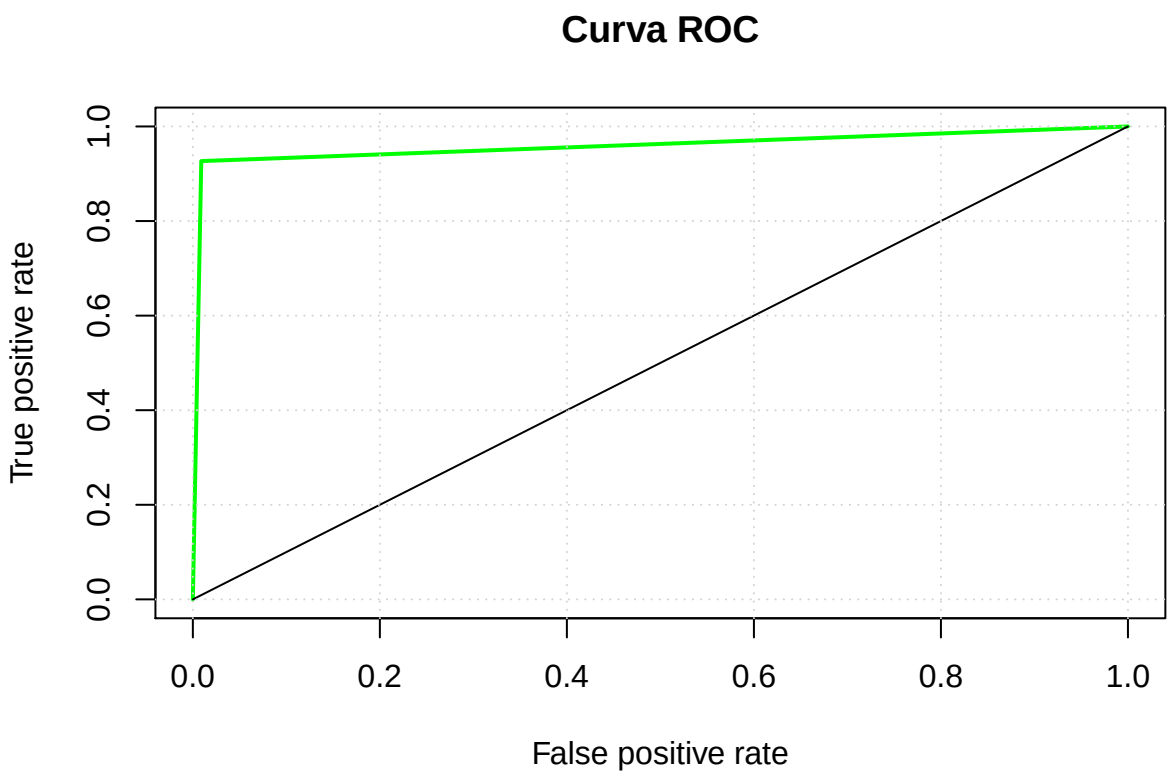
En cuanto al modelo entrenado con **Random Forest** también podemos observar una ligera mejoría tanto en el valor de *accuracy* como en el de *Kappa*. Esto también nos indica que la selección de variables en función de su importancia ha sido beneficiosa para aumentar su capacidad de predicción. Al contrario que ocurría con el modelo anterior, en este ha disminuido el número de falsos negativos por lo que las variables elegidas le ayudan a conocer los indicios que apuntan a que una transacción es fraudulenta.

```
# MODELO RANDOM FOREST
m4_rf<-entrenar_modelo('RF', datos$train, datos$test, '1')
m4_rf$info
```

```
## Confusion Matrix and Statistics
##
##     Predicha
## Real    0    1
##    0 8345   76
##    1  615 7782
##
##                Accuracy : 0.9589
##                  95% CI : (0.9558, 0.9619)
##     No Information Rate : 0.5328
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 0.9178
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9903
##             Specificity : 0.9314
##          Pos Pred Value : 0.9268
##          Neg Pred Value : 0.9910
##              Prevalence : 0.4672
##          Detection Rate : 0.4627
##    Detection Prevalence : 0.4993
##       Balanced Accuracy : 0.9608
##
##        'Positive' Class : 1
##
```

```
curva_roc(m4_rf$calidad$preds, datos$test$isFraud)
```




```
##
## El área bajo la curva ROC es 95.88673 %
```

Conclusiones

En esta subsección destacamos como conclusión la influencia de la selección de características en los modelos. Como hemos podido observar existen multitud de funciones que calculan la importancia de las columnas

utilizando una serie de técnicas. La que he aplicado se basa en el algoritmo *Random Forest* y pese a ser bastante costosa computacionalmente, ha resultado beneficiosa puesto que en ambos modelos se ha notado cierta mejoría con respecto a los anteriores, en los que escogíamos las variables en función de su correlación. Si bien se ha comentado anteriormente, es necesario ser cuidadoso con el tratamiento de la correlación puesto que el que una variable no esté correlada con la columna a predecir, no significa que no disponga de información útil que ayude a clasificar transacciones desconocidas. Por lo tanto, para nuestro problema es **preferible calcular la importancia de las variables** para luego entrenar los modelos con las más relevantes.

Conclusiones globales

Para finalizar la memoria vamos a exponer las conclusiones generales extraidas acerca de las cinco combinaciones de técnicas de preprocesamiento realizadas. En primer lugar destacamos que para este problema **no es recomendable eliminar los valores perdidos** puesto que se reduce drásticamente el conjunto de datos. Esta consecuencia da lugar a pobres clasificadores puesto que no hay suficientes muestras para entrenarlos y validarlos, y por ende, el coeficiente de *Kappa* es muy bajo. Por lo tanto, pese a que sea una técnica más costosa es mejor **imputar los NAs**, aunque si bien hemos comprobado con la función `mice` esta tarea también es difícil de llevar a cabo puesto que hay columnas con poca información como para predecirlos. Además debemos tener en cuenta los recursos limitados tanto por el tiempo como por el portátil, por lo que quizás con una mayor capacidad computacional se podrían haber efectuado imputaciones más profundas. Sin embargo, también podemos optar por técnicas más sencillas, como calcular la media, y de esa forma predecir los NAs restantes.

Además de la gran cantidad de valores perdidos del *dataset*, otro de sus inconvenientes son las columnas con **desviación igual a 0**. Más allá de disponer de columnas sin variación en sus valores, esta característica ha provocado en bastantes ocasiones errores al aplicar otras técnicas como la correlación o el PCA. Por ello es importante controlar la aparición de columnas sin información cada vez que se modifiquen sus valores, es decir, tanto en la imputación de los valores NAs como en el tratamiento de *outliers*. En relación a este último hemos comprobado que **no siempre las técnicas más sofisticadas son las que mejores resultados proporcionan**, por lo que para este *dataset* es mejor tratar este tipo de valores con la fórmula de los cuantiles.

La anterior conclusión también es aplicable al **PCA, el cual no siempre proporciona unas columnas mejores que las originales** puesto que al combinarlas se pierde información valiosa que puede resultar más útil. Este es el caso de nuestro *dataset* tal y como hemos podido comprobar puesto que los resultados al reducir la dimensionalidad del conjunto han sido peores cuando se ha aplicado el PCA. De igual modo debemos ser cuidadosos a la hora de eliminar columnas en función de la **correlación** ya que podemos descartar variables que si bien en principio no parecen relacionadas con la variable a predecir, sí que aportan información útil para entrenar los clasificadores. Es por ello por lo que podemos aplicar métodos más sofisticados que son capaces de calcular la **importancia de las variables** para escoger aquellas más relevantes. Si bien existen muchas opciones, la que he probado en esta práctica ha conseguido obtener los clasificadores que proporcionan los mejores resultados. De este modo la selección de características se encuentra fundamentada en técnicas eficientes y estables, aunque para nuestro *dataset* pueden ser bastante costosas de aplicar.

Por último destacamos la enorme mejora de los modelos al aplicar **oversampling** en lugar de su contrario *undersampling*. Esto indica algo que ya podíamos intuir, y es que a mayor número de datos mejor entrenamiento y validación de los modelos. En particular, en nuestro problema ha sido sumamente beneficioso puesto que a través de la generación de un mayor número de muestras y del balanceo de la clase `isFraud`, se han conseguido modelos con un 90% de precisión y coeficiente de *Kappa*. Esto nos indica que no etiqueta las muestras al azar sino que está aplicando lo que ha aprendido durante el entrenamiento. Si bien es cierto que también en parte se debe a la eficacia y robustez de **Random Forest**, ya que aunque también han mejorado los modelos entrenados con Naive Bayes esta técnica no es capaz clasificadores de tal calidad para nuestro problema.

Bibliografía

[1] Kaggle, IEEE-CIS Fraud Detection, https://www.kaggle.com/c/ieee-fraud-detection/data

[2] RDocumentation, df_status, https://www.rdocumentation.org/packages/funModeling/versions/1.9.3/topics/df_status

[3] Kaggle, Data Description (Details and Discussion), https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203

[4] Documentación sobre la función nearZeroVar, https://www.rdocumentation.org/packages/caret/versions/6.0-85/topics/nearZeroVar

[5] Documentación sobre la función mice, https://www.rdocumentation.org/packages/mice/versions/2.25/topics/mice

[6] Documentación acerca de la función findCorrelation, https://rdrr.io/rforge/caret/man/findCorrelation.html

[7] r-statistics.co, Selva Prabhakaran, Outlier Treatment, http://r-statistics.co/Outlier-Treatment-With-R.html

[8] Documentación sobre la función boxplot, https://www.rdocumentation.org/packages/grDevices/versions/3.6.2/topics/boxplot.stats

[9] Documentación sobre la función preProcess, https://www.rdocumentation.org/packages/caret/versions/6.0-86/topics/preProcess

[10] Machine Learning Plus, Selva Prabhakaran, Feature Selection – Ten Effective Techniques with Examples, 2018, https://www.machinelearningplus.com/machine-learning/feature-selection/

[11] Documentación acerca de la función downSample, https://www.rdocumentation.org/packages/caret/versions/6.0-85/topics/downSample

[12] Documentación sobre la función SMOTE, https://www.rdocumentation.org/packages/DMwR/versions/0.4.1/topics/SMOTE

[13] Documentación acerca de la función createDataPartition, https://www.rdocumentation.org/packages/caret/versions/6.0-85/topics/createDataPartition

[14] Documentación de la función confusionMatrix, https://www.rdocumentation.org/packages/caret/versions/3.45/topics/confusionMatrix

[15] DEGREES OF BELIEF, COMMON EVALUATION MEASURES FOR CLASSIFICATION MODELS, https://degreesofbelief.roryquinn.com/common-evaluation-measures-for-classification-models

[16] DATANOVIA, INTER-RATER RELIABILITY MEASURES IN R: Cohen's Kappa in R: For Two Categorical Variables , https://www.datanovia.com/en/lessons/cohens-kappa-in-r-for-two-categorical-variables/#interpretation-magnitude-of-the-agreement

[17] StackExchange, How do the number of imputations & the maximum iterations affect accuracy in multiple imputation? , https://stats.stackexchange.com/questions/219013/how-do-the-number-of-imputations-the-maximum-iterations-affect-accuracy-in-mul/219049

[18] Documentación sobre la función sample, https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/sample