Universidad de Granada

Sistemas Inteligentes para la Gestión en la Empresa (SIGE)

Departamento de Ciencias de la Computación e Inteligencia Artificial
E.T.S.I. Informática y Telecomunicación
Máster en Ingeniería Informática
Universidad de Granada
Curso 2018-2019

Práctica 1

Pre-procesamiento de Datos y Clasificación Binaria

Luis Gallego Quero 77375026-J

lgaq94@correo.ugr.es

23 de abril de 2019

Índice

Índice de figuras

1. Introducción

En la presente práctica nos enfrentamos al estudio y uso de diferentes técnicas tanto de preprocesado de datos como métodos de aprendizaje automático. Para ello haremos uso de un dataset con diversas variables con un claro fin, el de identificar que clientes realizarán una transacción en el futuro, independientemente de la cantidad de dinero que se use en esta. Para ello aplicaremos técnicas de aprendizaje automático que nos permitirán predecir si esos clientes realizarán la transacción en el futuro o no.

Para ello dividiremos las tareas en tres módulos principales. El primero, algo más reducido, se encargará de la lectura y exploración inicial de los datos. Este paso nos permitirá obtener una primera impresión sobre el dataset al que nos enfrentamos. En el segundo módulo realizaremos todas las tareas destinadas al preprocesamiento, con esto conseguiremos obtener un dataset lo más reducido y limpio posible con el que trabajar en el aprendizaje automático. Finalmente este se desarrollará en el tercer módulo y nos permitirá construir un modelo con el que realizar las distintas predicciones sobre el modelo creado.

A lo largo de la siguiente memoria se expondrán las diferentes técnicas usadas, el proceso experimental llevado a cabo hasta llegar a la mejor solución obtenida, a la vez que se discuten los diferentes detalles de optimización usados en cada caso.

2. Exploración

Como ya hemos comentado en la introducción, en este primer paso vamos a realizar la lectura y exploración de los datos. Esto nos permitirá rápidamente darnos cuenta de que no nos enfrentamos a un dataset cualquiera, ya que presenta diversas peculiaridades que comentaremos más adelante.

2.1 Lectura y exploración de los datos

Comenzamos leyendo los datos. Destacar que "test.csv" está descargado de la competición de kaggle, ya que como se comentará en el momento de la partición del dataset en el apartado último del preprocesamiento, en mi caso he partido train justo al finalizar este, lo que dió lugar a diversas cuestiones que se comentarán allí. Por tanto, para poder tener una valoración más sobre la bondad de nuestro modelo, se ha hecho uso de Kaggle.

```
# Lectura de datos
data_raw <- read_csv('./train_ok.csv')
test_raw <- read_csv('./test.csv')

# Eliminamos la variable ID de train
data_raw <- data_raw[,-1]
# Guardamos la variable ID de test y la
test <- test_raw
indices_test <- test$ID_code
# Eliminamos la variable ID de test
test <- test[,-1]

# Analizamos
status_train <- df_status(data_raw)
status_test <- df_status(test_raw)
```

variable <chr>	q_zeros <int>	p_zeros <dbl>	q_na <int>	p_na <dbl>	q_inf <int>	p_inf <dbl>	type <fctr>	unique <int>
target	181989	90.99	0	0.00	0	0	numeric	2
var_0	0	0.00	17	0.01	0	0	numeric	94670
var_1	1	0.00	11	0.01	0	0	numeric	108931
var_2	0	0.00	19	0.01	0	0	numeric	86554
var_3	0	0.00	16	0.01	0	0	numeric	74593
var_4	0	0.00	22	0.01	0	0	numeric	63513
var_5	3	0.00	21	0.01	0	0	numeric	141017
var_6	0	0.00	21	0.01	0	0	numeric	38599
var_7	0	0.00	20	0.01	0	0	numeric	103059
var_8	1	0.00	17	0.01	0	0	numeric	98615

Figura 1: Resultado de df_status para data_raw

Según esta primera valoración comentar que nos encontramos ante un problema complejo con 200.000 observaciones y 202 variables diferentes, además todas estás variables son valores numéricos excepto el ID identificativo que son carácteres. Como primera toma de contacto el resultado no es muy positivo ya que haciendo uso de df_status podemos ver que por lo general las distintas variables no superar el 0.01 % de NAs, el porcentaje de ceros apenas sobrepasa en algunas ocasiones al 0.1 e igual para el infinito, por lo que la primera parte de preprocesamiento en la que buscaremos eliminar el mayor número de varibles posibles se complica. De todas formas intentaremos reducir el mayor número de variables posibles.

Continuando este análisis previo, como hemos comentado antes, podemos ver que tenemos valores perdidos, no son muchos pero los hay. En un problema estándar podríamos pasar sin arreglarlo, pero no estamos en un problema estándar ya que estamos en un problema de clasificación binario de 0 o 1 en el que el 90 % son 0. Entonces cualquier algoritmo lo que haría sería clasificar para la clase mayoritaria (los 0), y en el caso de la clase minoritaria clasificaría al azar. Esto nos llevará a tener un buen accuracy, pero ese accuracy no es un

buen resultado ya que todas las clasificaciones de la clase mayoritaría van a fallar, o al menos gran parte de ellas.

También es especialmente relevante que el 90.99 % de los valores de la variable objetivo (target) son 0, esto nos lleva a pensar que si construimos nuestro propio predictor básico en el que nunca se realice una transacción por parte del cliente, en un alto porcentaje de las veces acertaremos. Esta situación nos lleva a la anteriormente descrita, conseguiremos un alto porcentaje de acierto, pero en los casos que realmente nos interesa fallaremos en gran medida.

2.2. Equilibrio de clases

Si bien ya hemos comentado algunos problemas que podemos observar en el dataset, el siguiente gráfico nos muestra uno de los problemas mayores de nuestro dataset, el claro desbalanceo entre los ejemplos negativos que tenemos frente a los positivos.

```
# Convertimos a as.factor para ggplot
data_raw_ggplot <- data_raw %>%
  mutate(target = as.factor(target))

# Visualizamos
ggplot(data_raw_ggplot) +
  geom_histogram(aes(x = target, fill = target), stat = 'count') +
  theme_hc()
```

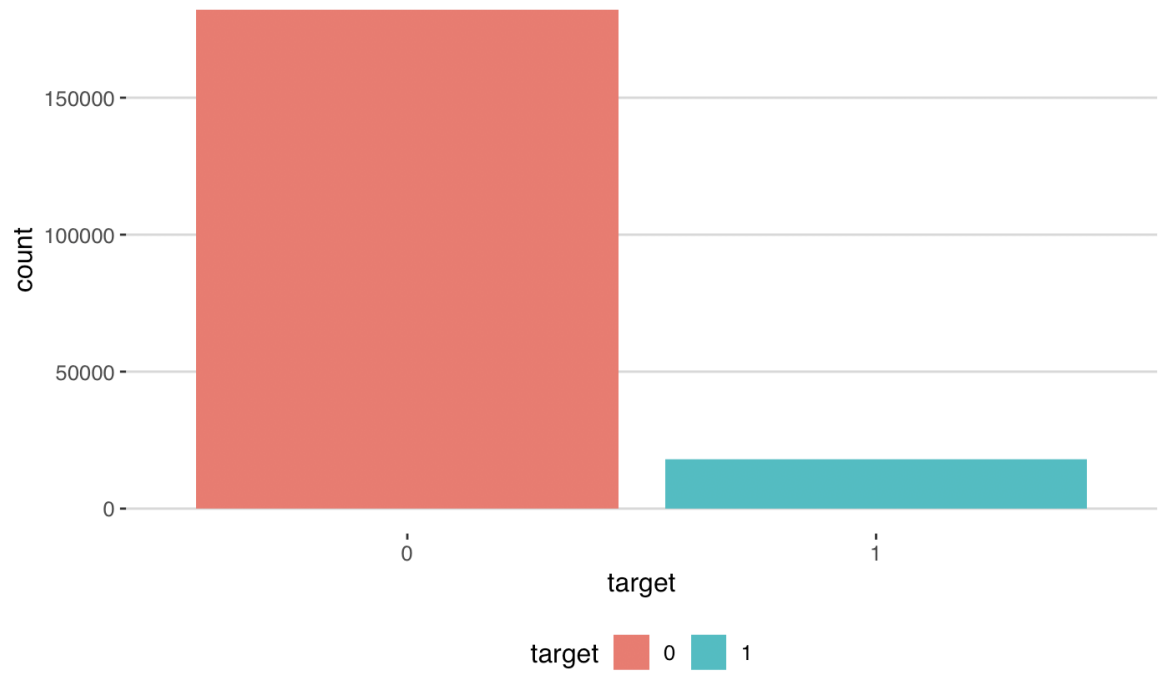

Figura 2: Equilibrio de clases

Es evidente tras el presente gráfico que tenemos un claro deficit de resultados positivos en nuestra variable objetivo, lo que provoca que no esté balanceado el dataset. Este problema tendremos que resolverlo posteriormente si queremos obtener resultados más óptimos, por lo que será atacado en los últimos momentos

del preprocesamiento. Principalmente al resolver el balanceo conseguiremos tener más ejemplos de la clase minoritaria, lo que dará lugar a que el clasificador pueda aprender mejor estos casos.

3. Preprocesado

Comenzamos nuestro segundo gran módulo, preprocesado, en el que vamos a intentar eliminar las variables que nos aportan menos información o que son más redundantes.

3.1. Eliminar columnas no útiles

Primero, haciendo uso de df_status, eliminaremos las variables que cumplan las diversas condiciones impuestas. Estas variables se eliminarán tanto de train como de test. Comentar que además de las condiciones que podemos ver más abajo hubiera sido interesante eliminar columnas con un gran número de ceros o NAs, pero nuestro dataset presenta un número especialmente bajo de estos. Además, realizaremos un primer filtrado un poco agresivo, para así eliminar un número sustancial de variables.

```
train <- data_raw
status <- df_status(data_raw)

# Tenemos que mantener 'target'
# Nos creamos una copia provisional para al borrar
# no eliminar esta variable
status <- status %>%
  filter(variable != 'target')

# Identificar columnas > 75% valores diferentes
dif_cols_mayor <- status %>%
  filter(unique > 0.75 * nrow(data_raw)) %>%
  select(variable)
# Identificar columnas <20% valores diferentes
dif_cols_menor <- status %>%
  filter(unique < 0.20 * nrow(data_raw)) %>%
  select(variable)

# Unificamos las variables que se van a eliminar
remove_cols <- bind_rows(
  list(
    dif_cols_mayor,
    dif_cols_menor
  )
)

# Eliminamos las variables
# Train
train <- train %>%
  select(-one_of(remove_cols$variable))
# Test
test <- test %>%
  select(-one_of(remove_cols$variable))
```

Inicialmente, hemos identificado las columnas con más del 75 % de valores diferentes. Esto nos ha llevado a eliminar 14 variables, aunque si hubieramos llevado el límite tan solo al 70 % de valores diferentes, el número de variables eliminadas se hubiera disparado hasta 40, lo que a nuestro criterio ya nos parece excesivo. Esto es así ya que mediante las columnas con menos del 20 % de variables diferentes también hemos eliminado hasta 31 variables. Además mediante el estudio de correlación también descartaremos un número sustancial de estas.

3.2. Correlación

Nuestro siguiente paso es el estudio de correlación, es este buscaremos localizar las variables que están fuertemente ligadas con la variable objetivo.

3.2.1. Variables con una alta correlacion con la variable objetivo

En este primer caso vamos a comprobar la correlación de nuestras variables respecto de la variable objetivo, target. En este caso nos interesa quedarnos con las variables más correladas, ya que de ellas depende el resultado de target. Por tanto, nos centraremos en eliminar aquellas con un valor bajo de correlación, que como veremos, son gran parte de ellas.

```
# Eliminamos los NAs
train_num <- train %>%
  na.exclude()

# Realizamos la correlación
cor_target <- correlation_table(train_num, target = 'target')

# Seleccionamos las variables que superen un valor mínimo
important_vars <- cor_target %>%
  filter(abs(target) >= 0.025)

# Eliminamos las variables seleccionadas
train <- train %>%
  select(one_of(important_vars$Variable))
test <- test %>%
  select(one_of(important_vars$Variable))
```

En nuestro caso hemos eliminado aquellas variables que tienen un valor de correlación menor al 0.025 con la variable objetivo, lo cual es un valor bastante bajo pero que nos elimina un total de 79 variables. Esto nos indica que en nuestro dataset en gran medida las variables no tienen una relación directa con la variable objetivo.

Esta criba es especialmente relevante ya que si los valores tienen demasiada variabilidad, es como si estuvieramos utilizando valores aleatorios prácticamente para nuestras predicciones, por lo que no se podrá sacar conclusiones de ellos y los resultados no serán positivos. Además, esto dificulta obtener un criterio para dividir, que en el caso de la clasificación mediante árboles se basa en ese concepto principalmente y son los clasificadores que nosotros usaremos.

3.3. Imputación de valores perdidos

Como comentamos al inicio de este trabajo, el número de valores perdidos que hay en nuestro dataset no es muy grande, como podemos ver en el siguiente gráfico. Por tanto, si estuvieramos trabajando en un problema estándar podríamos prescindir de corregir esos valores y no eliminarlos. En este caso es más adecuado obtener un valor para ellos, y para ello haremos uso de la media de los valores de la variable que corresponda al NA. Además esto nos beneficiará a la hora de balancaer nuestro dataset.

```
aggr(train,
     col=c('navyblue','red'),
     numbers=TRUE,
     sortVars=TRUE,
     labels=names(train),
     cex.axis=.7,
```

```
     gap=3,
     ylab=c("Histogram of missing data", "Pattern"))
```

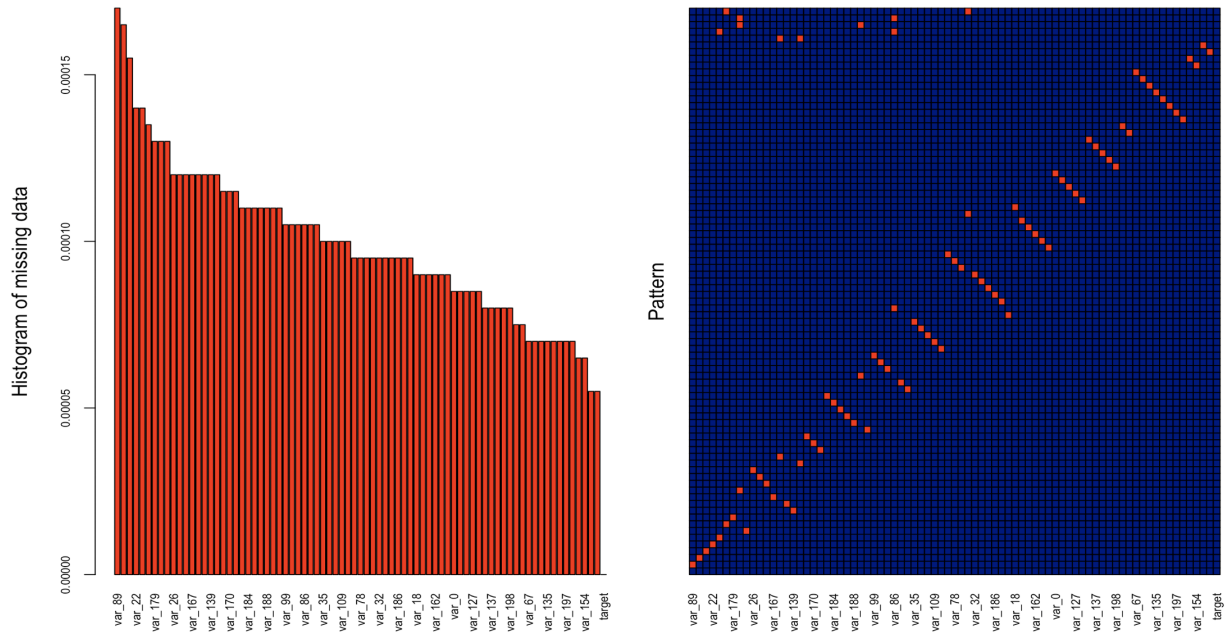

Figura 3: Datos perdidos

Para resolver el problema de los valores perdidos vamos a hacer uso de mice. Concretamente este hace uso de la media por defecto y una serie iteraciones en cada imputación. En mi caso he reducido estas imputaciones e iteraciones a 3, ya que su ejecución se alargaba considerablemente y en diversas pruebas que se realizaron con un número mayor de estas, los beneficios apenas eran notables.

```
# Por defecto tenemos m = 5
# Por defecto tenemos maxit = 5
imputation_train <- mice(train, m = 3, maxit = 3)

# Guardamos las imputaciones
train_imput <- complete(imputation_train)

# Guardamos
write_csv(train_imput, './copia_seguridad/train_imput.csv')
saveRDS(imputation_train, file = "./copia_seguridad/imputation_train.rds")
```

Comprobamos ahora que la imputación ha tenido éxito y los valores perdidos han sido sustituidos.

```
aggr(train_imput,
     col=c('navyblue','red'),
     numbers=TRUE,
     sortVars=TRUE,
     labels=names(train_imput),
     cex.axis=.7,
     gap=3,
     ylab=c("Histogram of missing data train", "Pattern"))
```

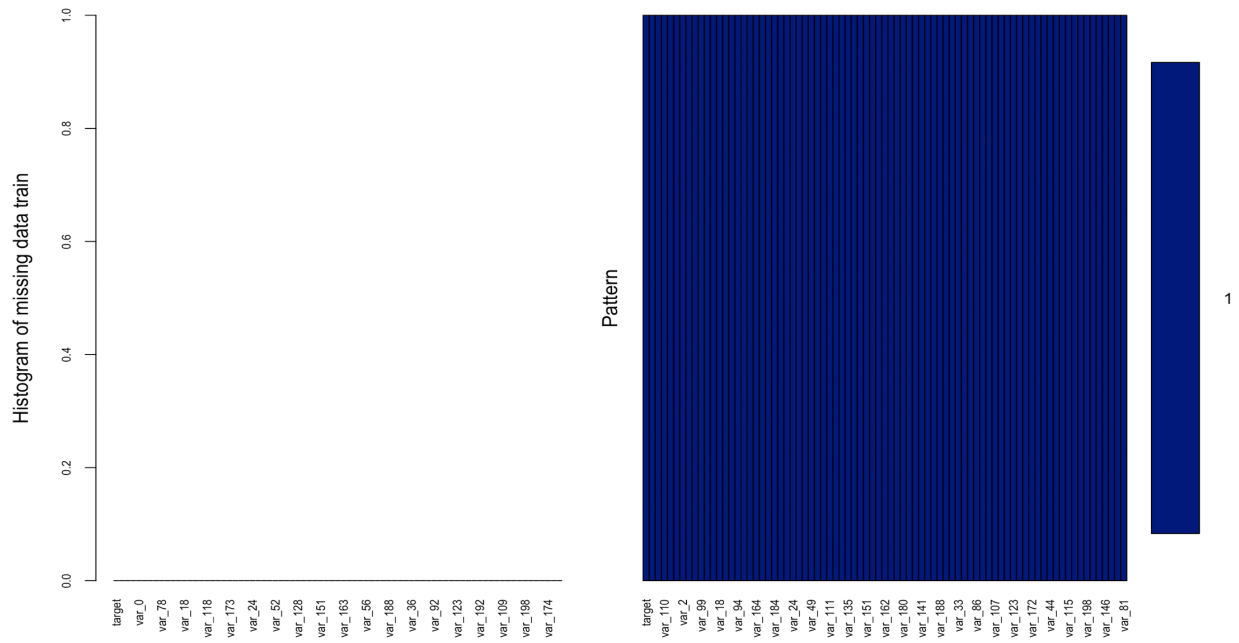

Figura 4: Datos perdidos tras imputación

3.4 Tratamiento de clases no balanceadas

Llegamos a una de las últimas fases del preprocesado, el balanceo. Ya ha sido comentado el claro desnivel que podemos encontrar en los ejemplos de los dos tipos de resultados de nuestro dataset, llegando hasta el 90 % en el caso del 0. Por tanto, en este paso vamos a crear muestras ficticias principalmente de la clase 1 mediante smote.

Para esto se han hecho multitud de pruebas, tanto aumentando la clase minoritaria, como reduciendo la mayoritaria y con diversas combinaciones de valores. Concretamente reduciendo la clase mayoritaria hemos obtenido peores resultados por lo para nuestro modelo final hemos hecho uso del aumento de la clase minoritaria (upsampling).

```
# Ajustamos las variables
train_smote <- train_imput %>%
  mutate_if(is.character, as.factor) %>%
  as.data.frame()

# Smote solo funciona con la variable target como factor
train_smote$target = as.factor(train_smote$target)

# Ejecutamos el balanceo
train_smote <- SMOTE(target ~ ., train_smote, perc.over = 600, dataperc.under = 100)

# Guardamos
write_csv(train_smote, './copia_seguridad/train_smote.csv')

# Mostramos el resultado
ggplot(train_smote) +
  geom_histogram(aes(x = target, fill = target), stat = 'count') +
  theme_hc()
```

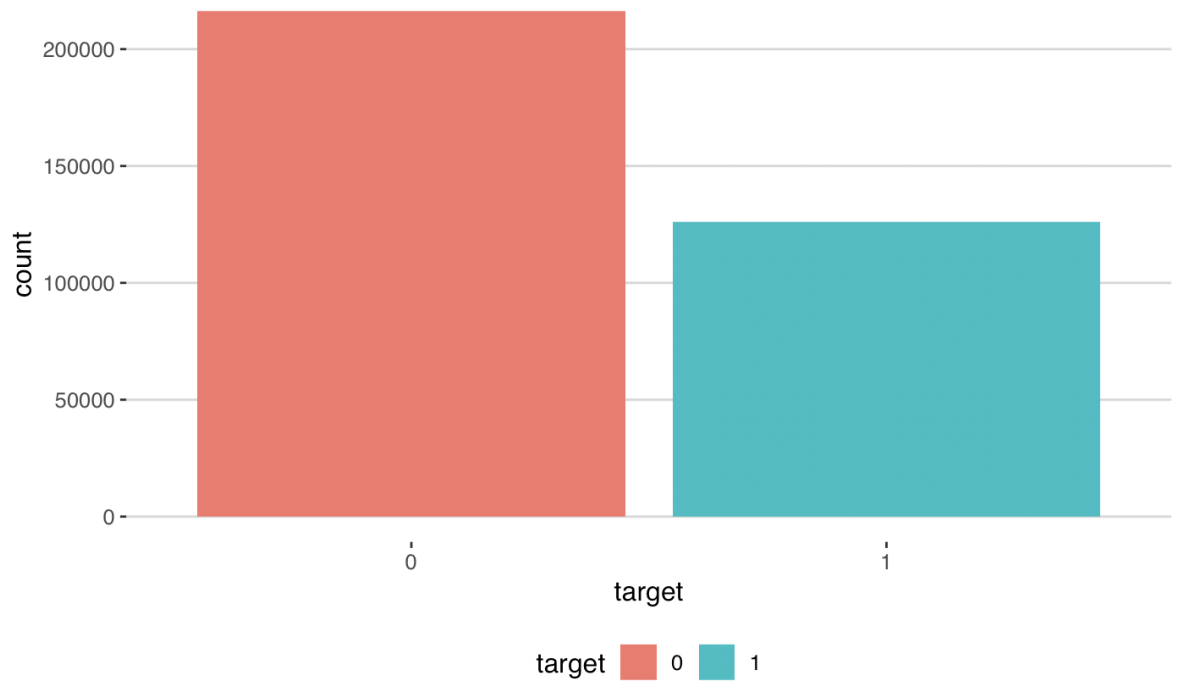

Figura 5: Correción de balanceo

Es evidente el aumento significativo que podemos encontrar en la clase minoritaria, llegando a tener hasta más de 100.000 resultados. Es destacable también que la clase minoritaria también ha aumentado levemente.

3.5. Preprocesado final y partición

Últimos pasos en el proceso de preprocesado y no por ello menos importantes. El primero ha sido cuestión de debate durante gran parte del desarrollo de esta práctica, y trata sobre el proceso de partición. Inicialmente y según las indicaciones dadas en clase, la partición del dataset train en train y validación se empezó realizando justo antes del momento en el que creamos el modelo. El debate era claro, si lo realizamos en ese momento, los datos de validación están "contaminados", ya que se le ha realizado un intensivo preprocesamiento y por tanto es más facil para nuestros clasificadores obtener mejores resultados.

Esto nos llevó a diveros debates con compañeros con más experiencia en este campo, y por tanto la partición fue cambiada al inicio del script para salvar dicha contaminación. Tras las aclaraciones mediante correos al profesor y siguiendo con sus explicaciones iniciales la partición ha vuelto al final del preprocesado. Por tanto, para poder obtener otra valoración que esté sin contaminar, también se ha hecho uso del test que proporciona kaggle.

Una vez realizada dicha partición solo nos queda un paso más. Este es un análisis de componentes principales que haremos mediante la función preProcess, y sobre todo, mediante su método "pca". Este nos proporciona diferentes funcionalidades como pueden ser la reducción de dimensionalidad y correción del ruido en los datos.

```
################ TRAIN ################
train_preprocesado <- train_smote %>%
  mutate(target = as.factor(ifelse(target == 1, 'Yes', 'No'))) %>%
  na.exclude()

# write_csv(train_preprocesado, './copia_seguridad/train_preprocesado.csv')
```

```
############### TEST ###############
test_clasificacion <- test

############### PARTICIÓN TRAIN - VAL ###############
# Partimos dataset en conjuntos de entrenamiento y validación
trainIndex <- createDataPartition(train_preprocesado$target, p = .8, list = FALSE, times = 1)
train_clasificacion <- train_preprocesado[trainIndex, ]
val_clasificacion <- train_preprocesado[-trainIndex, ]

############### PREPROCESS ###############
preprocess <- preProcess(train_preprocesado, method = "pca")
train_clasificacion <- predict(preprocess, train_clasificacion)
val_clasificacion <- predict(preprocess, val_clasificacion)
test_clasificacion <- predict(preprocess, test_clasificacion)

# Comprobación final
head(train_clasificacion, 10)
df_status(train_clasificacion)
```

Una visualización final del estado de los datos la podemos encontrar en las siguientes imágenes. Destacar como el uso de un un análisis de componentes principales nos ha modificado el nombre de las variables y además nos ha reducido el número de estas, dejando nuestro dataset en óptimas condiciones para la creación de modelos.

	target <fctr>	PC1 <dbl>	PC2 <dbl>	PC3 <dbl>	PC4 <dbl>	PC5 <dbl>	PC6 <dbl>	PC7 <dbl>	PC8 <dbl>
1	No	-0.77013222	1.6410129	-2.0170580	-1.3831711	0.0001576843	1.32115193	0.6349557	0.8312095
3	No	0.63494870	0.4232005	-0.5948597	-0.1652685	-1.0610581495	-0.92050168	-0.4795766	-1.1318954
4	No	-1.15253164	0.8658697	-2.0584999	0.2361738	1.7082804854	-0.99176564	-1.1590197	-2.6175780
5	No	-1.30392685	-0.1893418	-0.5440535	0.1071117	-0.4720703229	-1.64853176	1.1880469	2.4965781
6	No	-0.22009111	-1.1835659	-0.6823494	0.4495337	0.8904918457	1.21305540	-0.8499946	-0.2027448
7	No	-0.86884217	0.9754218	-2.4960025	-0.8906827	-1.3163790184	0.06006838	-0.9872964	-0.2375176
8	No	-1.83911461	-1.0833393	0.1764846	0.6243164	-0.9103586881	0.34803366	-3.3379278	0.6750383
9	No	-0.64239364	1.6149126	-1.4262585	-0.4041488	-0.3782121190	1.14074444	1.5638915	1.2042125
11	No	0.04201491	0.7581305	-2.0737697	1.7329275	0.8145917488	0.02446226	1.8307404	1.1404204
12	No	-1.88664464	-1.5824815	0.7576524	2.5580525	0.7298870317	-1.88334108	1.2627853	-0.4385514

Figura 6: Head train final

variable <chr>	q_zeros <int>	p_zeros <dbl>	q_na <int>	p_na <dbl>	q_inf <int>	p_inf <dbl>	type <fctr>	unique <int>
target	0	0	0	0	0	0	factor	2
PC1	0	0	0	0	0	0	numeric	212390
PC2	0	0	0	0	0	0	numeric	212390
PC3	0	0	0	0	0	0	numeric	212390
PC4	0	0	0	0	0	0	numeric	212390
PC5	0	0	0	0	0	0	numeric	212390
PC6	0	0	0	0	0	0	numeric	212390
PC7	0	0	0	0	0	0	numeric	212390
PC8	0	0	0	0	0	0	numeric	212390
PC9	0	0	0	0	0	0	numeric	212390

1–10 of 75 rows Previous 1 2 3 4 5 6 … 8 Next

Figura 7: df_status train final

4. Clasificación y discusión

En el apartado que nos ocupa vamos a presentar los dos modelos usados en el transcurso de nuestro trabajo. Para entender mejor el proceso de refinamiento y pruebas seguido hasta la culminación de los modelos de entrenamiento que presentamos, se expondrá de forma conjunta tanto la explicación del clasificador como la discusión de los resultados obtenidos.

Tal y como veremos en las siguientes líneas el proceso experimental ha sido desarrollado desde el caso más básico, con un simple preprocesado y modelo estándar de predicción, hasta la evolución más completa posible de ambos que hemos conseguido desarrollar.

4.1. Rpart

Inicialmente nuestro preprocesado se basaba en la eliminación de columnas no útiles y el estudio de correlación. Esto suponía un descenso notable en el número de variables, lo cual se aventuraba bastante positivo ya que el dataset no se nos presentaba especialmente ilustrativo para este primer paso de procesamiento. Entonces para realizar una primera toma de contacto construimos el primer modelo de entrenamiento mediante rpart. Para esto hubo que realizar algunos ajustes como eliminar los NAs.

Tal y como esperabamos este primer ensayo no resultó nada positivo, pues nuestro predictor tan solo predecía correctamente los de la clase mayoritaria, los ceros en este caso. Tras esto realizamos varias evaluaciones para comprobar hasta que punto influye el número de variables en el resultado final. Este estudio nos indicaba que reducirlas no nos influía especialmente, pero ya nos encontramos con una situación que nos empezó a resultar algo cuestionable. Y es que el valor de la curva roc haciendo uso de rpart siempre era el mismo. Además estas primeras curvas roc que se estaban generando eran el claro ejemplo de la peor predicción posible.

Como siguiente paso decidimos implementar el músculo del preprocesado, la imputación de valores perdidos y el tratamiento de clases no balanceadas. Estos pasos supusieron un gran tiempo en el desarrollo, ya que principalmente al tratarse de mi primera toma de contacto en el análisis de datos, los ajustes de los parámetros no eran correctos y las ejecuciones se alargaban varias horas.

Una vez controla la parametrización tanto de mice y de rpart (especialmente el grid), se sucedieron diversas pruebas, pero el valor de la curva roc apenas cambiaba, y lo que era más destacable era que nuestro clasificador apenas realizaba predicciones para la clase minoritaria. Esto fue resuelto con el uso se smote y el balanceo de clases, lo que supuso un avance considerable en el desarrollo.

Una vez llegados a este punto pasamos a utilizar principalmente random forest en nuestras pruebas, haciendo uso de rpart tan solo para obtener una valoración más.

```
rpartCtrl <- trainControl(verboseIter = F, classProbs = TRUE, summaryFunction = twoClassSummary)
rpartParametersGrid <- expand.grid(.cp = c(0.001, 0.01, 0.05))

# RPART
rpartModel <- train(target ~ .,
                    data = train_clasificacion,
                    method = "rpart",
                    metric = "ROC",
                    trControl = rpartCtrl,
                    tuneGrid = rpartParametersGrid)

# Guardamos
#saveRDS(rpartModel, file = "./copia_seguridad/rpartModel.rds")
# Primera info del modelo
print(rpartModel)

############### VAL ###############
```

```
prediction_rpart_val <- predict(rpartModel, val_clasificacion, type = "raw")
cm_train_rpart_val <- confusionMatrix(prediction_rpart_val,
                                      val_clasificacion[["target"]],
                                      mode = "everything",
                                      positive = "No")
cm_train_rpart_val
# Mediante probabilidades y curva ROC
predictionValidationProb_rpart_val <- predict(rpartModel, val_clasificacion, type = "prob")
auc_rpart_val <- roc(val_clasificacion$target,
                     predictionValidationProb_rpart_val[["Yes"]],
                     levels = unique(val_clasificacion[["target"]]))
auc_rpart_val

############### TEST ###############
prediction_rpart_test <- predict(rpartModel, test_clasificacion, type = "raw")
predictionValidationProb_rpart_test <- predict(rpartModel, test_clasificacion, type = "prob")

# Visualizamos
roc_validation_rpart_val <- plot.roc(auc_rpart_val,
                                     ylim=c(0,1),
                                     type = "S" ,
                                     print.thres = T,
                                     main=paste('Validation AUC VAL:',
                                                round(auc_rpart_val$auc[[1]], 2)))
```

Antes de pasar al proceso experimental seguido con random forest, comentar los resultados obtenidos mediante rpart en el mejor caso. Como vemos en la siguiente imagen hemos usado un grid de tres valores para el que obtenemos la mejor curva roc en el menor de ellos.

```
CART

273768 samples
    74 predictor
     2 classes: 'No', 'Yes'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 273768, 273768, 273768, 273768, 273768, 273768, ...
Resampling results across tuning parameters:

  cp     ROC        Sens       Spec
  0.001  0.7740466  0.8401876  0.6331348
  0.010  0.7594786  0.8059277  0.6687585
  0.050  0.7463153  0.7637374  0.7288932

ROC was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.001.
```

Figura 8: Modelo obtenido con rpart

La matriz de confusión obtenida con dicho valor de grid, es la siguiente. En esta destacar como el accuracy apenas sobrepasa el 0.75 y el valor kappa no llega al 0.5. Este último valor es especialmente relevante en problemas en los que nuestro dataset está muy desbalanceado, como es el problema que nos ocupa. También es interesante el valor de F1, que en este caso sobrepasa el 0.8.

```
Confusion Matrix and Statistics

          Reference
Prediction    No   Yes
       No  36374  9136
       Yes  6852 16079

               Accuracy : 0.7664
                 95% CI : (0.7632, 0.7696)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4884
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.8415
            Specificity : 0.6377
         Pos Pred Value : 0.7993
         Neg Pred Value : 0.7012
              Precision : 0.7993
                 Recall : 0.8415
                     F1 : 0.8198
             Prevalence : 0.6316
         Detection Rate : 0.5315
   Detection Prevalence : 0.6650
      Balanced Accuracy : 0.7396

       'Positive' Class : No
```

Figura 9: Matriz de confusión para rpart

Finalmente en el siguiente gráfico podemos ver la curva roc conseguida mediante rpart.

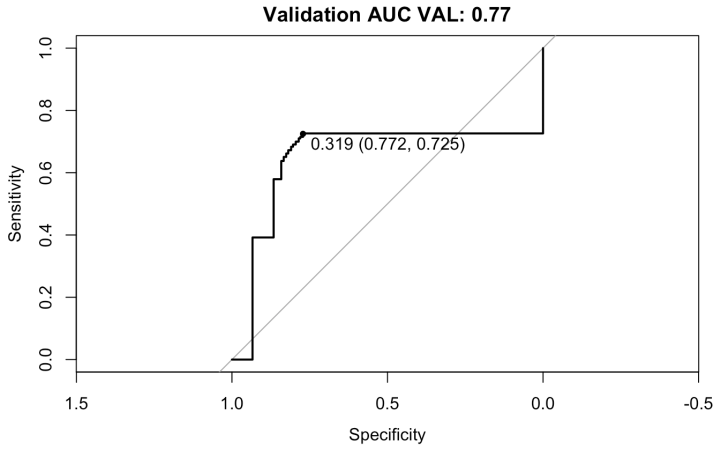

Figura 10: Curva roc para rpart

4.2. Random forest

Una vez explorado en gran medida rpart, nos lanzamos a trabajar con un clasificador que nos permitiera obtener mejores resultados. En este caso fue random forest.

Inicialmente nos encontramos los mismos problemas que veníamos acarreando, las ejecuciones se alargaban más horas de las esperadas. Una vez conseguido ejecutar el árbol más básico posible, que en esencia sería un rpart, ya estabamos listos para trabajar en un ajuste más óptimo. Esto nos llevó a una mejora considerable de la curva roc y de los diversos parámetros que nos ofrece la matriz de confusión. Llegando a un punto de inflexión en el que la curva roc llegó a mostrarnos valores superiores del 0.9.

En el punto anterior es cuando se sucedieron los debates en torno a la importancia de la contaminación de los datos al preprocesar nuestro conjunto de validación. Por tanto, obtener tal valor de curva roc era razonable, nuestros datos de validación estaban preprocesados por lo que nuestro modelo fácilmente obtenía buenos resultados. Tras esto se sucedieron una serie de pruebas con la partición de los datos al comienzo del preprocesado, lo que nos llevó a una gran redución de los valores de roc. Ahora nuestra validación apenas estaba preprocesada, tan solo se le habían eliminado ciertas variables.

Una vez aclarado la cuestión de la contaminación de los datos, y vuelto a nuestro punto inicial donde nuestra partición tenía lugar justo antes de la creación del modelo, nuestros esfuerzos se centraron en conseguir un modelo estable que en el que la mayoría de valores de la matriz de confusión llegaran a su valor óptimo. A la par de esto también intentamos mejorar nuestro score en la competición de kaggle que apenas conseguía pasar del 0.5 según la curva roc.

Para culminar nuestros intentos de mejora incluimos también en el preprocesado el análisis de componentes principales, lo que nos supuso una mejora más que considerable. Si bien en los resultados que obteníamos en nuestro conjunto de datos de validación no mejoraron excesivamente, ya que estos eran batante positivos desde antes, nuestro score en kaggle mediante el conjunto de test despegó totalmente, consiguiendo pasar de un valor aproximado de 0.56 según la curva roc, hasta 0.67 en el mejor de los casos obtenidos. Podemos concluir que este análisis final ha sido determinante para tener unos resultados considerables.

Finalmente probamos también con validación cruzada en este clasificador pero la mejora fue mínima y el tiempo de ejecución mucho mas largo, por lo que en el código que presentamos no haremos uso de esta validación cruzada.

```
rfCtrl <- trainControl(method = 'none',
                       number = 10,
                       repeats = 1,
                       search = 'random',
                       classProbs = TRUE )

# Numero de variables que tiene que predecir
mtry_ <- sqrt(ncol(train_clasificacion-1))
tunegrid <- data.frame(mtry = mtry_)

# RANDOM FOREST
rfModel <- train(target ~ .,
                 data = train_clasificacion,
                 method = "rf",
                 metric = "ROC",
                 ntree = 150,
                 tuneGrid = tunegrid,
                 trControl = rfCtrl)

#saveRDS(rfModel, file = "./copia_seguridad/rfModel_final.rds")
# Primera info del modelo
print(rfModel)
```

```
############### VAL ###############
prediction_rf_val <- predict(rfModel, val_clasificacion, type = "raw")
cm_train_rf_val <- confusionMatrix(prediction_rf_val,
                                   val_clasificacion[["target"]],
                                   mode = "everything",
                                   positive = "No")
cm_train_rf_val
# Mediante probabilidades y curva ROC
predictionValidationProb_rf_val <- predict(rfModel, val_clasificacion, type = "prob")
auc_rf_val <- roc(val_clasificacion$target,
                  predictionValidationProb_rf_val[["Yes"]],
                  levels = unique(val_clasificacion[["target"]]))
auc_rf_val

############### TEST ###############
prediction_rf_test <- predict(rfModel, test_clasificacion, type = "raw")
predictionValidationProb_rf_test <- predict(rfModel, test_clasificacion, type = "prob")


# Visualizamos
#roc_validation <- plot.roc(auc_rpart_val)
roc_validation_rf_val <- plot.roc(auc_rf_val,
                                  ylim=c(0,1),
                                  type = "S",
                                  print.thres = T,
                                  main=paste('Validation AUC VAL:',
                                             round(auc_rf_val$auc[[1]], 2)))
```

Una vez culminado todo el proceso los resultados son los siguientes. Respecto a la matriz de confusión destacar el accuracy de 0.9281, el kappa de 0.8438 y el F1 de 0.9439. Es evidente que los resultados son considerablemente mejores que los obtenidos mediante rpart.

```
Confusion Matrix and Statistics

          Reference
Prediction    No   Yes
       No  41412  3107
       Yes  1814 22108

               Accuracy : 0.9281
                 95% CI : (0.9261, 0.93)
    No Information Rate : 0.6316
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8438
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9580
            Specificity : 0.8768
         Pos Pred Value : 0.9302
         Neg Pred Value : 0.9242
              Precision : 0.9302
                 Recall : 0.9580
                     F1 : 0.9439
             Prevalence : 0.6316
         Detection Rate : 0.6051
   Detection Prevalence : 0.6505
      Balanced Accuracy : 0.9174

       'Positive' Class : No
```

Figura 11: Matriz de confusión para random forest

Finalmente en el siguiente gráfico podemos ver la curva roc conseguida mediante random forest.

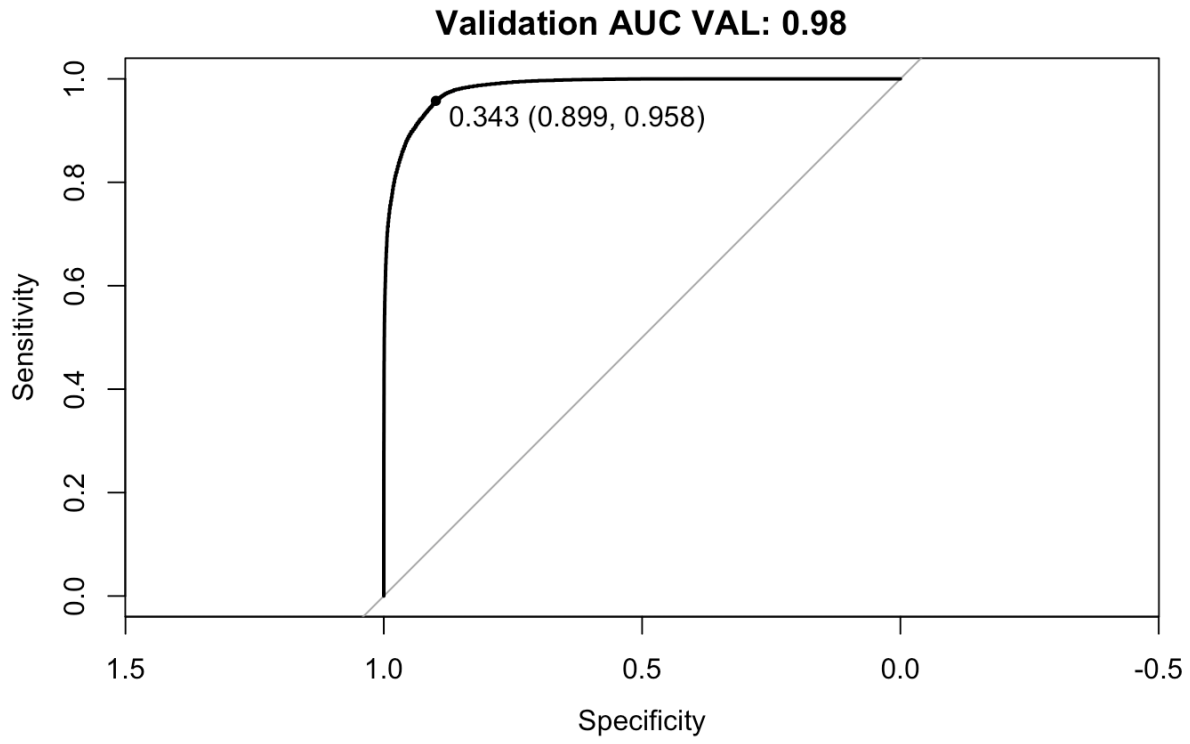

Figura 12: Curva roc para random forest

4.3. Submission kaggle

Como hemos comentado, nuestras pruebas se han extendido hasta la clasificación propia que kaggle ofrece. El código que crea dicho submission es el siguiente.

```
# Ajustamos para guardar
ID_code <- indices_test
target <- prediction_rf_test
# Creamos data_frame
dataFrame <- data.frame(ID_code,target)
# Formateamos los datos a su forma inicial
dataFrame <- dataFrame %>%
  mutate(target = as.numeric(ifelse(target == 'Yes', 1, 0)))
# Guardamos
write_csv(dataFrame, './copia_seguridad/sample_submission_rf_final.csv')

# Visualizamos comparativa 0-1
dataFrame <- dataFrame %>%
  mutate(target = as.factor(target))
ggplot(dataFrame) +
  geom_histogram(aes(x = target, fill = target), stat = 'count') +
  theme_hc()
```

Como podemos ver en el siguiente gráfico las predicciones que se han resuelto como positivas (1) también son considerablemente bajas.

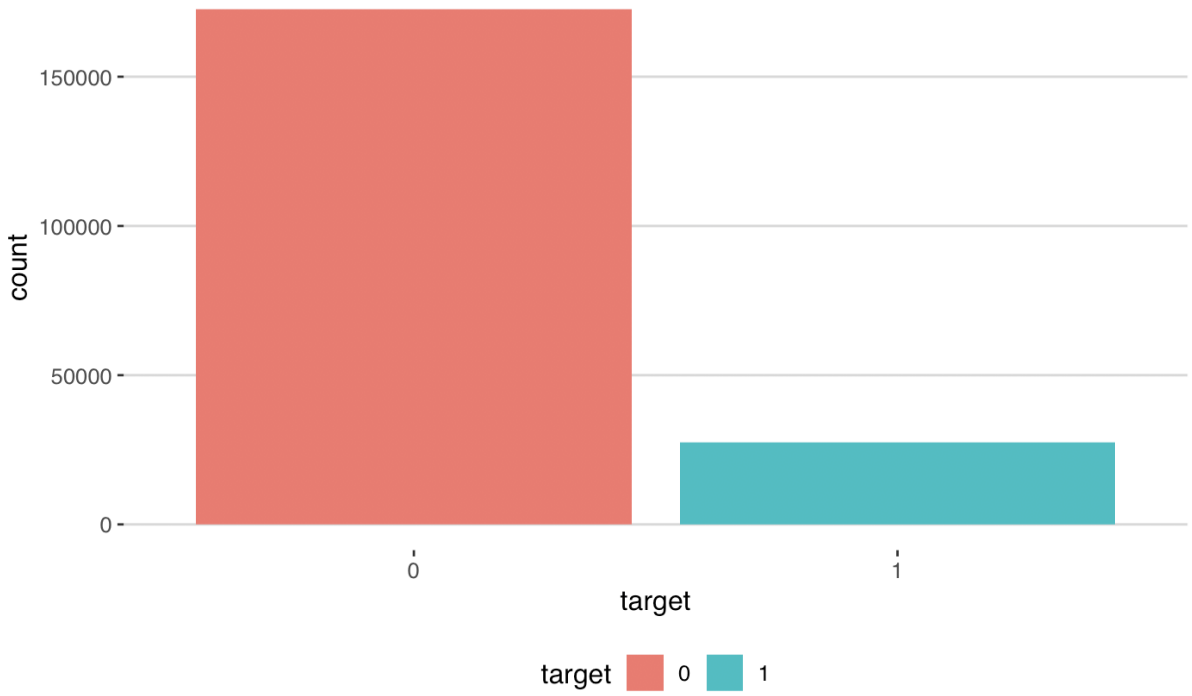

Figura 13: Balanceo en los resultados de test

5. Conclusiones

Como conclusiones al presente trabajo me gustaría exponer los resultados finales, siendo destacable el resultado obtenido en kaggle como podemos ver más abajo. Si bien este resultado es sobre el dataset de test obtenido desde dicha plataforma, los resultados obtenidos en el conjunto de validación son también interesantes.

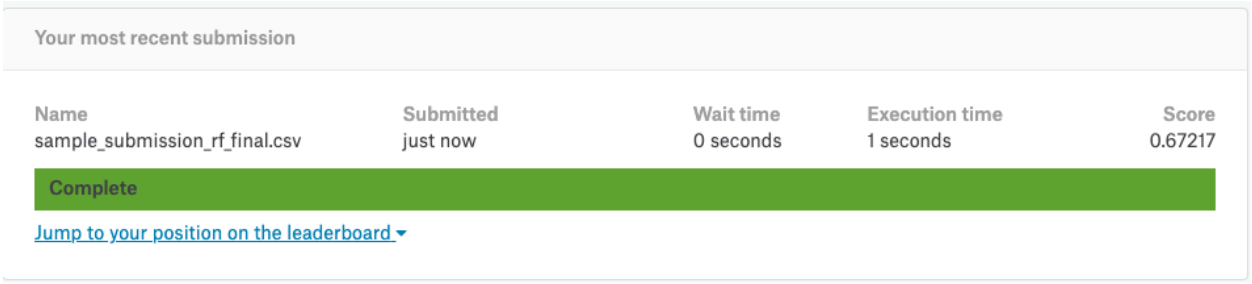

Figura 14: Score test kaggle

Por tanto, para culminar me gustaría expresar que la realización de este trabajo ha sido especialmente relevante para mi. Esto se debe a que hasta el mismo momento del comienzo de esta asignatura mis conocimientos sobre análisis de datos eran prácticamente nulos, y el transcurso de esta junto al de esta práctica han sido bastante didactivos y entretenidos.

También quisiera comentar la dificultad del dataset al que nos enfrentamos, que bajo mi punto de vista ha sido bastante complejo. Al tener ese claro desbalanceo llegar a conseguir un modelo con el que obtener unos resultados decentes ha sido dificil, y si además le sumamos el tiempo que conlleva cada prueba que realizamos, el tiempo necesario para desarrollar el trabajo se extiende considerablemente.

Por último, destacar la importancia del preprocesamiento de los datos, y es que sin este proceso sería prácticamente imposible el desarrollo de estos análisis. La cantidad de datos innecesarios, los datos perdidos o que te están produciendo ruido, junto a tiempos de ejecución infinitos debidos a esta gran cantidad de datos, hacen imposibles estás prácticas. Esta cuestión ha quedado más que evidente en el transcurso de la práctica y en los resultados obtenidos.

6. Bibliografía

Referencias

[1] Mice https://www.rdocumentation.org/packages/mice/versions/3.4.0/topics/mice

[2] Mice https://www.r-bloggers.com/imputing-missing-data-with-r-mice-package/

[3] Smote https://www.rdocumentation.org/packages/DMwR/versions/0.4.1/topics/SMOTE

[4] PreProcess https://www.rdocumentation.org/packages/caret/versions/6.0-82/topics/preProcess

[5] Random Forest https://rpubs.com/phamdinhkhanh/389752