Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Máster Profesional en Ingeniería Informática

Curso 2020/2021

PRÁCTICA 1: PRE-PROCESAMIENTO DE DATOS Y CLASIFICACIÓN BINARIA

Sistemas Inteligentes para la Gestión en la Empresa

Breve descripción

En esta práctica se analizarán datos del experimento ATLAS del CERN-LHC, que perseguía la identificación experimental de la partícula bosón de Higgs.

Autor

Álvaro de la Flor Bonilla (alvdebon@correo.ugr.es) 15408846-L

Propiedad Intelectual

RESUMEN

En esta práctica se analizarán datos del experimento ATLAS del CERN-LHC, que perseguía laidentificación experimental de la partícula bosón de Higgs.

El problema consiste en predecir si un registro de evento corresponde al decaimiento de un bosón de Higgs o se trata de ruido de fondo.

Se trabajará sobre el conjunto de datos ofrecido en la competición de Kaggle Higgs Boson Machine Learning Challenge: https://www.kaggle.com/c/higgs-boson/.El conjunto de datos se puede descargar directamente desde este enlace: http://sl.ugr.es/higgs_sige. Los eventos recogidos en este conjunto de datos han sido generados de forma sintética con un simulador.

La descripción de las variables se encuentra en la sección Datadel desafío de Kaggle. Cada evento está caracterizado por un identificador, los valores de 30 variables y la etiqueta correspondiente ('b': ruido de fondo, 's': bosón). La descripción detallada de las variables se encuentra en el siguiente enlace: https://higgsml.lal.in2p3.fr/files/2014/04/documentation_v1.8.pdf.

ÍNDICE DEL PROYECTO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

No se encuentran elementos de tabla de ilustraciones.

1 CONJUNTO DE DATOS

Como anteriormente nombramos, en esta práctica se analizará el conjunto de datos procedente de *"Kaggle"* denominado *"HiggsBosonCompetition"*. Los ficheros que se van a utilizar ("training.csv" y "test.csv") se encuentran disponibles en la carpeta *"data"*. Aun así, para evitar posibles errores, se provee una solución alternativa para la descarga de datos directa del repositorio.

```
```{r descargar}
if(!file.exists("data/training.csv")) {
  library(httr)
  url <- "http://sl.ugr.es/higgs_sige"
  GET(url, write_disk(temp <- tempfile(fileext = ".zip")))
  unzip(temp, exdir = "data")
  unlink(temp)
}
```
```

Ilustración 1 – Descarga de datos

De forma prioritario como hemos dicho antes, se utilizarán los datos con los que contamos en nuestro directorio. Tendremos que realizar dos asignaciones, uno para el conjunto de entrenamiento y otro para el conjunto de validación.

```
```{r leer-entrenamiento}
training_data_raw_higgs <- read_csv("data/training.csv")
training_data_raw_higgs
```
```

Ilustración 2 – Leer datos de entrenamiento

```
```{r leer-validacion}
test_data_raw_higgs <- read_csv("data/test.csv")
test_data_raw_higgs
```
```

Ilustración 3 – Leer datos de validación

En ambas instrucciones lo único que se ha hecho es la lectura del archivo CSV principal y su asociación con la respectiva variable en cada caso que hemos decidido. A partir de este momento, para el desarrollo de la práctica trabajaremos con la variable *"training_data_raw_higgs"* que contendrá el conjunto de datos con el que realizaremos el entrenamiento y estudio de nuestro conjunto de observaciones.

2 ANÁLISIS EXPLORATORIO

2.1 Estado de los datos

En primer lugar, vamos a realizar una primera observación del estado de los datos. Para ello haremos uso de la función *"df_status"* la cual permite conocer los valores de todos los campos de un conjunto con el objetivo de saber la cantidad y el porcentaje de ceros o valores perdidos. Esta función también permite saber la cantidad de valores únicos que hay para cada campo, así como saber si una variable es categórica, numérica…

Pero antes de hacer uso de esta función, tal y como se nos indica en el guion de práctica realizaremos la recodificación de los valores perdidos como *"NA"*, es decir, los valores (-999.0) los transformamos en *"NA"*.

```
```{r recodificar}
training_data_higgs <- training_data_raw_higgs %>%
  na_if(-999.0)
training_data_higgs
```
```

Ilustración 4 – Recodificación de valores

Una vez realizado este primer tratado de datos tenemos nuestro conjunto de datos listo para realizar el primer estudio observatorio sobre él.

```
```{r status}
df_status(training_data_higgs)
```
```

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
EventId	0	0.00	0	0.00	0	0	numeric	250000
DER_mass_MMC	0	0.00	38114	15.25	0	0	numeric	108337
DER_mass_transverse_met_lep	3	0.00	0	0.00	0	0	numeric	101637
DER_mass_vis	0	0.00	0	0.00	0	0	numeric	100558
DER_pt_h	41	0.02	0	0.00	0	0	numeric	115563
DER_deltaeta_jet_jet	6	0.00	177457	70.98	0	0	numeric	7086
DER_mass_jet_jet	0	0.00	177457	70.98	0	0	numeric	68365
DER_prodeta_jet_jet	58	0.02	177457	70.98	0	0	numeric	16592
DER_deltar_tau_lep	0	0.00	0	0.00	0	0	numeric	4692
DER_pt_tot	39	0.02	0	0.00	0	0	numeric	59042

1-10 of 33 rows

Ilustración 5 – Uso de la función *"df_status"*

Además, de forma conjunta a esta función, también hemos decidido utilizar *"vir_miss"* que nos permite saber de una forma rápida y gráfica el porcentaje de valores perdidos presente en nuestro conjunto de datos.

```
```{r tabla}
vis_miss(training_data_higgs, warn_large_data = FALSE)
```
```

Ilustración 6 – Conjunto de valores perdidos

De las observaciones anteriores podemos extraer algunos datos interesantes como:

- El valor de *"EventId"* es único.
- Existen dos valores diferentes para *"Label"*, que es nuestro objetivo de clasificación
- Existen cuatro valores diferentes para *"PRI_jet_num"* (el número de jets; un jet es una lluvia de hadrones, que se originan a partir de un quark o un gluón, agrupados tras producirse en colisiones de partículas.)
- En el 65,73% de los casos (164333/250000) NO ocurre una señal de aparición del Boson *"backgroud (b)"*.
- Aparecen valores perdidos (*"NA"*) en muchas de las variables.
    - *DER_mass_MMC* (15.25%)
    - *DER_deltaeta_jet_jet* (70.98%)
    - *DER_mass_jet_jet* (70.98%)
    - *DER_prodeta_jet_jet* (70.98%)
    - *DER_lep_eta_centrality* (70.98%)
    - *PRI_jet_leading_pt* (39.97%)
    - *PRI_jet_leading_eta* (39.97%)
    - *PRI_jet_leading_phi* (39.97%)
    - *PRI_jet_subleading_pt* (70.98%)
    - *PRI_jet_subleading_eta* (70.98%)
    - *PRI_jet_subleading_phi* (70.98%)

Siguiendo con los estudios anteriores, podemos observar que existen dos clases (`Label` y `PRI_jet_num`) que cuentan con un número muchísimo más reducido de variables.

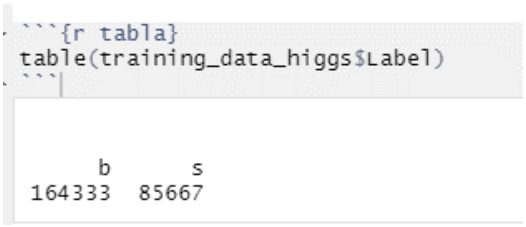


Ilustración 7 - Balanceo de clase

En primer lugar, estudiaremos la clase *"Label"*. Como vemos, existen 164 333 instancias de *"backgroud"* es decir, que no han mostrado señales de aparición del *"bosson"* lo cual representa un 65.73% del total (siendo el 34,27% restante observación que en este caso si resaltan la aparición del *"bosson de higgs"*). En un primer momento a partir de este dato podemos afirmar el desbalanceo de esta clase, por lo que tendremos que trabajar posteriormente en ella.

Además, podemos observar también este este desbalanceo del conjunto de datos en la variable *"Label"* a partir del siguiente gráfico, utilizando para ello la función *"ggplot"*.

```r
```{r clases, warning=FALSE}
ggplot(training_data_higgs) +
  geom_histogram(aes(x = Label, fill = as.factor(Label)), stat = "count") +
  labs(x = "", y = "") +
  scale_fill_discrete(name ="Clase", labels=c("(b)ackground", "higg(s)"))
```
```

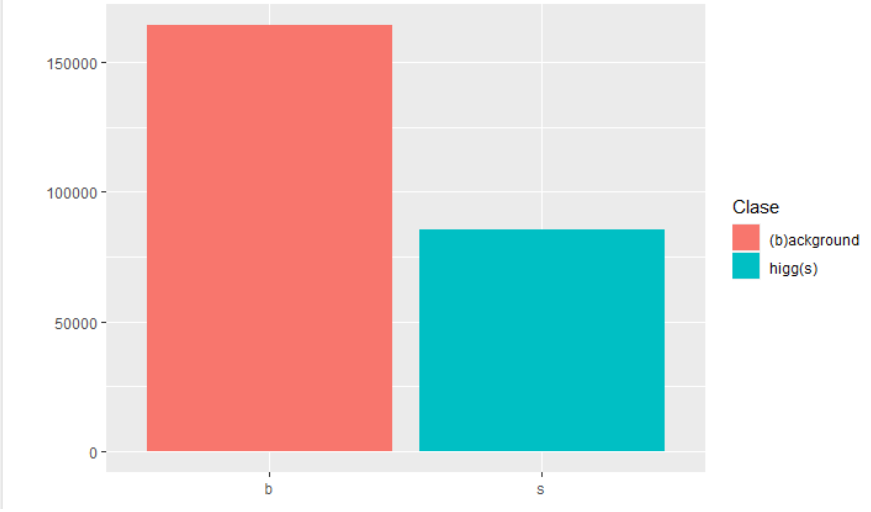


Ilustración 8 – Balanceo de la clase *"Label"*

De igual forma, también nos ha parecido interesante repetir el proceso anterior, pero en este caso utilizando la variable *"PRI_jet_num"*.

```r
```{r tabla}
table(training_data_higgs$PRI_jet_num)
```

    0   1   2   3
99913 77544 50379 22164

```{r clases, warning=FALSE}
ggplot(training_data_higgs) +
  geom_histogram(aes(x = PRI_jet_num, fill = as.factor(PRI_jet_num)), stat = "count") +
  labs(x = "", y = "") +
  scale_fill_discrete(name ="Number of Jets", labels=c("1", "2", "3", "4"))
```
```

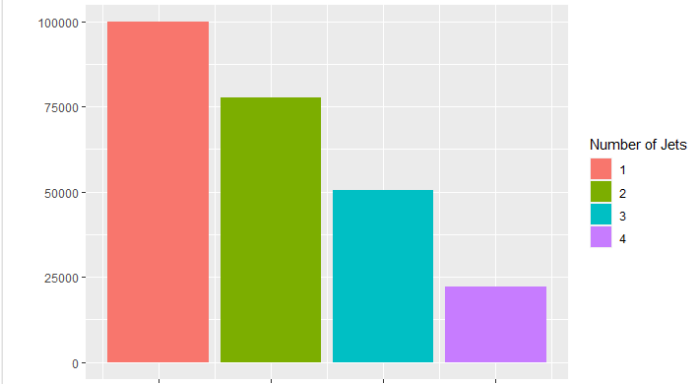


Ilustración 9 – Análisis de *"PRI_jet_num"*

Una vez más, en este caso también podemos observar un existente desbalanceo en el conjunto de las observaciones que será necesario corregir si posteriormente seguimos manteniendo esta variable.

Por otro lado, también podemos analizar como están distribuidas las clases para los valores de una determinada variable, en nuestro caso vamos a comparar las variables *"DER_met_phi_centrality"* y *"PRI_jet_num"* con *"Label".*

Ilustración 10 – Distribución de variables

Además, también podemos considerar una pseudo-distribución de probabilidad como la siguiente (solo se mostrará la existente con las variables *"Label"* y *"PRI_jet_num"*.

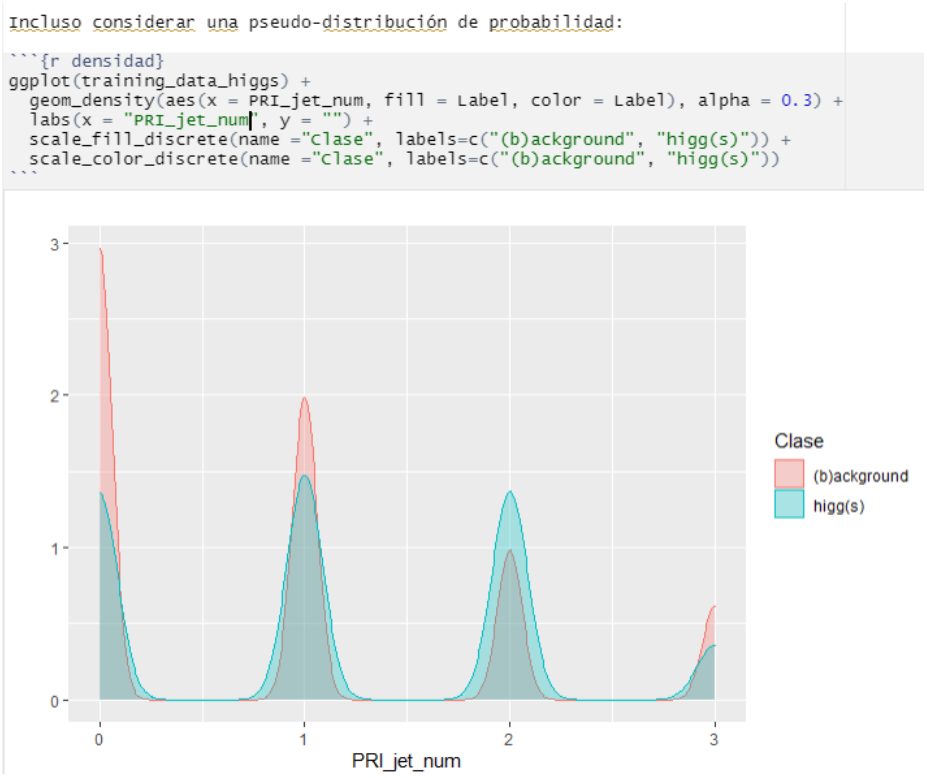


Ilustración 11 - Distribución de probabilidad

3 PRE-PROCESAMIENTO

3.1 Transformación y limpieza de valores numéricos

En esta sección en primer lugar trataremos los valores perdidos, para ello hemos decido utilizar el valor 39% como máximo (siguiendo las recomendaciones de alguno de los foros que hemos visitado).

También aprovecharemos para eliminar aquellas variables cuyos valores presentan más del 80% de valores distintos, ya que realmente no aportan información útil para nuestro sistema. Para ello hemos realizado este tratamiento.

```r
```{r}
status_higgs <- df_status(training_data_higgs)

## columnas con NAs
na_cols_higgs <- status_higgs %>%
  filter(p_na > 39) %>%
  select(variable)

## columnas con valores diferentes
dif_cols <- status_higgs %>%
  filter(unique > 0.8 * nrow(training_data_raw_higgs)) %>%
  select(variable)

## eliminar columnas
remove_cols_higgs <- bind_rows(
  list(na_cols_higgs, dif_cols)
)

data_reduced_higgs <- training_data_higgs %>%
  select(-one_of(remove_cols_higgs$variable))
df_status(training_data_higgs)
df_status(data_reduced_higgs)
```
```

Ilustración 12 - Primer tratamiento

En el conjunto de líneas anterior lo primero que hemos hecho es seleccionar las columnas que presentan más de un 39 % de valores perdidos con un sencillo filtro y las hemos añadido (sus nombres) a la variable *"na_cols_higgs"*. Hemos repetido exactamente el mismo proceso para el caso de las columnas que presentan un valor superior al 80 % de valores distintos. Finalmente, en el último paso hemos filtrado estas variables de nuestro dataset original quedándonos con un nuevo conjunto de datos filtrados nombrado *"data_reduced_higgs"*. De hecho, tras este tratamiento hemos conseguido reducir de 33 a 22 el número de variable con el que cuenta nuestro conjunto.

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
DER_mass_MMC	0	0.00	38114	15.25	0	0	numeric	108337
DER_mass_transverse_met_lep	3	0.00	0	0.00	0	0	numeric	101637
DER_mass_vis	0	0.00	0	0.00	0	0	numeric	100558
DER_pt_h	41	0.02	0	0.00	0	0	numeric	115563
DER_deltar_tau_lep	0	0.00	0	0.00	0	0	numeric	4692
DER_pt_tot	39	0.02	0	0.00	0	0	numeric	59042
DER_sum_pt	0	0.00	0	0.00	0	0	numeric	156098
DER_pt_ratio_lep_tau	0	0.00	0	0.00	0	0	numeric	5931
DER_met_phi_centrality	53	0.02	0	0.00	0	0	numeric	2829
PRI_tau_pt	0	0.00	0	0.00	0	0	numeric	59639

1-10 of 22 rows Previous 1 2 3 Next

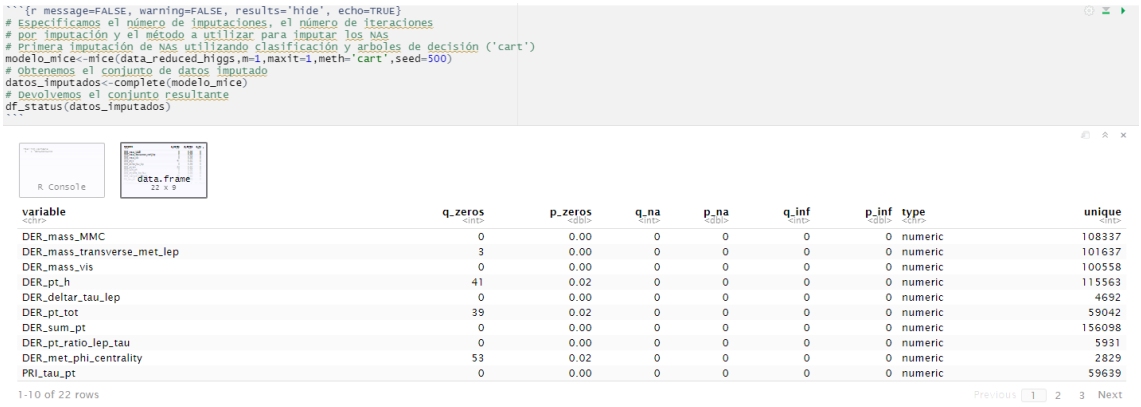
Ilustración 13 – Primer tratamiento

Solo queda una variable con valores perdidos (15%), que se los imputaremos a continuación.

3.2 Tratamiento de valores perdidos, imputación de valores

En este apartado procedemos a tratar los numerosos valores perdidos que contiene el conjunto de datos. Si bien existen multitud de técnicas para aplicar, en este caso utilizaremos la imputación de tipo *"NAS"*.

Es una de las más utilizadas según las búsquedas que he ido realizando. Para su aplicación hemos decidido hacer uso de la librería *"MICE"* debido a la multitud de parametrización y elección de operaciones que permite.



```{r message=FALSE, warning=FALSE, results='hide', echo=TRUE}
# Especificamos el número de imputaciones, el número de iteraciones
# por imputación y el método a utilizar para imputar los NAs
# Primera imputación de NAs utilizando clasificación y arboles de decisión ('cart')
modelo_mice<-mice(data_reduced_higgs,m=1,maxit=1,meth='cart',seed=500)
# Obtenemos el conjunto de datos imputado
datos_imputados<-complete(modelo_mice)
# Devolvemos el conjunto resultante
df_status(datos_imputados)
```

R Console

data.frame
22 x 9

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
DER_mass_MMC	0	0.00	0	0	0	0	numeric	108337
DER_mass_transverse_met_lep	3	0.00	0	0	0	0	numeric	101637
DER_mass_vis	0	0.00	0	0	0	0	numeric	100558
DER_pt_h	41	0.02	0	0	0	0	numeric	115563
DER_deltar_tau_lep	0	0.00	0	0	0	0	numeric	4692
DER_pt_tot	39	0.02	0	0	0	0	numeric	59042
DER_sum_pt	0	0.00	0	0	0	0	numeric	156098
DER_pt_ratio_lep_tau	0	0.00	0	0	0	0	numeric	5931
DER_met_phi_centrality	53	0.02	0	0	0	0	numeric	2829
PRI_tau_pt	0	0.00	0	0	0	0	numeric	59639

1-10 of 22 rows

Previous 1 2 3 Next

Ilustración 14 – Imputación de valores perdidos

En cuanto a este método, como parámetros hemos usado:

- **data** que contiene el conjunto de todos los valores con los que estamos tratando.
- **m** que es el número de múltiples imputaciones (hemos seleccionado 1).
- **maxit** indica el número de iteraciones.
- **meth** hace referencia al método de imputación que se va a usar. En nuestro caso hemos utilizado *"cart"* que imputa los datos faltantes univariantes utilizando árboles de clasificación y regresión.

Una vez realizado este procedimiento no existen variables pérdidas en nuestro conjunto de datos que a partir de ahora lo llamaremos *"datos_imputados"*.

3.3 Correlación

El objetivo de esta sección es estudiar la correlación entre las variables y la columna a predecir, que en el caso de esta práctica es la columna *"Label"*. Realmente lo que buscamos es conocer si existe alguna variable o grupo de variables que expliquen el comportamiento de la variable categórica a predecir. También por otro lado nos vendría bien conocer si existen variables muy correladas entre sí, ya que podríamos reducir la dimensionalidad eliminado las columnas cuyo coeficiente es muy alto ya que estarían aportando la misma información.

```
#### Correlación entre las variables y la columna a predecir
```{r message=FALSE, warning=FALSE, results='hide', echo=TRUE}
data_num <-
  datos_imputados %>%
  mutate_if(is.character, as.factor) %>%
  mutate_if(is.factor, as.numeric)
cor(data_num)
```
```

Ilustración 15 – Cálculo de correlación entre variables

Como fruto del código anterior obtenemos la siguiente matriz de relación de correlaciones entre las distintas variables que cuenta nuestro dataset.

	DER_mass_MMC	DER_mass_transverse_met_lep	DER_mass_vis	DER_pt_h	DER_deltar_tau_lep	DER_pt_tot	DER_sum_pt	DER_pt_ratio_lep_tau
DER_mass_MMC	1.000000000	0.1793575468	0.9067450029	0.065483476	0.5449058921	0.0453064010	0.166721128	7.728756e-02
DER_mass_transverse_met_lep	0.179357547	1.0000000000	0.1901094877	-0.249115929	0.0432514093	0.0177575270	-0.146836679	3.495036e-01
DER_mass_vis	0.906745003	0.1901094877	1.0000000000	-0.062562021	0.5797116166	-0.0007021339	0.088685244	9.748985e-02
DER_pt_h	0.065483476	-0.2491159291	-0.0625620208	1.000000000	-0.5393792155	0.3105009862	0.832733067	8.918737e-02
DER_deltar_tau_lep	0.544905892	0.0432514093	0.5797116166	-0.539379215	1.0000000000	-0.1480807915	-0.432603440	4.704619e-02
DER_pt_tot	0.045306401	0.0177575270	-0.0007021339	0.310500986	-0.1480807915	1.0000000000	0.381159784	3.919290e-02
DER_sum_pt	0.166721128	-0.1468366794	0.0886852442	0.832733067	-0.4326034399	0.3811597842	1.000000000	1.087911e-01
DER_pt_ratio_lep_tau	0.077287562	0.3495036141	0.0974898450	0.089187365	0.0470461910	0.0391929033	0.108791092	1.000000e+00
DER_met_phi_centrality	0.047382987	-0.4197573354	-0.0908458625	0.539356019	-0.2054414978	0.1784479071	0.420679094	-3.880267e-02
PRI_tau_pt	0.280934657	-0.1454641450	0.2900111589	0.407421442	-0.2020349898	0.0957537343	0.485847266	-4.746325e-01
PRI_tau_eta	0.004511727	-0.0021091715	0.0021265979	0.001665387	0.0036322272	0.0035958956	0.002037141	6.109418e-04
PRI_tau_phi	-0.003103116	0.0011320839	-0.0036240957	0.005247808	-0.0112290140	0.0014521885	0.003931113	3.629803e-06
PRI_lep_pt	0.355160521	0.3106475180	0.4054824410	0.360938860	-0.0699566710	0.1096165666	0.460938009	7.011422e-01
PRI_lep_eta	0.004718463	-0.0067770678	0.0021956598	0.008354230	0.0006988287	0.0079865871	0.008780885	2.357009e-03
PRI_lep_phi	-0.002620260	0.0003403302	-0.0020175330	-0.002923037	-0.0007755337	-0.0042494477	-0.001891877	-5.052982e-03
PRI_met	0.119104455	0.1837163774	-0.0873303668	0.679585496	-0.4023449144	0.2697386616	0.520129360	5.338559e-02
PRI_met_phi	-0.001161742	-0.0159253407	-0.0014667742	0.008584925	-0.0015697713	0.0025148368	0.006712046	-1.186095e-03
PRI_met_sumet	0.134383362	-0.1678106395	0.0533004391	0.782546973	-0.4070017792	0.4489253138	0.904481460	4.990357e-02
PRI_jet_num	0.066815517	-0.2105370655	-0.0268600335	0.623401443	-0.3479044282	0.3604085395	0.758503365	5.631197e-02
PRI_jet_all_pt	0.052621531	-0.2100089179	-0.0529024313	0.808616264	-0.4487369594	0.4033824289	0.965628389	7.911642e-02
Weight	0.025248159	0.4198434262	0.1021718563	-0.414084439	0.1978807760	-0.2195074521	-0.414826552	1.883977e-01
Label	0.031345031	-0.3514279559	-0.0140552738	0.192526329	0.0122454813	-0.0152874267	0.153235932	-1.953979e-01

	DER_met_phi_centrality	PRI_tau_pt	PRI_tau_eta	PRI_tau_phi	PRI_lep_pt	PRI_lep_eta	PRI_lep_phi	PRI_met	PRI_met_phi
DER_mass_MMC	0.0473829866	2.809347e-01	0.0045117270	-3.103116e-03	0.3551605206	4.718463e-03	-2.620260e-03	0.1191044546	-0.0011617416
DER_mass_transverse_met_lep	-0.4197573354	-1.454641e-01	-0.0021091715	1.132084e-03	0.3106475180	-6.777068e-03	3.403302e-04	0.1837163774	-0.0159253407
DER_mass_vis	-0.0908458625	2.900112e-01	0.0021265979	-3.624096e-03	0.4054824410	2.195660e-03	-2.017533e-03	-0.0873303668	-0.0014667742
DER_pt_h	0.5393560189	4.074214e-01	0.0016653874	5.247808e-03	0.3609388604	8.354230e-03	-2.923037e-03	0.6795854956	0.0085849246

Ilustración 16 – Relaciones de correlación

Como la calidad visual es muy pobre, hemos decido utilizar los datos anteriores para crear un mapa de calor aprovechando la tabla de correlaciones.

```
```{r visualizacion-correlacion}
rcorr(as.matrix(data_num))
corrplot(cor(data_num), type = "upper", diag = F, order = "hclust", tl.col = "black", t
heatmap(x = cor(data_num), symm = TRUE)
```
```

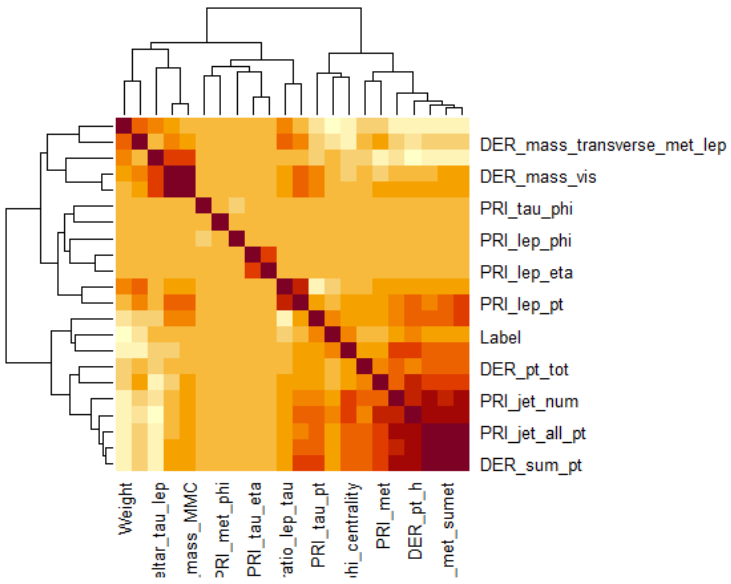


Ilustración 17 - Mapa de calor

Visualmente, podemos ver como a medida que aumenta la correlación entre las variables comparadas es coloreado el cuadro de un color rojo de mayor intensidad que su alrededor.

También podemos calcular la correlación haciendo uso de la función *"cor_target"* y quedarnos con las variables que tienen una correlación por encima de 0.03 (V.A.).

Ahora sí podemos calcular la correlación:

```{r}
cor_target <- correlation_table(data_num, target='Label')
```

Y quedarnos solo con las variables que tienen una correlación por encima del 0.03 (en valor absoluto):

```{r}
important_vars <- cor_target %>%
  filter(abs(Label) >= 0.03)

data <- datos_imputados %>%
  select(one_of(important_vars$variable))
df_status(data)
```

Ilustración 18 – Cálculo de la correlación de variables

Fruto del filtrado que anterior, hemos conseguido reducir en número de filas a un total de 13. A partir de este momento, podemos comenzar a crear un modelo predictivo para evaluar la importancia de las variables que tenemos hasta ahora. Para hacer esta evaluación hemos utilizado el siguiente código.

```{r entrenamiento-rpart}
# Parámetros
rpartCtrl <- trainControl(verboseIter = F, classProbs = TRUE, summaryFunction = twoClassSummary)
rpartParametersGrid <- expand.grid(.cp = c(0.01, 0.05))
# Conjuntos de entrenamiento y validación
trainIndex <- createDataPartition(data$Label, p = .4, list = FALSE, times = 1)
train <- data[trainIndex, ]
# Entrenamiento del modelo
rpartModel <- train(Label ~ .,
                    data = train,
                    method = "rpart",
                    metric = "ROC",
                    trControl = rpartCtrl,
                    tuneGrid = rpartParametersGrid)
# Visualización del modelo
rpart.plot(rpartModel$finalModel)
```

Ilustración 19 – Creación de modelo predictivo

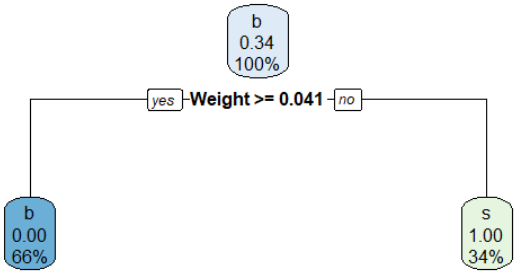


Ilustración 20 – Resultado de modelo predictivo

Como resultado podemos ver algo extraño, ya que nuestro modelo indica que la predicción de nuestra variable clasificatoria *"Label"* solo depende de *"Weight"*.

```{r validacion-rpart}
# Predicciones con clases
val        <- data[-trainIndex, ]
prediction <- predict(rpartModel, val, type = "raw")
# Predicciones con probabilidades
predictionValidationProb <- predict(rpartModel, val, type = "prob")
```

Y calculamos las métricas de calidad del clasificador (matriz de confusión y curva ROC):

```{r validacion-metricas}
cm_train <- confusionMatrix(table(prediction, val[["Label"]]))
cm_train
auc <- roc(val$Label, predictionValidationProb[["b"]], levels = unique(val[["Label"]]))
roc_validation <- plot.roc(auc,
                           ylim=c(0,1),
                           type = "S" ,
                           print.thres = TRUE,
                           main=paste('Validation AUC:', round(auc$auc[[1]], 2)))
```

Ilustración 21 - Validación de datos

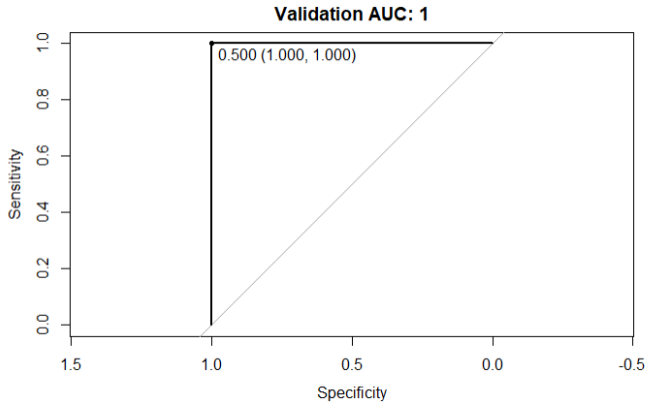


Ilustración 22 – Validación AUC

Visto la validación que podemos ver en la ilustración 22 y viendo que la variable *"Weight"* no se encuentra en los datos de validación vamos a optar por eliminarla y repetir el proceso de predicción.

```
```{r validacion-metricas}
data$Weight <- NULL
```
```

```
```{r entrenamiento-rpart}
# Parámetros
rpartCtrl <- trainControl(verboseIter = F, classProbs = TRUE, summaryFunction = twoClassSummary)
rpartParametersGrid <- expand.grid(.cp = c(0.01, 0.05))
# Conjuntos de entrenamiento y validación
trainIndex <- createDataPartition(data$Label, p = .4, list = FALSE, times = 1)
train <- data[trainIndex, ]
# Entrenamiento del modelo
rpartModel <- train(Label ~ .,
                    data = train,
                    method = "rpart",
                    metric = "ROC",
                    trControl = rpartCtrl,
                    tuneGrid = rpartParametersGrid)
# Visualización del modelo
rpart.plot(rpartModel$finalModel)
```
```

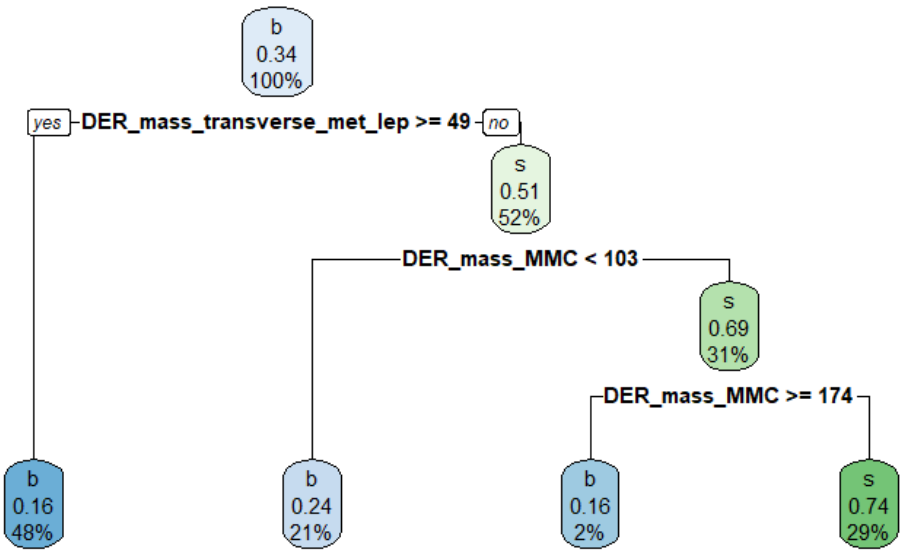


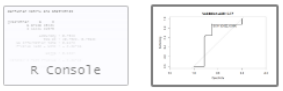
Ilustración 23 – Resultado del nuevo modelo predictivo

Este modelo predictivo parece tener mucho más sentido que el anterior, por lo que vamos a calcular la validación y la métrica de calidad del clasificador.

```r
# Predicciones con clases
val        <- data[-trainIndex, ]
prediction <- predict(rpartModel, val, type = "raw")
# Predicciones con probabilidades
predictionValidationProb <- predict(rpartModel, val, type = "prob")
```

Y calculamos las métricas de calidad del clasificador (matriz de confusión y curva ROC):

```r
cm_train <- confusionMatrix(table(prediction, val[["Label"]]))
cm_train
auc <- roc(val$Label, predictionValidationProb[["b"]], levels = unique(val[["Label"]]))
roc_validation <- plot.roc(auc,
                           ylim=c(0,1),
                           type = "S" ,
                           print.thres = TRUE,
                           main=paste('Validation AUC:', round(auc$auc[[1]], 2)))
```



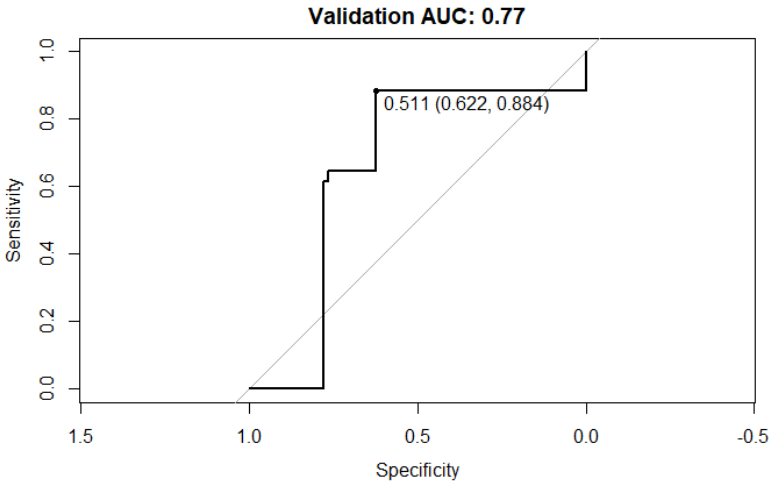


Ilustración 24 – Validación del modelo predictivo

En esta ocasión los valores obtenidos son muchos más lógicos que los obtenidos en el caso anterior.

3.4 Outliers

Los *"outliers"* son aquellos valores que se caracterizan por estar fuera del rango normal para su atributo, es decir, su valor está muy diferenciado al resto del conjunto y pueden causar problemas a la hora de utilizar el calificador.

```r
datos_finales<-data[, -which(names(data) %in% c("Label"))]
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, Label=data$Label)
# Devolvemos los datos resultantes
datos_finales
```

Ilustración 25 – Eliminación de outliers

3.5 Balanceo de la clase a predecir

Como pudimos observar en el análisis exploratorio, la mayoría de las observaciones perteneces al conjunto de *"background"*, es decir, a no tener evidencias del bosson. Para evitar este problema se va a llegar a cabo técnicas de *"oversampling"*, es decir, se producirán más muestras mientras se restan algunas de la clase mayoritaria para balancear ambas.

```r
data_rose <- ROSE(Label ~ ., data = datos_finales, seed = 1)$data
```

```r clases, warning=FALSE
ggplot(data_rose) +
  geom_histogram(aes(x = Label, fill = as.factor(Label)), stat = "count") +
  labs(x = "", y = "") +
  scale_fill_discrete(name ="Clase", labels=c("(b)ackground", "higg(s)"))
```

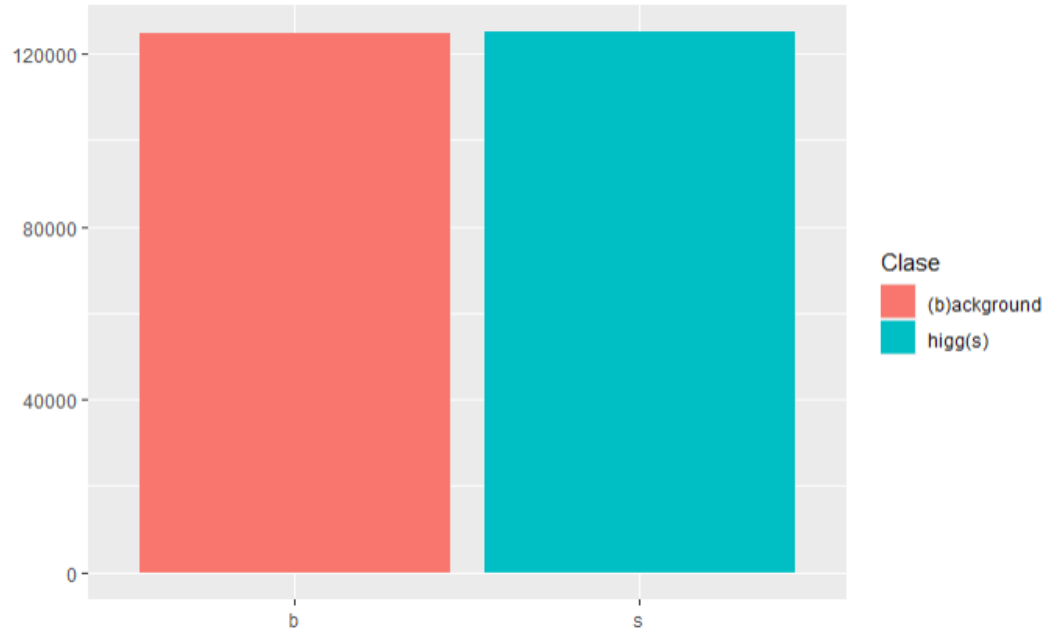


Ilustración 26 - Balanceo de clase

Para el balanceo de las clases hemos utilizado la librería *"ROSE"*, la cual requiere como parámetros:

- La variable por balancear.
- Conjunto de nuestros datos preprocesados.

Como vemos en la ilustración 26 se ha producido un reparto equitativo 50% a 50%.

4 CLASIFICACIÓN

4.1 Evaluación

Lo primero que vamos a hacer es crear una función para evaluar los distintos modelos creados.

```r
#' Cálculo de valores ROC
#' @param data Datos originales
#' @param predictionProb Predicciones
#' @param target_var Variable objetivo de predicción
#' @param positive_class Clase positiva de la predicción
#'
#' @return Lista con valores de resultado \code{$auc}, \code{$roc}
#'
#' @examples
#' rfModel <- train(Class ~ ., data = train, method = "rf", metric = "ROC", trControl = rfCtrl, tuneGrid = rfParametersGrid)
#' roc_res <- my_roc(data = validation, predict(rfModel, validation, type = "prob"), "Class", "Good")
my_roc <- function(data, predictionProb, target_var, positive_class) {
  auc <- roc(data[[target_var]], predictionProb[[positive_class]], levels = unique(data[[target_var]]))
  roc <- plot.roc(auc, ylim=c(0,1), type = "S" , print.thres = T, main=paste('AUC:', round(auc$auc[[1]], 2)))
  return(list("auc" = auc, "roc" = roc))
}
```

Ilustración 27 – Evaluación ROC

4.2 Partición de los datos

Vamos a utilizar los datos que hemos procesado hasta ahora para crear un por un lado el conjunto de entrenamiento y por otro el conjunto de validación.

```r
trainIndex <- createDataPartition(data_rose$Label, p = .75, list = FALSE)
train <- data[ trainIndex, ]
val   <- data[-trainIndex, ]
```

Ilustración 28 – Partición de datos

4.3 RPART

4.3.1 Entrenamiento

Vamos a crear dos modelos utilizando el método *"rpart"* en el primero de ellos no utilizaremos la validación cruzada y en el segundo de ellos sí.

Modelo 1:

```r
rpartCtrl <- trainControl(
  verboseIter = F,
  classProbs = TRUE,
  summaryFunction = twoClassSummary)
rpartParametersGrid <- expand.grid(
  .cp = c(0.001, 0.01, 0.1, 0.5))
rpartModel1 <- train(
  Label ~ .,
  data = train,
  method = "rpart",
  metric = "ROC",
  trControl = rpartCtrl,
  tuneGrid = rpartParametersGrid)
```

Modelo 2 (con validación cruzada):

```r
rpartCtrl2 <- trainControl(
  verboseIter = F,
  classProbs = TRUE,
  method = "repeatedcv",
  number = 10,
  repeats = 1,
  summaryFunction = twoClassSummary)
rpartModel2 <- train(Label ~ .,
                     data = train,
                     method = "rpart",
                     metric = "ROC",
                     trControl = rpartCtrl2,
                     tuneGrid = rpartParametersGrid)
```

Ilustración 29 – Creación de modelos

4.3.2 Visualización

A continuación, vamos a visualizar el primer modelo (omitimos el segundo ya que es exactamente igual).

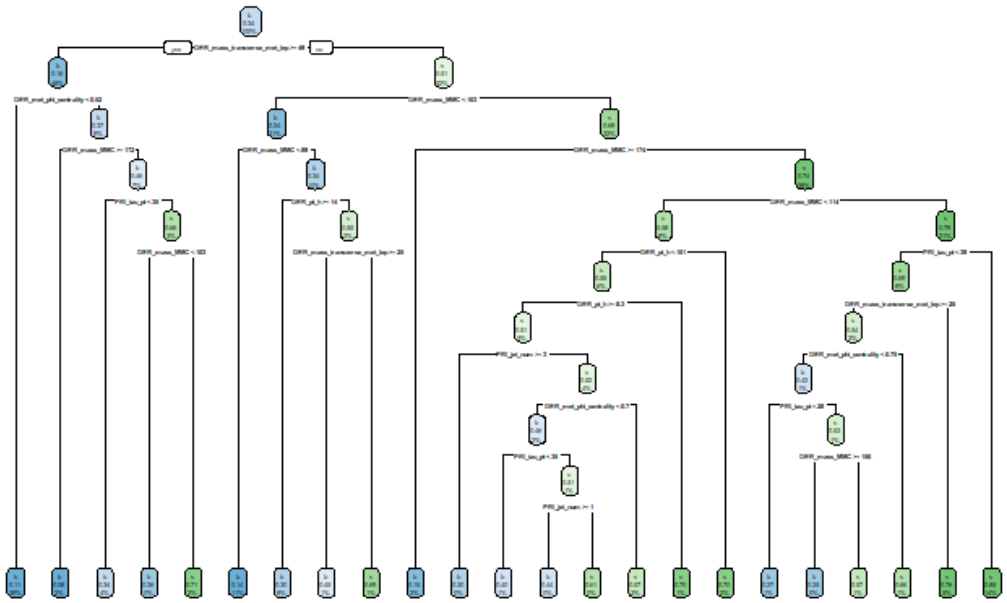


Ilustración 30 - Visualización del modelo

Y con más detalle la importancia de las variables son:

```
Importancia de variables:
```{r}
varImp(rpartModel1)
```
```

	Overall <dbl>
DER_mass_MMC	100.00000000
PRI_tau_pt	55.14329253
DER_mass_transverse_met_lep	53.98239482
DER_met_phi_centrality	36.21687592
DER_pt_ratio_lep_tau	35.69482470
DER_sum_pt	15.09639967
DER_pt_h	9.61445663
PRI_jet_all_pt	9.20197927
PRI_jet_num	2.99560026
PRI_lep_pt	0.09380227

Ilustración 31 – Importancia de las variables

4.4 LogitBoost

4.5 Entrenamiento

En este caso, prácticamente la única diferencia que hemos realizado respecto al caso anterior es utilizar *"LogitBoost"* como método.

```r
```{r}
svm_grid <- expand.grid(.nIter = 5)
svm_control <- trainControl(method = "repeatedcv", number = 10,
                            repeats = 5)
svm_model <- train(Label ~ .,data = train,
                   method = "LogitBoost",
                   metric = "ROC",
                   trControl = svm_control,
                   tuneGrid = svm_grid)
```
```

Ilustración 32 – Entrenamiento con *"LogitBoost"*

5 DISCUSIÓN DE RESULTADOS

5.1 RPART

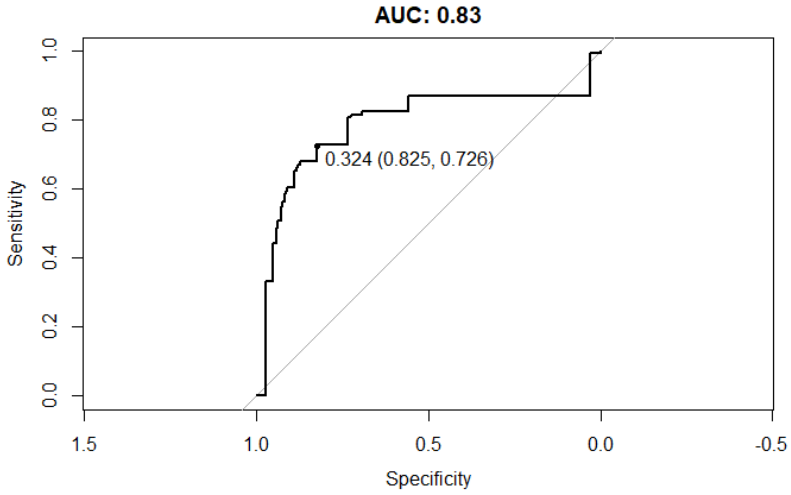


Ilustración 33 - AUC de RPART

5.2 LogitBoost

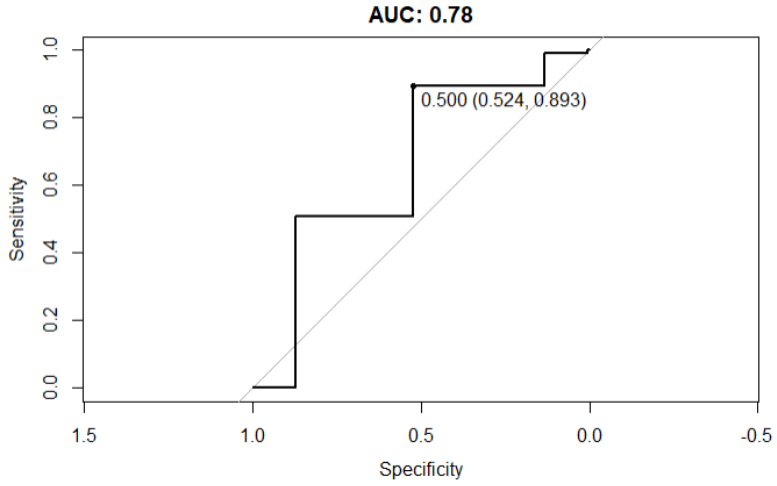


Ilustración 34 - AUC de RPART

6 CONCLUSIONES

Como puede verse en los gráficos de arriba finalmente el uso de *“RPART”* muestra un desempeño mejor y más lineal en comparación del mostrado por *“LOGITBOOST”*.

Con esta practica he aprendido la importancia que tiene la aplicación de tareas de procesamiento antes de aplicar los algoritmos clasificadores. Sobre todo, me ha resultado bastante interesante ya que no había visto antes las tareas de eliminación de columnas a partir de correlaciones entre variables.

También he aprendido la importancia que tiene no eliminar a veces las columnas por su porcentaje de valores perdidos si no que existen metodología muy interesante que no había visto antes como la imputación de valores perdidos.

Sin embargo, reconozco que puede que el pre-procesamiento que he realizado puede que no haya sido el mejor, ya que por ejemplo mi equipo no es capaz de ejecutar entrenamientos como el de *“Random Forest”*