

Pre-procesamiento y clasificación binaria

José Antonio Córdoba Gómez

4/6/2021

Contents

1	Introducción	1
2	Obtención de datos	2
3	Análisis exploratorio	2
3.1	Naturaleza de los datos	2
3.2	¿Balanceamiento o desbalanceamiento?	6
3.3	Correlaciones	7
3.4	Detección de Outliers	8
4	Preprocesamiento	17
4.1	Tratando valores perdidos	17
4.2	Tratamiento de valores nulos	19
4.3	Discretización	19
4.4	Normalización	19
4.5	Análisis de Correlaciones	20
5	Clasificación	21
5.1	Introducción	21
5.2	Random Forest	22
5.3	Árboles de decisión	24
5.4	KNN	26
5.5	Redes Neuronales Artificiales	28
5.6	Comparativa	42
6	Balanceamiento	42
6.1	Random Forest	44
6.2	KNN	45
6.3	Redes Neuronales Artificiales	47
6.4	Comparativa	61
7	Conclusiones	61
8	Referencias	62

1 Introducción

Durante el transcurso de este ejercicio práctico vamos a tratar de realizar tareas de pre-procesamiento vistas en clase sobre datos del experimento **ATLAS** del *CERN-LHC* donde se pretendía identificar de forma experimental la partícula del bosón de *Higgs*. El problema por tanto, consiste en poder clasificar de forma

binaria si una lectura del experimento se correspondía con el decaimiento de un **bosón de Higgs** (objeto esperado) o **ruido de fondo**.

Además, se tratará de aplicar sobre estos datos algunos algoritmos de aprendizaje automático con ayuda de la librería *caret* y realizaremos una pequeña comparación con los resultados que obtengamos.

Para concluir, se expondrán las conclusiones que hemos elaborado durante el transcurso del ejercicio práctico.

2 Obtención de datos

El primer paso es obtener el conjunto de datos, para ello, vamos a basarnos en las funciones de la plantilla del ejercicio que encontramos en el repositorio de la asignatura.

```
downloadAtlasData <- function (){  
  if(!file.exists("data/training.csv") || !file.exists("data/test.csv")) {  
    writeLines(sprintf("descargando {training.csv, test.csv} en: \n%s/data", getwd()))  
    url_datos <- "http://sl.ugr.es/higgs_sige"  
    GET(url_datos, write_disk(temp <- tempfile(fileext = ".zip")))  
    unzip(temp, exdir = "data")  
    unlink(temp)  
  }else{  
    writeLines(sprintf("training y test ya se encuentran en : \n%s/data", getwd()))  
  }  
}  
downloadAtlasData()  
  
## training y test ya se encuentran en :  
## /Users/pp/Desktop/Desktop/practica1/sige2021/prácticas/p1/data
```

Una vez descargados los datos sobre el directorio *workingDirectory/data/* necesitamos leer el conjunto de entrenamiento en una variable de R para poder explorar y preprocesar los datos.

```
atlas_training_raw <- read_csv("data/training.csv")
```

Sabemos previamente que existen valores perdidos en el conjunto de datos que se han condificado con el valor **-999.0**, por lo que vamos a tratar de transformar estos valores perdidos al tipo nativo de R para esta semántica, que es **NA**, antes de realizar el análisis exploratorio del conjunto de datos.

```
atlas_training_raw <- atlas_training_raw %>%  
  na_if(-999.0)
```

3 Análisis exploratorio

3.1 Naturaleza de los datos

Vamos a comenzar realizando un resumen básico del conjunto de datos, que nos indicará valores mínimos, medios y máximos para cada una de las dimensiones, así como la distribución en los cuartiles y el número de valores perdidos (NAs) que ha identificado. Si no hubieramos recodificado en el paso anterior, este resumen sería irreal.

```
summary(atlas_training_raw)  
  
##      EventId          DER_mass_MMC      DER_mass_transverse_met_lep  
##  Min.   :100000   Min.   :  9.04   Min.   : 0.00  
##  1st Qu.:162500   1st Qu.: 91.89   1st Qu.:19.24  
##  Median :225000   Median :112.41   Median :46.52  
##  Mean   :225000   Mean   :121.86   Mean   :49.24
```

```

## 3rd Qu.:287499 3rd Qu.: 135.48 3rd Qu.: 73.60
## Max. :349999  Max. :1192.03  Max. :690.08
## NA's :38114
## DER_mass_vis DER_pt_h DER_deltaeta_jet_jet DER_mass_jet_jet
## Min. : 6.329 Min. : 0.00 Min. :0.00 Min. : 13.6
## 1st Qu.: 59.389 1st Qu.: 14.07 1st Qu.:0.88 1st Qu.: 112.0
## Median : 73.752 Median : 38.47 Median :2.11 Median : 225.9
## Mean : 81.182 Mean : 57.90 Mean :2.40 Mean : 371.8
## 3rd Qu.: 92.259 3rd Qu.: 79.17 3rd Qu.:3.69 3rd Qu.: 478.2
## Max. :1349.351 Max. :2835.00 Max. :8.50 Max. :4975.0
## NA's :177457 NA's :177457
## DER_prodeta_jet_jet DER_deltar_tau_lep DER_pt_tot DER_sum_pt
## Min. :-18.07 Min. :0.208 Min. : 0.000 Min. : 46.10
## 1st Qu.: -2.63 1st Qu.:1.810 1st Qu.: 2.841 1st Qu.: 77.55
## Median : -0.24 Median :2.491 Median : 12.316 Median : 120.66
## Mean : -0.82 Mean :2.373 Mean : 18.917 Mean : 158.43
## 3rd Qu.: 0.96 3rd Qu.:2.961 3rd Qu.: 27.591 3rd Qu.: 200.48
## Max. : 16.69 Max. :5.684 Max. :2834.999 Max. :1852.46
## NA's :177457
## DER_pt_ratio_lep_tau DER_met_phi_centrality DER_lep_eta_centrality
## Min. : 0.047 Min. :-1.4140 Min. : 0.00
## 1st Qu.: 0.883 1st Qu.: -1.3710 1st Qu.: 0.00
## Median : 1.280 Median : -0.3560 Median : 0.45
## Mean : 1.438 Mean : -0.1283 Mean : 0.46
## 3rd Qu.: 1.777 3rd Qu.: 1.2250 3rd Qu.: 0.88
## Max. :19.773 Max. : 1.4140 Max. : 1.00
## NA's :177457
## PRI_tau_pt PRI_tau_eta PRI_tau_phi PRI_lep_pt
## Min. : 20.00 Min. : -2.49900 Min. : -3.142000 Min. : 26.00
## 1st Qu.: 24.59 1st Qu.: -0.92500 1st Qu.: -1.575000 1st Qu.: 32.38
## Median : 31.80 Median : -0.02300 Median : -0.033000 Median : 40.52
## Mean : 38.71 Mean : -0.01097 Mean : -0.008171 Mean : 46.66
## 3rd Qu.: 45.02 3rd Qu.: 0.89800 3rd Qu.: 1.565000 3rd Qu.: 53.39
## Max. :764.41 Max. : 2.49700 Max. : 3.142000 Max. :560.27
##
## PRI_lep_eta PRI_lep_phi PRI_met PRI_met_phi
## Min. : -2.50500 Min. : -3.14200 Min. : 0.109 Min. : -3.14200
## 1st Qu.: -1.01400 1st Qu.: -1.52200 1st Qu.: 21.398 1st Qu.: -1.57500
## Median : -0.04500 Median : 0.08600 Median : 34.802 Median : -0.02400
## Mean : -0.01951 Mean : 0.04354 Mean : 41.717 Mean : -0.01012
## 3rd Qu.: 0.95900 3rd Qu.: 1.61800 3rd Qu.: 51.895 3rd Qu.: 1.56100
## Max. : 2.50300 Max. : 3.14200 Max. : 2842.617 Max. : 3.14200
##
## PRI_met_sumet PRI_jet_num PRI_jet_leading_pt PRI_jet_leading_eta
## Min. : 13.68 Min. : 0.0000 Min. : 30.00 Min. : -4.50
## 1st Qu.: 123.02 1st Qu.: 0.0000 1st Qu.: 44.42 1st Qu.: -1.34
## Median : 179.74 Median : 1.0000 Median : 65.56 Median : 0.00
## Mean : 209.80 Mean : 0.9792 Mean : 84.82 Mean : 0.00
## 3rd Qu.: 263.38 3rd Qu.: 2.0000 3rd Qu.: 103.34 3rd Qu.: 1.34
## Max. :2003.98 Max. : 3.0000 Max. :1120.57 Max. : 4.50
## NA's :99913 NA's :99913
## PRI_jet_leading_phi PRI_jet_subleading_pt PRI_jet_subleading_eta
## Min. : -3.14 Min. : 30.00 Min. : -4.50
## 1st Qu.: -1.58 1st Qu.: 37.31 1st Qu.: -1.61

```

```

## Median :-0.03      Median : 47.90      Median :-0.01
## Mean   :-0.01      Mean   : 57.68      Mean   :-0.01
## 3rd Qu.: 1.56      3rd Qu.: 66.64      3rd Qu.: 1.59
## Max.   : 3.14      Max.   :721.46      Max.   : 4.50
## NA's    :99913      NA's    :177457      NA's    :177457
## PRI_jet_subleading_phi PRI_jet_all_pt      Weight          Label
## Min.   :-3.14      Min.   : 0.00      Min.   :0.001502  Length:250000
## 1st Qu.:-1.58      1st Qu.: 0.00      1st Qu.:0.018636  Class  :character
## Median : 0.00      Median : 40.51      Median :1.156188  Mode   :character
## Mean   : 0.00      Mean   : 73.06      Mean   :1.646767
## 3rd Qu.: 1.58      3rd Qu.: 109.93     3rd Qu.:2.404128
## Max.   : 3.14      Max.   :1633.43      Max.   :7.822543
## NA's    :177457

```

Tras este resumen inicial, podemos observar como existen dimensiones con una gran cantidad (que no proporción, aunque la podemos calcular) de valores perdidos, como son:

1. DER_mass_MMC
2. DER_deltaeta_jet_jet
3. DER_prodeta_jet_jet
4. DER_lep_eta_centrality
5. PRI_jet_leading_pt
6. PRI_jet_leading_eta
7. PRI_jet_leading_phi
8. PRI_jet_subleading_pt
9. PRI_jet_subleading_eta
10. PRI_jet_subleading_phi

Es decir, cerca de un tercio de las dimensiones presentan una alta cantidad de valores perdidos, y de estos, la mayoría son valores primitivos sobre *jet*.

Además, podemos observar que todas las dimensiones son numéricas, exceptuando la dimensión de clase *Label*, que es de tipo carácter.

A continuación vamos a tratar de observar más información sobre la naturaleza de los datos.

```
df_status(atlas_training_raw)
```

```

##                   variable q_zeros p_zeros  q_na p_na q_inf p_inf
## 1                  EventId      0  0.00      0  0.00      0      0
## 2             DER_mass_MMC      0  0.00 38114 15.25      0      0
## 3  DER_mass_transverse_met_lep      3  0.00      0  0.00      0      0
## 4             DER_mass_vis      0  0.00      0  0.00      0      0
## 5                 DER_pt_h     41  0.02      0  0.00      0      0
## 6  DER_deltaeta_jet_jet      6  0.00 177457 70.98      0      0
## 7  DER_mass_jet_jet      0  0.00 177457 70.98      0      0
## 8  DER_prodeta_jet_jet     58  0.02 177457 70.98      0      0
## 9  DER_deltar_tau_lep      0  0.00      0  0.00      0      0
## 10  DER_pt_tot            39  0.02      0  0.00      0      0
## 11  DER_sum_pt            0  0.00      0  0.00      0      0
## 12  DER_pt_ratio_lep_tau      0  0.00      0  0.00      0      0
## 13  DER_met_phi_centrality     53  0.02      0  0.00      0      0
## 14  DER_lep_eta_centrality 15752  6.30 177457 70.98      0      0
## 15          PRI_tau_pt      0  0.00      0  0.00      0      0
## 16          PRI_tau_eta      0  0.00      0  0.00      0      0
## 17          PRI_tau_phi     32  0.01      0  0.00      0      0

```

```

## 18          PRI_lep_pt      0    0.00      0  0.00      0    0
## 19          PRI_lep_eta    35   0.01      0  0.00      0    0
## 20          PRI_lep_phi    33   0.01      0  0.00      0    0
## 21          PRI_met         0    0.00      0  0.00      0    0
## 22          PRI_met_phi    44   0.02      0  0.00      0    0
## 23          PRI_met_sumet   0    0.00      0  0.00      0    0
## 24          PRI_jet_num     99913 39.97      0  0.00      0    0
## 25          PRI_jet_leading_pt 0    0.00  99913 39.97      0    0
## 26          PRI_jet_leading_eta 26   0.01  99913 39.97      0    0
## 27          PRI_jet_leading_phi 19   0.01  99913 39.97      0    0
## 28          PRI_jet_subleading_pt 0    0.00 177457 70.98      0    0
## 29          PRI_jet_subleading_eta 9    0.00 177457 70.98      0    0
## 30          PRI_jet_subleading_phi 10   0.00 177457 70.98      0    0
## 31          PRI_jet_all_pt    99913 39.97      0  0.00      0    0
## 32          Weight          0    0.00      0  0.00      0    0
## 33          Label            0    0.00      0  0.00      0    0
##           type unique
## 1  numeric 250000
## 2  numeric 108337
## 3  numeric 101637
## 4  numeric 100558
## 5  numeric 115563
## 6  numeric 7086
## 7  numeric 68365
## 8  numeric 16592
## 9  numeric 4692
## 10 numeric 59042
## 11 numeric 156098
## 12 numeric 5931
## 13 numeric 2829
## 14 numeric 1001
## 15 numeric 59639
## 16 numeric 4971
## 17 numeric 6285
## 18 numeric 61929
## 19 numeric 4987
## 20 numeric 6285
## 21 numeric 87836
## 22 numeric 6285
## 23 numeric 179740
## 24 numeric      4
## 25 numeric 86589
## 26 numeric 8557
## 27 numeric 6284
## 28 numeric 42463
## 29 numeric 8627
## 30 numeric 6285
## 31 numeric 103559
## 32 numeric 104096
## 33 character      2

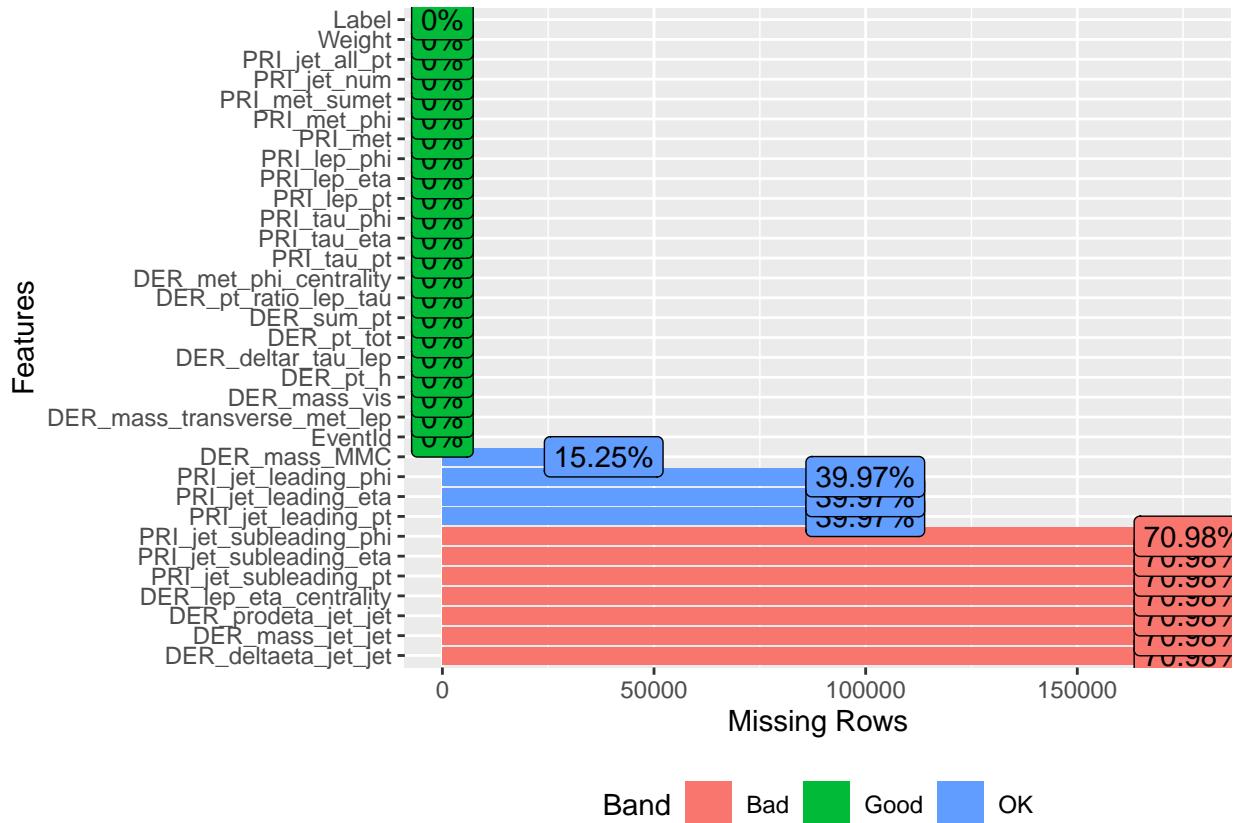
```

Pobservar como las dimensiones *PRI_jet_num* y *PRI_jet_all_pt* tienen una proporción de valores nulos del 40%.

Vamos a tratar de estudiar a continuación la proporción de valores nulos, para ello vamos a mostrar un de

forma gráfica cuál es esa proporción.

```
plot_missing(atlas_training_raw)
```



Podemos observar que la proporción de valores perdidos (p_{na}) sobre las dimensiones

1. DER_deltaeta_jet_jet
2. DER_mass_jet_jet
3. DER_proleta_jet_jet
4. DER_prodelta_jet_jet
5. DER_lep_eta_centrality
6. PRI_jet_subleading_pt
7. PRI_jet_subleading_eta
8. PRI_jet_subleading_phi

es superior al 70% de los datos, una tasa altísima.

Por otra parte, las dimensiones 1. PRI_jet_leading_pt 2. PRI_jet_all_pt encuentran una proporción de valores perdidos cercana al 40%, no tan alta como la lista de dimensiones anterior, pero bastante notable.

Si aplicamos una eliminación de estos valores perdidos y nulos, podríamos ver el conjunto de datos gravemente reducido, con lo que la capacidad de generalización posterior se vería resentida.

3.2 ¿Balanceamiento o desbalanceamiento?

Durante esta sección vamos a tratar de observar si el conjunto de datos tiene una proporción balanceada o desbalanceada entre sus clases (entre si es el decaimiento del bosón o ruido).

En caso de que nos encontremos en una situación de desbalanceamiento, es decir, que una de las clases sea

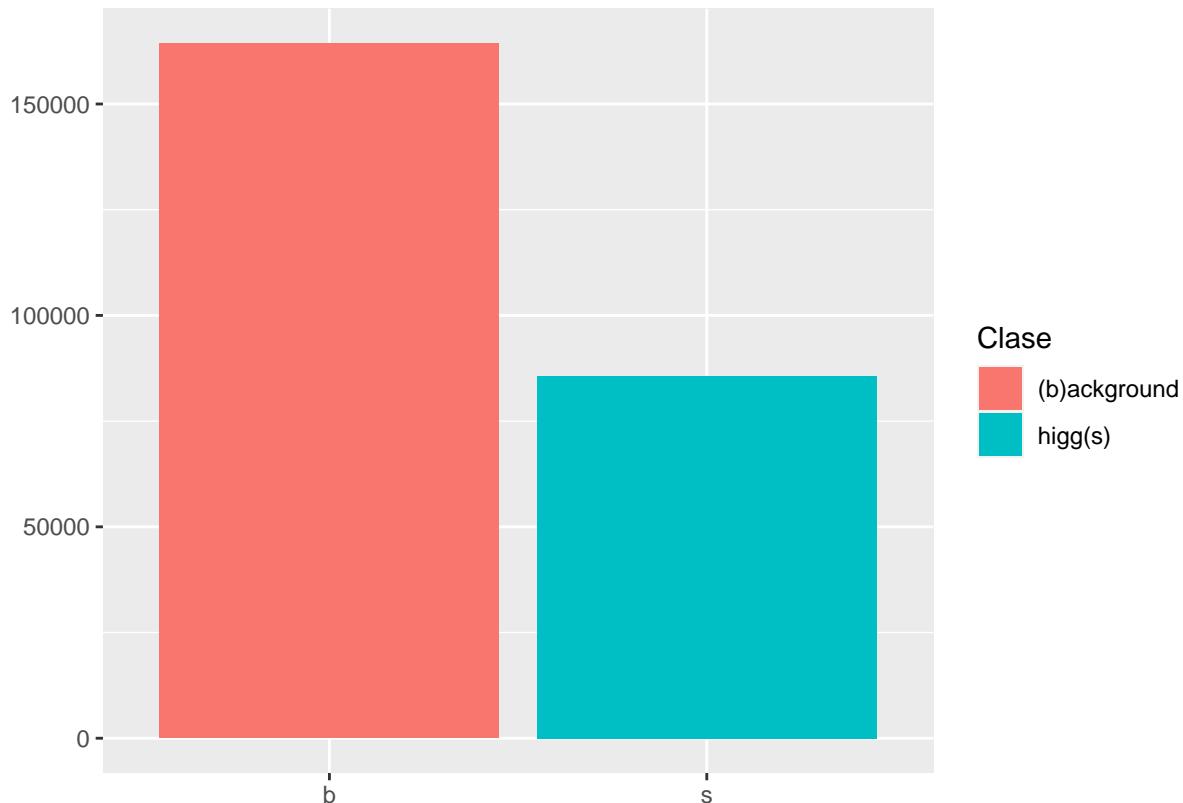
minoritaria, el proceso de aprendizaje que realicemos más tarde, se puede ver afectado, ya que en el proceso de generalización, la clase minoritaria se verá perjudicada.

```
table(atlas_training_raw$Label)
```

```
##  
##      b      s  
## 164333  85667
```

Con la proporción numérica se puede observar correctamente, como por desgracia, las clases están desbalanceadas. Vamos a mostrarlo de forma gráfica y seguir el conocimiento popular de que *una imagen vale más que mil palabras* para comprender de un vistazo este desbalanceamiento.

```
ggplot(atlas_training_raw) +  
  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)"))  
)
```



Este hecho puede implicar que tengamos que plantear alguna estrategia para manejar datos desbalanceados durante el proceso de clasificación automática.

3.3 Correlaciones

De forma inicial podemos echar un vistazo sobre las correlaciones que existen entre las diferentes variables.

```
plot_correlation(atlas_training_raw)
```

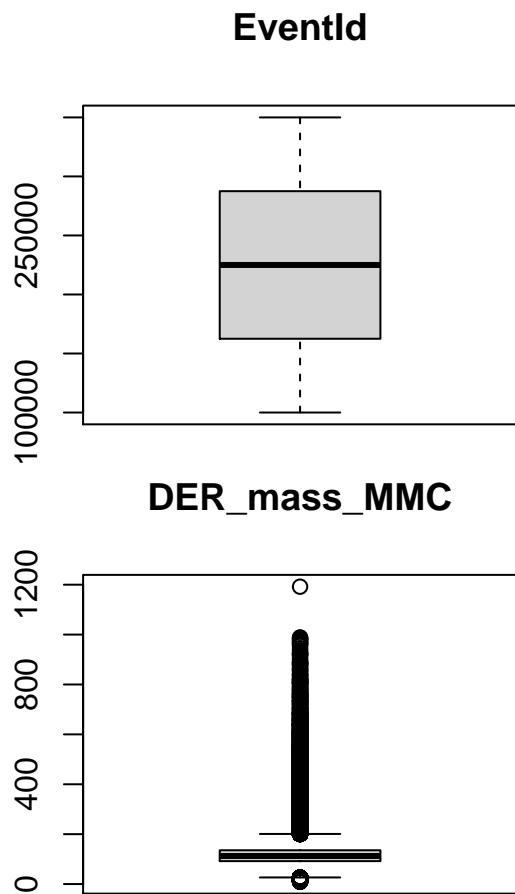
El gráfico anterior es difícil de estudiar debido al alto número de dimensiones que existen en el conjunto de datos, pero de un primer vistazo, podemos ver que existen algunas dimensiones con una alta correlación

tanto directa como indirecta, como puede ser $DER_pt_ratio_lep_tau$ y $DER_mass_transverse_met_lep$ o PRI_{tau}_pt , $DER_{met}_phi_{centrality}$ y DER_{pt}_h .

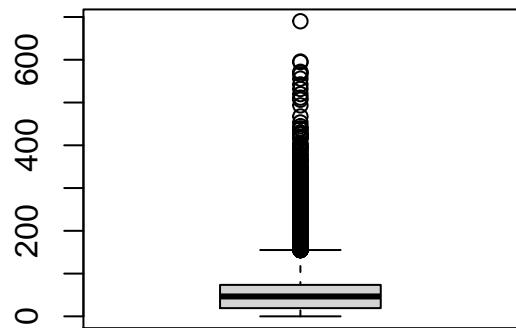
3.4 Detección de Outliers

Dentro del conjunto de datos pueden existir observaciones que sean representativas de la población, pudiendo así, distorsionar el resultado del aprendizaje. Por tanto, vamos a tratar de echar un vistazo a estos valores atípicos dentro de nuestro conjunto de datos.

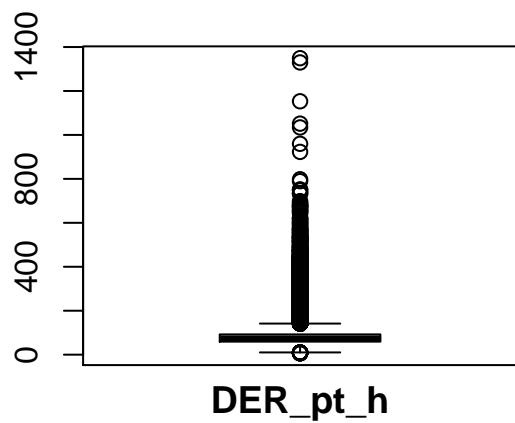
```
for (i in 1:length(atlas_training_raw)) {  
  if(i!=33)  
    boxplot(atlas_training_raw[,i],  
            main=names(atlas_training_raw[i]),  
            type="l")  
}
```



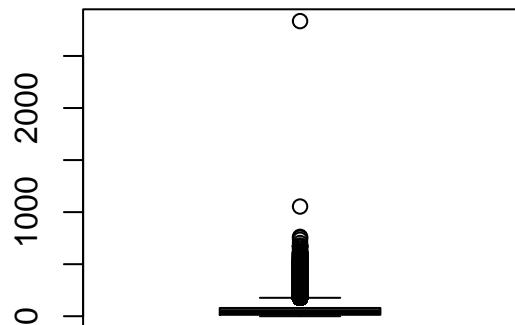
DER_mass_transverse_met_lep



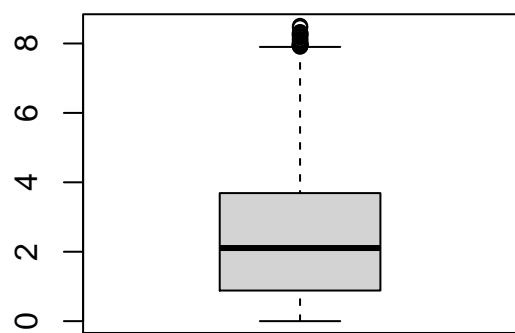
DER_mass_vis



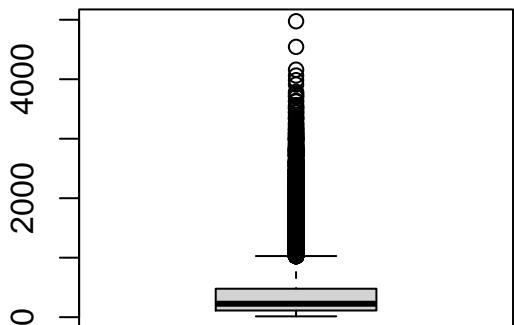
DER_pt_h



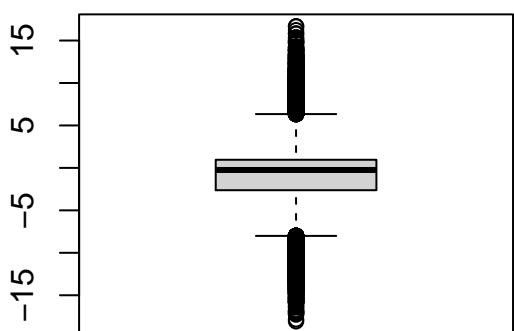
DER_deltaeta_jet_jet



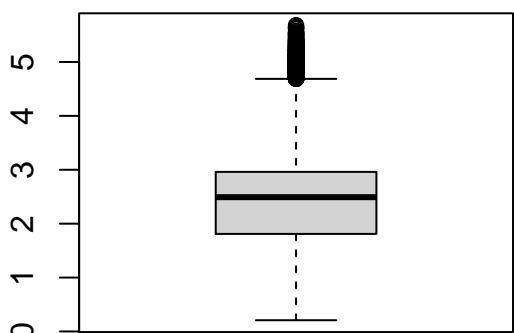
DER_mass_jet_jet



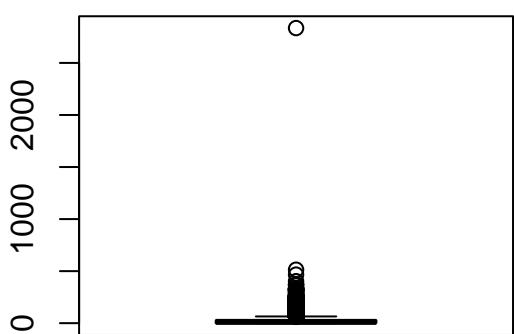
DER_prodeta_jet_jet



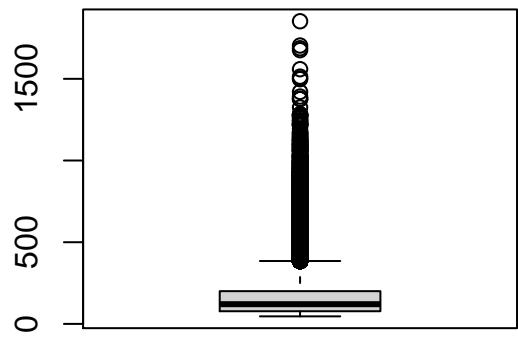
DER_deltar_tau_lep



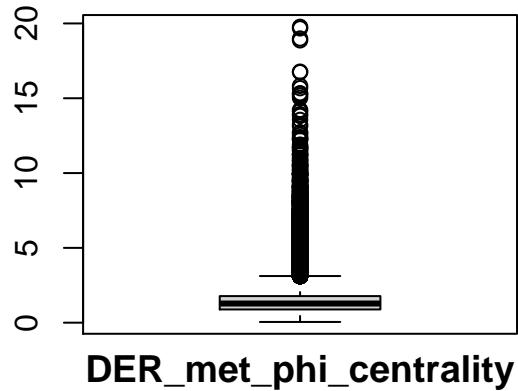
DER_pt_tot



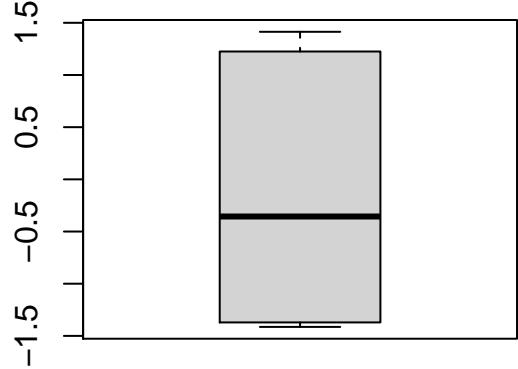
DER_sum_pt



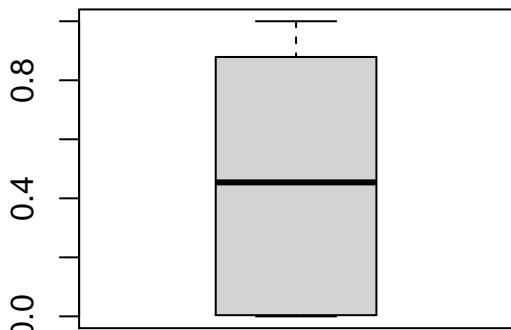
DER_pt_ratio_lep_tau



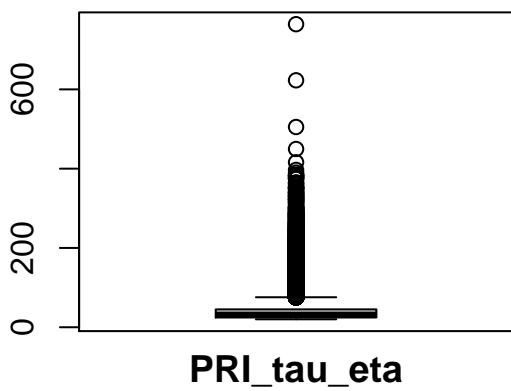
DER_met_phi_centrality



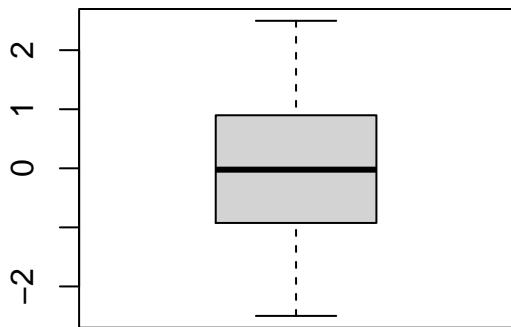
DER_lep_eta_centrality



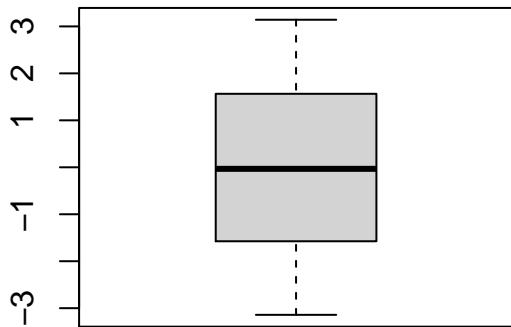
PRI_tau_pt



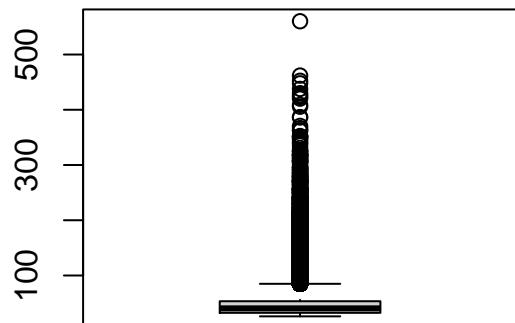
PRI_tau_eta



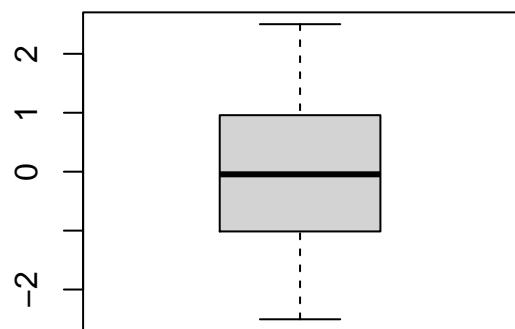
PRI_tau_phi



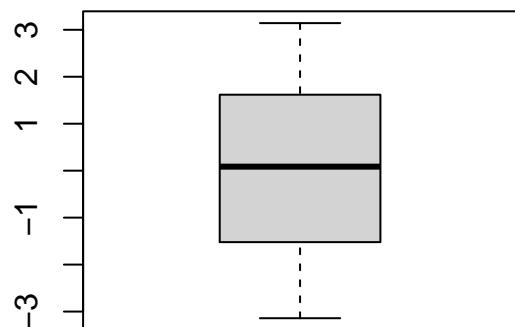
PRI_lep_pt



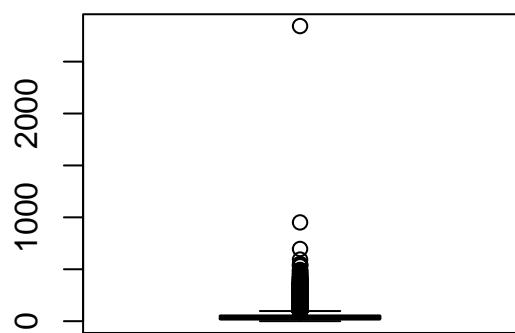
PRI_lep_eta



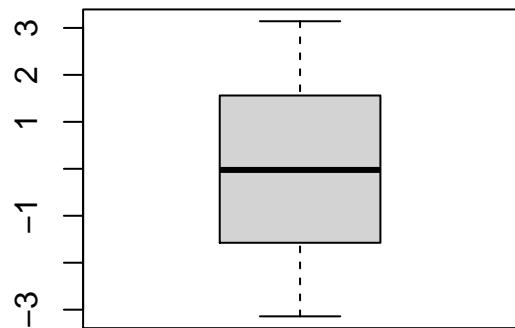
PRI_lep_phi



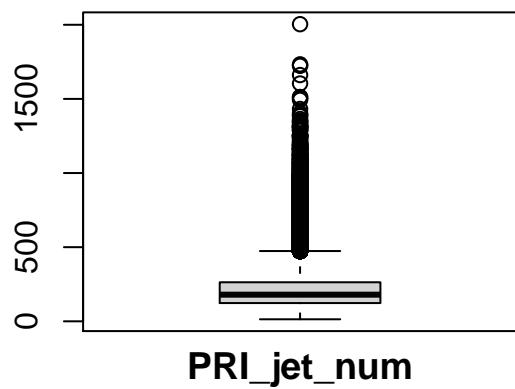
PRI_met



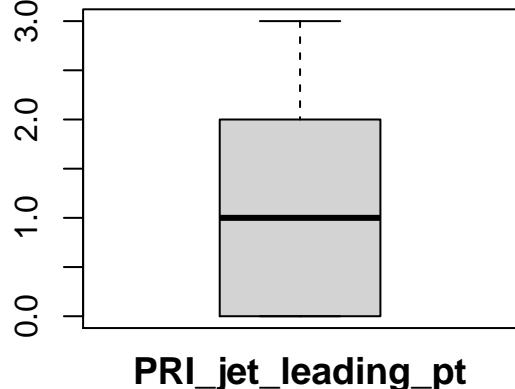
PRI_met_phi



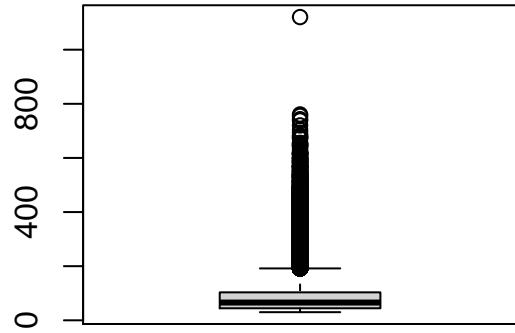
PRI_met_sumet



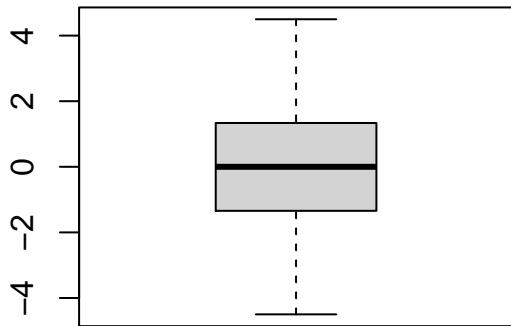
PRI_jet_num



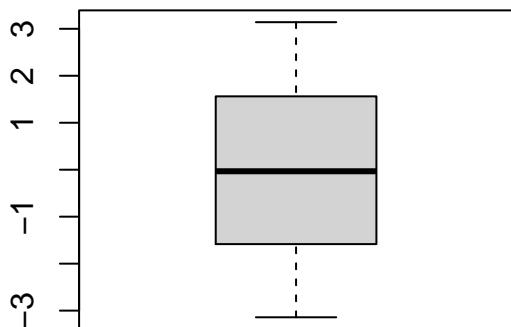
PRI_jet_leading_pt



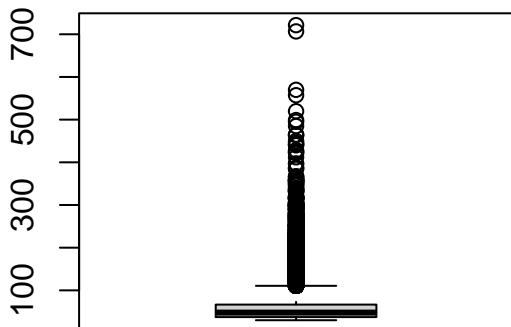
PRI_jet_leading_eta



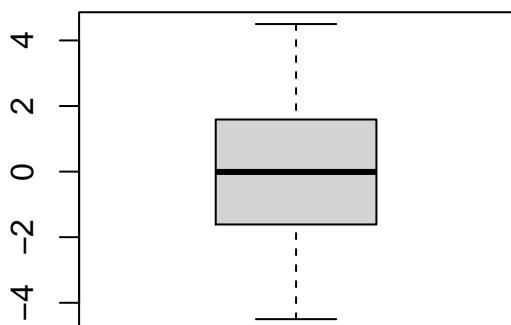
PRI_jet_leading_phi



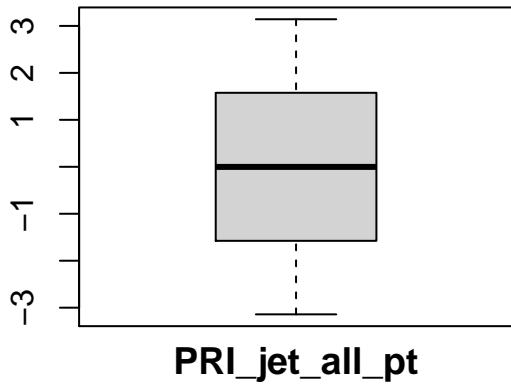
PRI_jet_subleading_pt



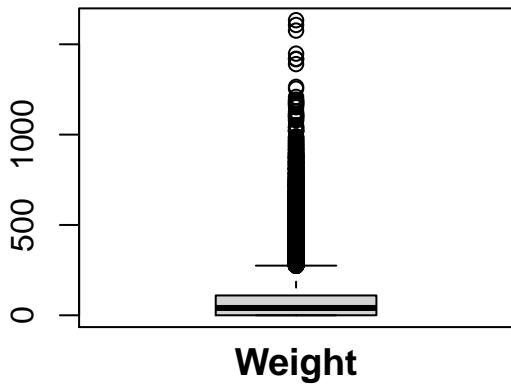
PRI_jet_subleading_eta



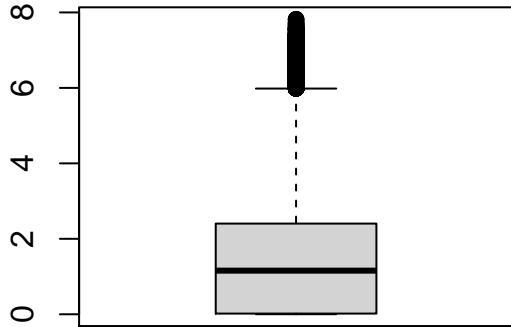
PRI_jet_subleading_phi



PRI_jet_all_pt



Weight



Podemos observar que dimensiones como:

1. DER_mass_MMC
2. DER_mass_transverse_met_lep
3. DER_mass_vis
4. DER_pt_h
5. DER_mass_jet_jet
6. DER_proleta_jet_jet
7. DER_pt_tot
8. DER_sum_pt
9. DER_pt_ratio_lep_tau
10. PRI_tau_pt
11. PRI_lep_pt
12. PRI_met

13. PRI_met_sumet
14. PRI_jet_leading_pt
15. PRI_jet_subleading_pt
16. PRI_jet_all_pt

muestran a primera vista que contienen una gran proporción de outliers. Como son muchísimas dimensiones que presentan esta problemática, una sustracción de estos valores atípicos nos puede causar una disminución drástica del conjunto de observaciones, viendose de nuevo, afectado gravemente el proceso de generalización.

4 Preprocesamiento

4.1 Tratando valores perdidos

Anteriormente ya realizamos una recodificación de los valores perdidos, pero no decidimos qué realizar con ellos. Comentamos que una eliminación de éstos, puede implicar un empeoramiento notorio en el proceso de generalización. Otra opción sería realizar una sustitución de estos valores perdidos por otros valores como la mediana o la media. Esta segunda opción tampoco está exenta de problemática, ya que adulteraría el conjunto de datos, por valores que pueden no tener nada que ver y que también pueden influir en el conjunto de datos.

Por lo tanto, vamos a decidir eliminar los valores perdidos, ya que considero mejor perder datos que adulterarlos. Esto es debido a que como son datos experimentales, podríamos realizar una captura mayor de datos y minimizar este posible impacto, mientras que la segunda opción podría alterar de forma irremediable el resultado final de la generalización.

Como vimos anteriormente en el análisis exploratorio de los datos, existían dimensiones con un 70% de los valores perdidos, por lo que vamos a eliminarlos, tal y como hemos dicho anteriormente. Para ello vamos a introducir en una columna todas las dimensiones que cumplan el filtro de que la proporción de valores perdidos superen el 70%.

```
dimensiones_nas <- df_status(atlas_training_raw) %>%
  filter(p_na > 70) %>%
  select(variable)
```

	variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf
## 1	EventId	0	0.00	0	0.00	0	0
## 2	DER_mass_MMC	0	0.00	38114	15.25	0	0
## 3	DER_mass_transverse_met_lep	3	0.00	0	0.00	0	0
## 4	DER_mass_vis	0	0.00	0	0.00	0	0
## 5	DER_pt_h	41	0.02	0	0.00	0	0
## 6	DER_deltaeta_jet_jet	6	0.00	177457	70.98	0	0
## 7	DER_mass_jet_jet	0	0.00	177457	70.98	0	0
## 8	DER_prodeta_jet_jet	58	0.02	177457	70.98	0	0
## 9	DER_deltar_tau_lep	0	0.00	0	0.00	0	0
## 10	DER_pt_tot	39	0.02	0	0.00	0	0
## 11	DER_sum_pt	0	0.00	0	0.00	0	0
## 12	DER_pt_ratio_lep_tau	0	0.00	0	0.00	0	0
## 13	DER_met_phi_centrality	53	0.02	0	0.00	0	0
## 14	DER_lep_eta_centrality	15752	6.30	177457	70.98	0	0
## 15	PRI_tau_pt	0	0.00	0	0.00	0	0
## 16	PRI_tau_eta	0	0.00	0	0.00	0	0
## 17	PRI_tau_phi	32	0.01	0	0.00	0	0
## 18	PRI_lep_pt	0	0.00	0	0.00	0	0
## 19	PRI_lep_eta	35	0.01	0	0.00	0	0
## 20	PRI_lep_phi	33	0.01	0	0.00	0	0
## 21	PRI_met	0	0.00	0	0.00	0	0
## 22	PRI_met_phi	44	0.02	0	0.00	0	0

```

## 23          PRI_met_sumet      0    0.00      0  0.00      0    0
## 24          PRI_jet_num     99913  39.97      0  0.00      0    0
## 25          PRI_jet_leading_pt   0    0.00  99913 39.97      0    0
## 26          PRI_jet_leading_eta   26   0.01  99913 39.97      0    0
## 27          PRI_jet_leading_phi   19   0.01  99913 39.97      0    0
## 28          PRI_jet_subleading_pt   0    0.00 177457 70.98      0    0
## 29          PRI_jet_subleading_eta   9    0.00 177457 70.98      0    0
## 30          PRI_jet_subleading_phi   10   0.00 177457 70.98      0    0
## 31          PRI_jet_all_pt     99913  39.97      0  0.00      0    0
## 32          Weight            0    0.00      0  0.00      0    0
## 33          Label             0    0.00      0  0.00      0    0
##           type unique
## 1    numeric 250000
## 2    numeric 108337
## 3    numeric 101637
## 4    numeric 100558
## 5    numeric 115563
## 6    numeric 7086
## 7    numeric 68365
## 8    numeric 16592
## 9    numeric 4692
## 10   numeric 59042
## 11   numeric 156098
## 12   numeric 5931
## 13   numeric 2829
## 14   numeric 1001
## 15   numeric 59639
## 16   numeric 4971
## 17   numeric 6285
## 18   numeric 61929
## 19   numeric 4987
## 20   numeric 6285
## 21   numeric 87836
## 22   numeric 6285
## 23   numeric 179740
## 24   numeric 4
## 25   numeric 86589
## 26   numeric 8557
## 27   numeric 6284
## 28   numeric 42463
## 29   numeric 8627
## 30   numeric 6285
## 31   numeric 103559
## 32   numeric 104096
## 33 character      2

```

Por tanto, aplicamos la reducción de dimensionalidad y creamos un conjunto de datos nuevo llamado *atlas_training* sobre el que vamos a realizar posteriormente las tareas de generalización.

```

atlas_training <- atlas_training_raw %>%
  select(-one_of(dimisiones_nas$variable))

dimensions <- data.frame(ncol(atlas_training_raw), ncol(atlas_training))
dimensions

##  ncol.atlas_training_raw. ncol.atlas_training.

```

Como podemos apreciar, el conjunto de datos ha visto su dimensionalidad reducida en 7 dimensiones al haber eliminado las columnas:

```
dimensiones_nas
```

```
##           variable
## 1   DER_deltaeta_jet_jet
## 2   DER_mass_jet_jet
## 3   DER_proleta_jet_jet
## 4 DER_lep_eta_centrality
## 5   PRI_jet_subleading_pt
## 6   PRI_jet_subleading_eta
## 7   PRI_jet_subleading_phi
```

Ahora vamos a realizar la eliminación de las instancias correspondientes a los valores perdidos

```
atlas_training <- na.omit(atlas_training)
valores_perdidos <- data.frame(nrow(atlas_training_raw), nrow(atlas_training))
valores_perdidos
```

```
##   nrow.atlas_training_raw. nrow.atlas_training.
## 1                  250000          138096
```

Podemos observar que hemos reducido el conjunto de datos de forma drástica y hemos obtenido cerca del 55% de los datos originales. Vamos a ser consecuentes con la decisión tomada anteriormente y ya veremos qué ocurre en el proceso de generalización por haber tomado esta decisión.

4.2 Tratamiento de valores nulos

También podríamos realizar un tratamiento de las dimensiones que contengan una alta proporción de datos nulos, pero en este caso, en la exploración previa no detectamos una alta proporción de valores nulos, por lo que no vamos a reducir aún más, el conjunto de datos.

Tampoco vamos a reducir el conjunto de datos por la proporción de valores únicos ya que el conjunto de datos no presenta gran proporción de los mismo.

4.3 Discretización

Realizar una discretización sobre el conjunto de datos que tenemos debería estar respaldado por un conocimiento extenso del dominio del problema, ya que en caso contrario, tomar decisiones de agrupación en categorías de ciertas variables podría carecer de semántica en este conjunto de datos y perjudicar seriamente al proceso de aprendizaje. Es por este motivo que descartamos el proceso de discretización de variables.

4.4 Normalización

Tal y como pudimos observar en la fase de análisis exploratorio, el conjunto de datos contiene dimensiones que contienen distribuciones muy diferentes entre sí y con valores muy dispersos. Por tanto, esto motiva que realicemos un proceso de normalización de las diferentes variables que podemos encontrar en el conjunto de datos entre valores [0,1].

```
normalizeAtlas <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

for(i in 1:length(atlas_training[-1]))
  atlas_training[,i] <- normalizeAtlas(atlas_training[,i])
```

```

head(atlas_training, n = 10)

## # A tibble: 10 x 26
##   EventId DER_mass_MMC DER_mass_transv~ DER_mass_vis DER_pt_h DER_deltar_tau_~
##   <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 0.        0.109     0.0869    0.0952    0.0266    0.523
## 2 4.00e-6   0.128     0.116     0.101     0.0457    0.598
## 3 2.00e-5   0.0676    0.0228    0.0545    0.110     0.210
## 4 2.40e-5   0.118     0.0486    0.106     0.101     0.500
## 5 2.80e-5   0.123     0.0175    0.0919    0.0277    0.492
## 6 3.60e-5   0.100     0.150     0.0652    0.184     0.255
## 7 4.40e-5   0.0888    0.0173    0.0719    0.0292    0.491
## 8 4.80e-5   0.115     0.108     0.101     0.102     0.361
## 9 6.40e-5   0.0884    0.00732   0.0638    0.0448    0.503
## 10 9.20e-5  0.111     0.00124   0.110     0.165     0.205
## # ... with 20 more variables: DER_pt_tot <dbl>, DER_sum_pt <dbl>,
## #   DER_pt_ratio_lep_tau <dbl>, DER_met_phi_centrality <dbl>, PRI_tau_pt <dbl>,
## #   PRI_tau_eta <dbl>, PRI_tau_phi <dbl>, PRI_lep_pt <dbl>, PRI_lep_eta <dbl>,
## #   PRI_lep_phi <dbl>, PRI_met <dbl>, PRI_met_phi <dbl>, PRI_met_sumet <dbl>,
## #   PRI_jet_num <dbl>, PRI_jet_leading_pt <dbl>, PRI_jet_leading_eta <dbl>,
## #   PRI_jet_leading_phi <dbl>, PRI_jet_all_pt <dbl>, Weight <dbl>, Label <chr>

```

4.5 Análisis de Correlaciones

Como vimos anteriormente, existen algunas dimensiones que presentan correlaciones bastante fuertes, por lo que pueden presentar una alta relevancia a la hora de realizar el aprendizaje y predecir el valor final de la variable de clase.

Vamos a crear un dataframe nuevo que incluya todas las 26 columnas que teníamos hasta el momento menos la columna de clase. Con ello, vamos a seleccionar aquellas dimensiones que presenten, en valor absoluto (para incluir las correlaciones directas e inversas), una correlación superior o igual al 25%

```

atlas_training_cor <- atlas_training[1:length(atlas_training)-1]
cor_mat <- cor(atlas_training_cor)
cor_mat[!lower.tri(cor_mat)] <- NA # remove diagonal and redundant values
columns_i_want <- data.frame( cor_mat ) %>%
  rownames_to_column() %>%
  gather(key="variable", value="correlation", -rowname) %>%
  filter(abs(correlation) >= 0.25) %>%
  filter(rowname != 'Weight')

unique(columns_i_want['rowname'])

##          rowname
## 1      DER_mass_vis
## 2      DER_deltar_tau_lep
## 3      PRI_lep_pt
## 5      DER_pt_ratio_lep_tau
## 6      DER_met_phi_centrality
## 11     DER_sum_pt
## 13     PRI_tau_pt
## 15     PRI_met
## 16     PRI_met_sumet
## 17     PRI_jet_num
## 18     PRI_jet_leading_pt

```

```

## 19          PRI_jet_all_pt
## 43          PRI_lep_eta
## 47          PRI_jet_leading_phi

Como podemos observar, terminaríamos con un total de 14 dimensiones, sin contar la variable de clase, que serían 15.

atlas_training_only_label <- data.frame(atlas_training$Label)

atlas_training <- atlas_training %>%
  select(one_of(columns_i_want$rowname))

unique(columns_i_want$rowname)

## [1] "DER_mass_vis"           "DER_deltar_tau_lep"    "PRI_lep_pt"
## [4] "DER_pt_ratio_lep_tau"   "DER_met_phi_centrality" "DER_sum_pt"
## [7] "PRI_tau_pt"             "PRI_met"                 "PRI_met_sumet"
## [10] "PRI_jet_num"            "PRI_jet_leading_pt"     "PRI_jet_all_pt"
## [13] "PRI_lep_eta"            "PRI_jet_leading_phi"

atlas_training['Label'] <- atlas_training_only_label

```

5 Clasificación

5.1 Introducción

Llegados a este punto del documento tenemos ya los datos procesados con nuestro criterio, por lo que podemos comenzar con la tarea de aprendizaje y generalización. Para ello, tenemos multiples algoritmos de aprendizaje como son:

1. Random Forest
2. Árboles de decisión
3. Redes Neuronales Artificiales

5.1.1 Metodología

Como aprendimos en la asignatura anterior (TID), primero realizaremos una partición del conjunto de datos ya procesados en dos partes. El primer segmento lo denominaremos de entrenamiento y contendrá una proporción del 70% de los datos iniciales, y el segundo segmento, se denominará de test y contendrá el 30% de los datos restantes.

Esta partición se realizará de forma repetida y se la introduciremos al clasificador de forma repetida, disminuyendo la aleatoriedad de la entrada (*Cross Validation*).

A la salida del clasificador tendremos que adjuntar su precisión y valorar la tasa de falsos positivos y verdaderos positivos y estimar si el desbalanceo de las clases ha supuesto un gran problema al clasificador. También será necesario analizar el área que nos deja la curva ROC sobre el conjunto de validación.

Por último será necesario evaluar cuán bueno ha sido este proceso de generalización entregándole como entrada al clasificador el conjunto de test.

5.1.2 Particionamiento de los datos y ‘Cross Validation’

Aplicando la metodología anteriormente expuesta, realizamos una particion 70-30 de los datos.

```

set.seed(1000)

particion      <- createDataPartition(as.factor(atlas_training$Label), p=0.7, list=FALSE)

```

```

entrenamiento <- atlas_training[particion, ]
test         <- atlas_training[-particion, ]

```

Y aplicamos el conjunto de validación cruzada de 5.

```
cv <- trainControl(verboseIter = TRUE, classProbs = TRUE, summaryFunction = twoClassSummary, method = "cv")
```

5.2 Random Forest

```

randomForest <- train(Label ~ ., data = entrenamiento, metric = "ROC", method = "rf", trControl = cv)

## + Fold1: mtry= 2
## - Fold1: mtry= 2
## + Fold1: mtry= 8
## - Fold1: mtry= 8
## + Fold1: mtry=14
## - Fold1: mtry=14
## + Fold2: mtry= 2
## - Fold2: mtry= 2
## + Fold2: mtry= 8
## - Fold2: mtry= 8
## + Fold2: mtry=14
## - Fold2: mtry=14
## + Fold3: mtry= 2
## - Fold3: mtry= 2
## + Fold3: mtry= 8
## - Fold3: mtry= 8
## + Fold3: mtry=14
## - Fold3: mtry=14
## + Fold4: mtry= 2
## - Fold4: mtry= 2
## + Fold4: mtry= 8
## - Fold4: mtry= 8
## + Fold4: mtry=14
## - Fold4: mtry=14
## + Fold5: mtry= 2
## - Fold5: mtry= 2
## + Fold5: mtry= 8
## - Fold5: mtry= 8
## + Fold5: mtry=14
## - Fold5: mtry=14
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 8 on full training set

```

Una vez completado el proceso de aprendizaje, podemos evaluar el desempeño de este modelo sobre el conjunto de evaluación con una matriz de confusión, sabiendo que cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real.

```

randomForestPrediccion <- predict(randomForest, test)
randomForestMatrizConfusion <- confusionMatrix(table(randomForestPrediccion, test[["Label"]]))
randomForestMatrizConfusion

## Confusion Matrix and Statistics
##

```

```

## 
## randomForestPrediccion      b      s
##                               b 19698  4508
##                               s  4061 13161
##
##                               Accuracy : 0.7932
##                               95% CI : (0.7892, 0.7971)
## No Information Rate : 0.5735
## P-Value [Acc > NIR] : < 2.2e-16
##
##                               Kappa : 0.5758
##
## Mcnemar's Test P-Value : 1.45e-06
##
##                               Sensitivity : 0.8291
##                               Specificity : 0.7449
## Pos Pred Value : 0.8138
## Neg Pred Value : 0.7642
## Prevalence : 0.5735
## Detection Rate : 0.4755
## Detection Prevalence : 0.5843
## Balanced Accuracy : 0.7870
##
## 'Positive' Class : b
##

```

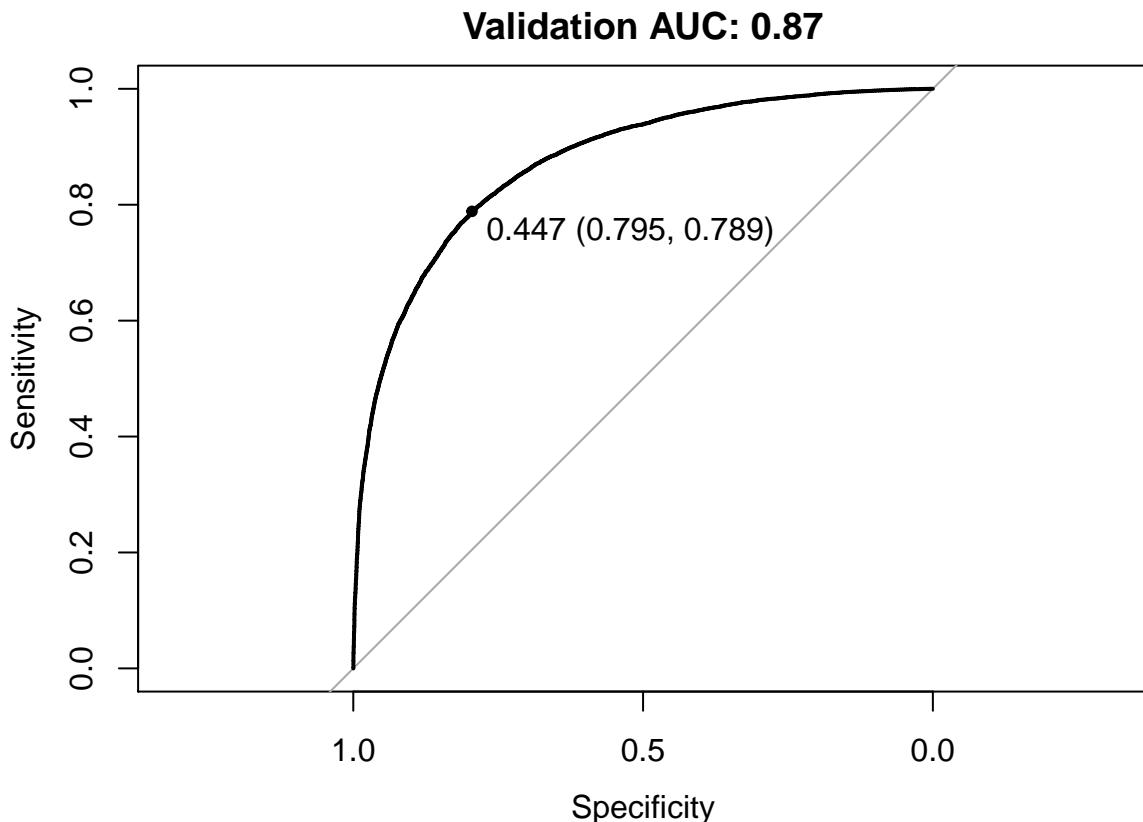
Como podemos observar, nos devuelve una precisión del 79%. Un dato importante es el ‘Balanced Accuracy’ que nos indica la existencia de desbalanceo entre las clases, algo que ya pudimos entrever en el análisis exploratorio de datos. Sería interesante realizar alguna técnica de balanceamiento para solucionar esta cuestión.

A continuación vamos a tratar de representar de forma gráfica la sensibilidad frente a la especificidad del clasificador binario según varíe el umbral de discriminación, o lo que es lo mismo, la proporción de verdaderos positivos frente a la proporción de falsos negativos en variación del umbral de discriminación. Recordemos que el umbral de discriminación es el valor a partir el cuál decidimos que un caso es un positivo.

```

randomForestAUC <- predict(randomForest, test, type = "prob")
randomForestROC <- roc(test$Label, randomForestAUC[["s"]], levels = unique(test[["Label"]]))
radnomForestROCplot <- plot.roc(randomForestROC, ylim=c(0,1), type = "S" , print.thres = T, main=paste0(

```



Como podemos observar en el anterior gráfico, el área de la curva es de 0.87, es decir, cercano al 0.9, por lo que se estima que el test es muy bueno.

5.3 Árboles de decisión

Durante esta sección vamos a tratar crear otro modelo de clasificación a partir de otro de los algoritmos de aprendizaje supervisado que nos proporciona *caret*, que es el árbol de decisión. La decisión de tomar este algoritmo de aprendizaje se basa en la naturaleza de los datos, en que nos devolvería un conjunto de reglas a partir de las cuales podríamos tomar futuras decisiones y que no es un algoritmo computacionalmente excesivamente costoso.

```
arbolDeDecision <- train(Label ~ ., data = entrenamiento, metric = "ROC", method = "rpart",
                           trControl = ...)

## + Fold1: cp=0.01978
## - Fold1: cp=0.01978
## + Fold2: cp=0.01978
## - Fold2: cp=0.01978
## + Fold3: cp=0.01978
## - Fold3: cp=0.01978
## + Fold4: cp=0.01978
## - Fold4: cp=0.01978
## + Fold5: cp=0.01978
## - Fold5: cp=0.01978
## Aggregating results
## Selecting tuning parameters
## Fitting cp = 0.0198 on full training set

arbolDeDecisionPrediccion <- predict(arbolDeDecision, test)
arbolDeDecisionMatrizConfusion <- confusionMatrix(table(arbolDeDecisionPrediccion, test[[Label]]))
```

```

arbolDeDecisionMatrizConfusion

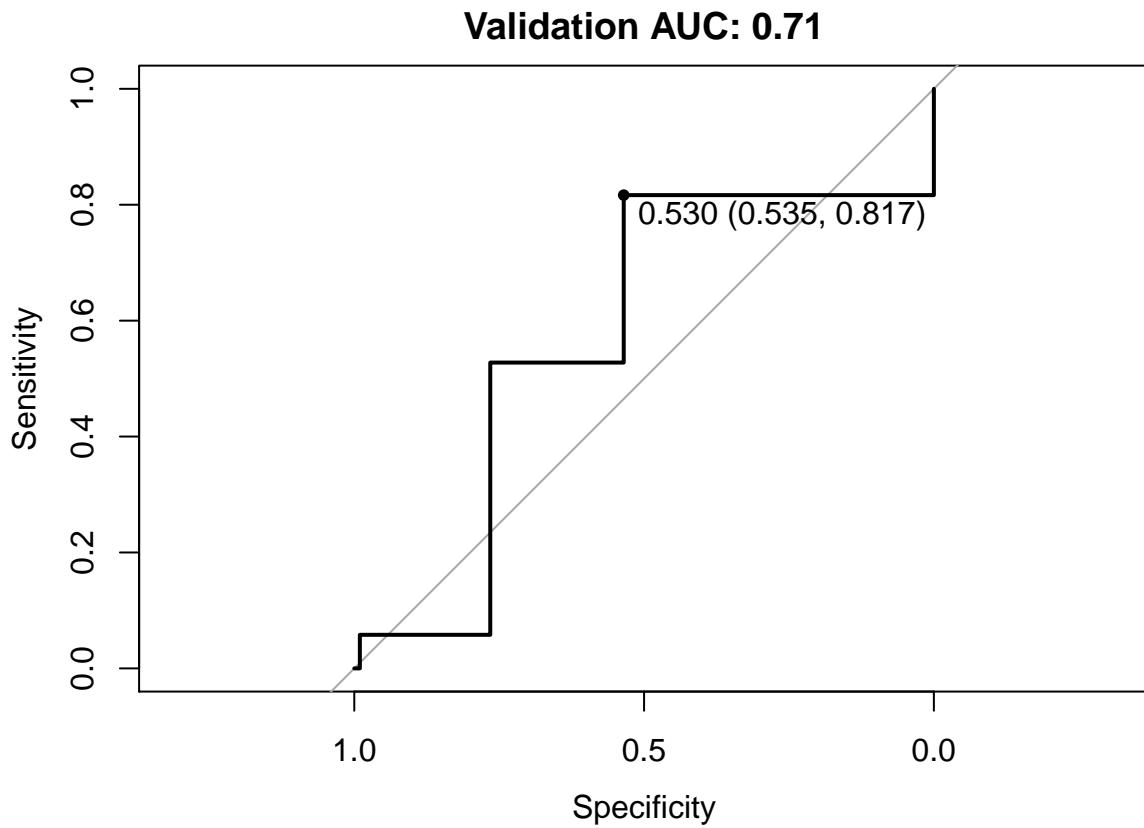
## Confusion Matrix and Statistics
##
##
## arbolDeDecisionPrediccion      b      s
##                         b 19401  8214
##                         s  4358  9455
##
##          Accuracy : 0.6965
##          95% CI : (0.6921, 0.701)
##  No Information Rate : 0.5735
##  P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 0.3618
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##          Sensitivity : 0.8166
##          Specificity : 0.5351
##  Pos Pred Value : 0.7026
##  Neg Pred Value : 0.6845
##          Prevalence : 0.5735
##  Detection Rate : 0.4683
## Detection Prevalence : 0.6666
##  Balanced Accuracy : 0.6758
##
##  'Positive' Class : b
##

```

Como podemos observar, el rendimiento del clasificador es menor al del *random forest*, aunque presenta un menor valor de *balanced accuracy*.

```

arbolDeDecisionAUC <- predict(arbolDeDecision, test, type = "prob")
arbolDeDecisionROC <- roc(test$Label, arbolDeDecisionAUC[["s"]], levels = unique(test[["Label"]]))
arbolDeDecisionROCplot <- plot.roc(arbolDeDecisionROC, ylim=c(0,1), type = "S" , print.thres = T, main=
```



En este caso, también notamos un desempeño más bajo del test, siendo de calidad media.

5.4 KNN

Vamos a tratar de aplicar el algoritmo KNN a nuestro conjunto de datos en busca de un mejor modelo.

```
knn <- train(Label ~ ., data = entrenamiento, metric = "ROC", method = "knn", trControl = cv)

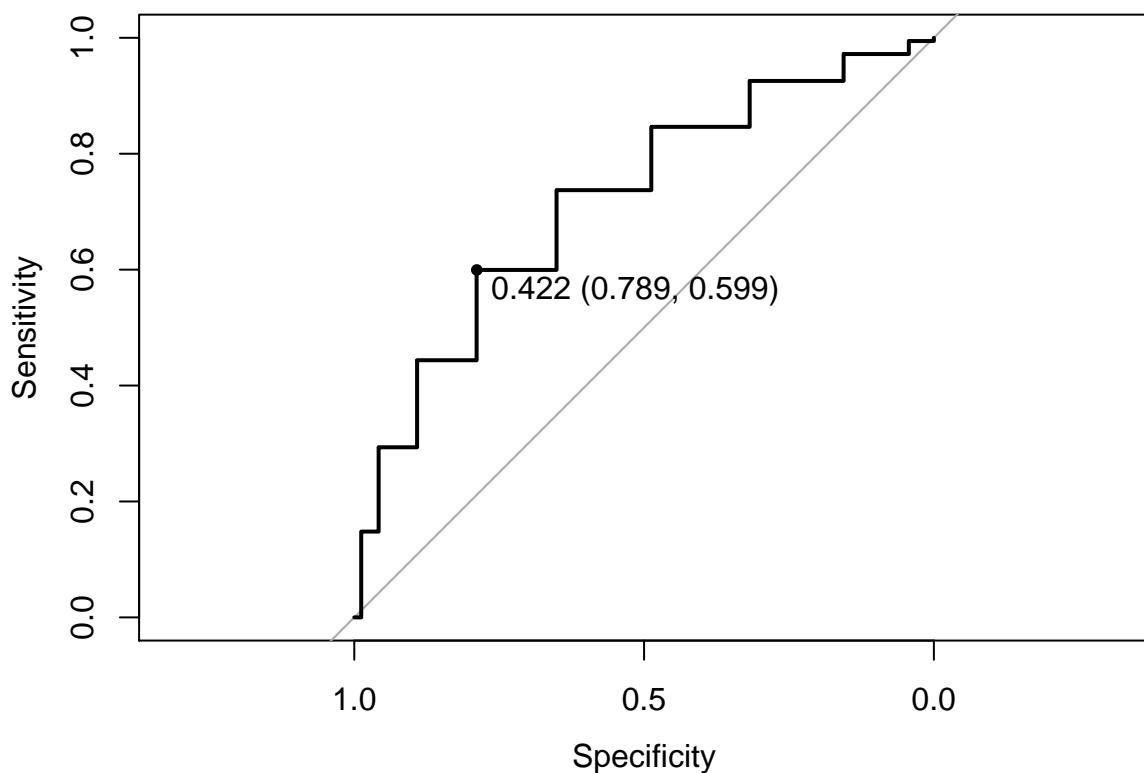
## + Fold1: k=5
## - Fold1: k=5
## + Fold1: k=7
## - Fold1: k=7
## + Fold1: k=9
## - Fold1: k=9
## + Fold2: k=5
## - Fold2: k=5
## + Fold2: k=7
## - Fold2: k=7
## + Fold2: k=9
## - Fold2: k=9
## + Fold3: k=5
## - Fold3: k=5
## + Fold3: k=7
## - Fold3: k=7
## + Fold3: k=9
## - Fold3: k=9
## + Fold4: k=5
## - Fold4: k=5
## + Fold4: k=7
```

```

## - Fold4: k=7
## + Fold4: k=9
## - Fold4: k=9
## + Fold5: k=5
## - Fold5: k=5
## + Fold5: k=7
## - Fold5: k=7
## + Fold5: k=9
## - Fold5: k=9
## Aggregating results
## Selecting tuning parameters
## Fitting k = 9 on full training set
knnPrediccion <- predict(knn, test)
knnMatrizConfusion <- confusionMatrix(table(knnPrediccion, test[["Label"]]))
knnMatrizConfusion

## Confusion Matrix and Statistics
##
## knnPrediccion      b      s
##           b 17510  6167
##           s  6249 11502
##
##           Accuracy : 0.7003
##           95% CI : (0.6959, 0.7047)
##   No Information Rate : 0.5735
##   P-Value [Acc > NIR] : <2e-16
##
##           Kappa : 0.3877
##
##   Mcnemar's Test P-Value : 0.4673
##
##           Sensitivity : 0.7370
##           Specificity : 0.6510
##   Pos Pred Value : 0.7395
##   Neg Pred Value : 0.6480
##           Prevalence : 0.5735
##           Detection Rate : 0.4227
##   Detection Prevalence : 0.5715
##           Balanced Accuracy : 0.6940
##
##           'Positive' Class : b
##
knnAUC <- predict(knn, test, type = "prob")
knnROC <- roc(test$Label, knnAUC[["s"]], levels = unique(test[["Label"]]))
knnROCPplot <- plot.roc(knnROC, ylim=c(0,1), type = "S" , print.thres = T, main=paste('Validation AUC:',
```

Validation AUC: 0.76



5.5 Redes Neuronales Artificiales

Llegados a este punto, vamos a aplicar el uso de redes neuronales artifical con la esperanza de encontrar un sistema de clasificación mejor que los vistos hasta el momento. La motivación nace en que las redes neuronales sobresalen en áreas donde la detección de características es difícil de expresar o está oculta, ideal para nuestro caso.

```
rnn <- train(Label ~ ., data = entrenamiento, metric = "ROC", method = "nnet", trControl = cv)

## + Fold1: size=1, decay=0e+00
## # weights: 17
## initial value 53720.073705
## iter 10 value 49670.665535
## iter 20 value 46140.028310
## iter 30 value 45944.274969
## iter 40 value 45912.764453
## iter 50 value 45910.354683
## final value 45910.013305
## converged
## - Fold1: size=1, decay=0e+00
## + Fold1: size=3, decay=0e+00
## # weights: 49
## initial value 76304.073822
## iter 10 value 51364.445651
## iter 20 value 49255.477053
## iter 30 value 47816.640313
## iter 40 value 46036.851921
## iter 50 value 44016.626430
```

```

## iter  60 value 40096.533923
## iter  70 value 37238.867844
## iter  80 value 36806.689739
## iter  90 value 36712.050609
## iter 100 value 36701.963366
## final  value 36701.963366
## stopped after 100 iterations
## - Fold1: size=3, decay=0e+00
## + Fold1: size=5, decay=0e+00
## # weights:  81
## initial  value 55209.230428
## iter   10 value 49379.137012
## iter   20 value 46514.587572
## iter   30 value 44836.745920
## iter   40 value 42360.973314
## iter   50 value 38515.422195
## iter   60 value 35925.493506
## iter   70 value 35292.448162
## iter   80 value 35126.295775
## iter   90 value 35052.270891
## iter 100 value 34930.127278
## final  value 34930.127278
## stopped after 100 iterations
## - Fold1: size=5, decay=0e+00
## + Fold1: size=1, decay=1e-01
## # weights:  17
## initial  value 53217.854151
## iter   10 value 49472.201309
## iter   20 value 47371.736096
## iter   30 value 46393.599697
## iter   40 value 46300.165494
## final  value 46300.060657
## converged
## - Fold1: size=1, decay=1e-01
## + Fold1: size=3, decay=1e-01
## # weights:  49
## initial  value 60726.730143
## iter   10 value 49759.743502
## iter   20 value 47832.890642
## iter   30 value 45562.084403
## iter   40 value 43418.933421
## iter   50 value 40383.772872
## iter   60 value 38108.677505
## iter   70 value 37378.780551
## iter   80 value 37044.805465
## iter   90 value 36826.833382
## iter 100 value 36614.731545
## final  value 36614.731545
## stopped after 100 iterations
## - Fold1: size=3, decay=1e-01
## + Fold1: size=5, decay=1e-01
## # weights:  81
## initial  value 64512.469043
## iter   10 value 49449.128301

```

```

## iter 20 value 46861.173433
## iter 30 value 45157.271939
## iter 40 value 44090.817479
## iter 50 value 42482.888038
## iter 60 value 40576.310219
## iter 70 value 38463.315063
## iter 80 value 37682.206974
## iter 90 value 36887.593426
## iter 100 value 36486.715014
## final value 36486.715014
## stopped after 100 iterations
## - Fold1: size=5, decay=1e-01
## + Fold1: size=1, decay=1e-04
## # weights: 17
## initial value 59378.067101
## iter 10 value 50792.015328
## iter 20 value 48745.240592
## iter 30 value 46105.551991
## iter 40 value 45193.700225
## iter 50 value 45175.562785
## iter 60 value 45171.132370
## final value 45171.129232
## converged
## - Fold1: size=1, decay=1e-04
## + Fold1: size=3, decay=1e-04
## # weights: 49
## initial value 52841.124521
## iter 10 value 49300.430505
## iter 20 value 47501.174571
## iter 30 value 45215.293141
## iter 40 value 40874.890629
## iter 50 value 38590.715481
## iter 60 value 36845.542504
## iter 70 value 36292.150844
## iter 80 value 36041.127294
## iter 90 value 35950.474990
## iter 100 value 35913.456995
## final value 35913.456995
## stopped after 100 iterations
## - Fold1: size=3, decay=1e-04
## + Fold1: size=5, decay=1e-04
## # weights: 81
## initial value 52775.529175
## iter 10 value 49464.713806
## iter 20 value 46999.941840
## iter 30 value 43605.637352
## iter 40 value 39267.859854
## iter 50 value 36788.230719
## iter 60 value 35613.790548
## iter 70 value 35429.709880
## iter 80 value 35283.592633
## iter 90 value 35171.603924
## iter 100 value 35126.236360
## final value 35126.236360

```

```

## stopped after 100 iterations
## - Fold1: size=5, decay=1e-04
## + Fold2: size=1, decay=0e+00
## # weights: 17
## initial value 65913.497873
## iter 10 value 49744.271038
## iter 20 value 46121.898640
## iter 30 value 45210.336688
## iter 40 value 45202.467018
## iter 50 value 45195.519625
## final value 45194.715907
## converged
## - Fold2: size=1, decay=0e+00
## + Fold2: size=3, decay=0e+00
## # weights: 49
## initial value 52997.806038
## iter 10 value 50361.010905
## iter 20 value 47964.396980
## iter 30 value 42904.143840
## iter 40 value 37234.198209
## iter 50 value 36150.615438
## iter 60 value 35907.195944
## iter 70 value 35784.804122
## iter 80 value 35707.146370
## iter 90 value 35592.545642
## iter 100 value 35576.738457
## final value 35576.738457
## stopped after 100 iterations
## - Fold2: size=3, decay=0e+00
## + Fold2: size=5, decay=0e+00
## # weights: 81
## initial value 56008.320373
## iter 10 value 48881.252050
## iter 20 value 45565.453251
## iter 30 value 43428.637880
## iter 40 value 41241.135545
## iter 50 value 38448.472741
## iter 60 value 36674.664613
## iter 70 value 35482.833895
## iter 80 value 34996.518765
## iter 90 value 34800.645057
## iter 100 value 34712.205336
## final value 34712.205336
## stopped after 100 iterations
## - Fold2: size=5, decay=0e+00
## + Fold2: size=1, decay=1e-01
## # weights: 17
## initial value 55896.487968
## iter 10 value 49227.703191
## iter 20 value 47538.254656
## iter 30 value 46451.403061
## iter 40 value 46335.634821
## iter 50 value 46332.346475
## final value 46332.296354

```

```

## converged
## - Fold2: size=1, decay=1e-01
## + Fold2: size=3, decay=1e-01
## # weights: 49
## initial value 56977.268757
## iter 10 value 52809.750245
## iter 20 value 52306.959730
## iter 30 value 50173.205348
## iter 40 value 48320.108732
## iter 50 value 45565.728335
## iter 60 value 41817.124628
## iter 70 value 40800.896455
## iter 80 value 38854.797604
## iter 90 value 37267.693948
## iter 100 value 36922.532496
## final value 36922.532496
## stopped after 100 iterations
## - Fold2: size=3, decay=1e-01
## + Fold2: size=5, decay=1e-01
## # weights: 81
## initial value 53914.559984
## iter 10 value 49316.866476
## iter 20 value 47546.143270
## iter 30 value 45213.501816
## iter 40 value 41836.175868
## iter 50 value 38785.573517
## iter 60 value 37385.284659
## iter 70 value 36828.495149
## iter 80 value 36582.839696
## iter 90 value 36372.045890
## iter 100 value 36268.095018
## final value 36268.095018
## stopped after 100 iterations
## - Fold2: size=5, decay=1e-01
## + Fold2: size=1, decay=1e-04
## # weights: 17
## initial value 53542.975137
## iter 10 value 49044.997135
## iter 20 value 46217.648758
## iter 30 value 45273.875831
## iter 40 value 45214.008441
## iter 50 value 45200.925596
## final value 45198.889396
## converged
## - Fold2: size=1, decay=1e-04
## + Fold2: size=3, decay=1e-04
## # weights: 49
## initial value 54716.179135
## iter 10 value 49757.565161
## iter 20 value 47924.499871
## iter 30 value 42871.679990
## iter 40 value 37995.581447
## iter 50 value 36791.872658
## iter 60 value 36407.538725

```

```

## iter 70 value 36126.928319
## iter 80 value 36014.059500
## iter 90 value 35970.058474
## iter 100 value 35941.009735
## final value 35941.009735
## stopped after 100 iterations
## - Fold2: size=3, decay=1e-04
## + Fold2: size=5, decay=1e-04
## # weights: 81
## initial value 53872.890520
## iter 10 value 49615.156658
## iter 20 value 46449.382355
## iter 30 value 44371.715925
## iter 40 value 40464.714444
## iter 50 value 37231.343336
## iter 60 value 35574.080542
## iter 70 value 35261.741436
## iter 80 value 35146.436974
## iter 90 value 35081.567075
## iter 100 value 35022.404429
## final value 35022.404429
## stopped after 100 iterations
## - Fold2: size=5, decay=1e-04
## + Fold3: size=1, decay=0e+00
## # weights: 17
## initial value 56770.726642
## iter 10 value 51849.432372
## iter 20 value 49229.961427
## iter 30 value 47700.437412
## iter 40 value 47054.823972
## iter 50 value 46396.884636
## iter 60 value 46081.825663
## iter 70 value 45973.722970
## iter 80 value 45951.006470
## final value 45949.907567
## converged
## - Fold3: size=1, decay=0e+00
## + Fold3: size=3, decay=0e+00
## # weights: 49
## initial value 55593.337870
## iter 10 value 50393.095249
## iter 20 value 48840.783005
## iter 30 value 47194.977026
## iter 40 value 45820.396042
## iter 50 value 43957.612075
## iter 60 value 38988.913691
## iter 70 value 37132.663990
## iter 80 value 36160.907508
## iter 90 value 35889.295898
## iter 100 value 35815.822946
## final value 35815.822946
## stopped after 100 iterations
## - Fold3: size=3, decay=0e+00
## + Fold3: size=5, decay=0e+00

```

```

## # weights:  81
## initial  value 52739.162479
## iter   10 value 49677.715451
## iter   20 value 47557.799345
## iter   30 value 44859.373592
## iter   40 value 42866.690497
## iter   50 value 39467.622471
## iter   60 value 36771.029171
## iter   70 value 35467.544547
## iter   80 value 35111.486957
## iter   90 value 34921.625711
## iter 100 value 34762.225623
## final  value 34762.225623
## stopped after 100 iterations
## - Fold3: size=5, decay=0e+00
## + Fold3: size=1, decay=1e-01
## # weights:  17
## initial  value 57383.836120
## iter   10 value 52106.259474
## iter   20 value 47822.885718
## iter   30 value 46319.285117
## iter   40 value 46222.027920
## iter   50 value 46220.669034
## iter   60 value 46218.526108
## final  value 46218.517624
## converged
## - Fold3: size=1, decay=1e-01
## + Fold3: size=3, decay=1e-01
## # weights:  49
## initial  value 59572.972027
## iter   10 value 49741.270728
## iter   20 value 47582.063826
## iter   30 value 44269.252836
## iter   40 value 40778.827555
## iter   50 value 39261.595698
## iter   60 value 37914.824081
## iter   70 value 37615.756880
## iter   80 value 37446.932894
## iter   90 value 37250.190332
## iter 100 value 37080.051179
## final  value 37080.051179
## stopped after 100 iterations
## - Fold3: size=3, decay=1e-01
## + Fold3: size=5, decay=1e-01
## # weights:  81
## initial  value 58985.314043
## iter   10 value 48911.863318
## iter   20 value 45512.031523
## iter   30 value 42760.239485
## iter   40 value 40084.286135
## iter   50 value 38237.318516
## iter   60 value 37304.470787
## iter   70 value 36875.477684
## iter   80 value 36324.264541

```

```

## iter  90 value 36218.361579
## iter 100 value 36186.801171
## final  value 36186.801171
## stopped after 100 iterations
## - Fold3: size=5, decay=1e-01
## + Fold3: size=1, decay=1e-04
## # weights:  17
## initial  value 59286.998454
## iter   10 value 51069.782679
## iter   20 value 46900.584643
## iter   30 value 46240.548889
## iter   40 value 45978.664524
## iter   50 value 45951.830777
## iter   60 value 45950.408992
## iter   60 value 45950.408943
## iter   60 value 45950.408943
## final  value 45950.408943
## converged
## - Fold3: size=1, decay=1e-04
## + Fold3: size=3, decay=1e-04
## # weights:  49
## initial  value 53076.058612
## iter   10 value 50002.194560
## iter   20 value 46648.704481
## iter   30 value 42737.653858
## iter   40 value 38407.299548
## iter   50 value 36841.070211
## iter   60 value 36552.226666
## iter   70 value 36348.209764
## iter   80 value 36008.477035
## iter   90 value 35869.435754
## iter 100 value 35827.309201
## final  value 35827.309201
## stopped after 100 iterations
## - Fold3: size=3, decay=1e-04
## + Fold3: size=5, decay=1e-04
## # weights:  81
## initial  value 52615.911646
## iter   10 value 49838.149957
## iter   20 value 47157.419382
## iter   30 value 45257.808301
## iter   40 value 42367.513773
## iter   50 value 37423.308810
## iter   60 value 36104.114753
## iter   70 value 35432.151810
## iter   80 value 35042.154172
## iter   90 value 34860.429347
## iter 100 value 34779.280624
## final  value 34779.280624
## stopped after 100 iterations
## - Fold3: size=5, decay=1e-04
## + Fold4: size=1, decay=0e+00
## # weights:  17
## initial  value 53454.072603

```

```

## iter 10 value 50633.116775
## iter 20 value 46181.596813
## iter 30 value 45289.938032
## iter 40 value 45118.942777
## iter 50 value 45115.786523
## final value 45113.485924
## converged
## - Fold4: size=1, decay=0e+00
## + Fold4: size=3, decay=0e+00
## # weights: 49
## initial value 53312.551795
## iter 10 value 50263.526566
## iter 20 value 48543.433763
## iter 30 value 46089.371340
## iter 40 value 40484.539235
## iter 50 value 37222.154197
## iter 60 value 36734.788096
## iter 70 value 36501.414581
## iter 80 value 36123.982135
## iter 90 value 36036.450269
## iter 100 value 35980.462539
## final value 35980.462539
## stopped after 100 iterations
## - Fold4: size=3, decay=0e+00
## + Fold4: size=5, decay=0e+00
## # weights: 81
## initial value 54464.136872
## iter 10 value 48944.296000
## iter 20 value 45802.921112
## iter 30 value 42965.832021
## iter 40 value 39779.615411
## iter 50 value 38281.493131
## iter 60 value 36317.385355
## iter 70 value 35730.425194
## iter 80 value 35314.020833
## iter 90 value 35119.701554
## iter 100 value 35027.973163
## final value 35027.973163
## stopped after 100 iterations
## - Fold4: size=5, decay=0e+00
## + Fold4: size=1, decay=1e-01
## # weights: 17
## initial value 53714.355952
## iter 10 value 50095.991669
## iter 20 value 46757.734987
## iter 30 value 46355.126678
## iter 40 value 46156.253107
## iter 50 value 46152.016326
## iter 60 value 46149.479206
## iter 60 value 46149.479144
## iter 60 value 46149.479144
## final value 46149.479144
## converged
## - Fold4: size=1, decay=1e-01

```

```

## + Fold4: size=3, decay=1e-01
## # weights: 49
## initial value 54445.195744
## iter 10 value 49930.920912
## iter 20 value 47743.932362
## iter 30 value 45904.221249
## iter 40 value 43097.675100
## iter 50 value 40715.785175
## iter 60 value 39005.542400
## iter 70 value 38172.899537
## iter 80 value 37535.803480
## iter 90 value 36822.763938
## iter 100 value 36499.857748
## final value 36499.857748
## stopped after 100 iterations
## - Fold4: size=3, decay=1e-01
## + Fold4: size=5, decay=1e-01
## # weights: 81
## initial value 53274.879977
## iter 10 value 49788.060335
## iter 20 value 47650.804238
## iter 30 value 45433.202774
## iter 40 value 42292.990778
## iter 50 value 40032.316587
## iter 60 value 38736.592273
## iter 70 value 37258.761019
## iter 80 value 36692.790365
## iter 90 value 36524.934008
## iter 100 value 36432.927941
## final value 36432.927941
## stopped after 100 iterations
## - Fold4: size=5, decay=1e-01
## + Fold4: size=1, decay=1e-04
## # weights: 17
## initial value 59750.192506
## iter 10 value 49640.745609
## iter 20 value 47895.947727
## iter 30 value 47474.221362
## iter 40 value 46205.453362
## iter 50 value 45162.184433
## iter 60 value 45117.901530
## final value 45117.804621
## converged
## - Fold4: size=1, decay=1e-04
## + Fold4: size=3, decay=1e-04
## # weights: 49
## initial value 55688.761417
## iter 10 value 49746.734746
## iter 20 value 47309.922952
## iter 30 value 42393.815108
## iter 40 value 37570.519098
## iter 50 value 36515.387255
## iter 60 value 36045.988816
## iter 70 value 35878.970358

```

```

## iter  80 value 35737.866986
## iter  90 value 35706.948602
## iter 100 value 35663.102749
## final  value 35663.102749
## stopped after 100 iterations
## - Fold4: size=3, decay=1e-04
## + Fold4: size=5, decay=1e-04
## # weights:  81
## initial  value 53593.064246
## iter   10 value 49302.419710
## iter   20 value 46283.861994
## iter   30 value 43643.907803
## iter   40 value 40093.018814
## iter   50 value 37079.336704
## iter   60 value 36549.610644
## iter   70 value 36262.348483
## iter   80 value 35983.890077
## iter   90 value 35575.564668
## iter 100 value 35351.650231
## final  value 35351.650231
## stopped after 100 iterations
## - Fold4: size=5, decay=1e-04
## + Fold5: size=1, decay=0e+00
## # weights:  17
## initial  value 54891.823689
## iter   10 value 50028.963757
## iter   20 value 46128.462170
## iter   30 value 45892.146554
## iter   40 value 45879.490829
## final  value 45879.488474
## converged
## - Fold5: size=1, decay=0e+00
## + Fold5: size=3, decay=0e+00
## # weights:  49
## initial  value 54195.276324
## iter   10 value 49904.359809
## iter   20 value 47492.306040
## iter   30 value 41414.773163
## iter   40 value 37680.792532
## iter   50 value 36376.493877
## iter   60 value 36016.803210
## iter   70 value 35884.357644
## iter   80 value 35784.742073
## iter   90 value 35754.205927
## iter 100 value 35739.230815
## final  value 35739.230815
## stopped after 100 iterations
## - Fold5: size=3, decay=0e+00
## + Fold5: size=5, decay=0e+00
## # weights:  81
## initial  value 55404.792161
## iter   10 value 50163.521641
## iter   20 value 48460.805058
## iter   30 value 45726.672918

```

```

## iter 40 value 43251.797819
## iter 50 value 39965.393417
## iter 60 value 36266.280548
## iter 70 value 35646.111369
## iter 80 value 35283.795824
## iter 90 value 35104.967385
## iter 100 value 34990.397580
## final value 34990.397580
## stopped after 100 iterations
## - Fold5: size=5, decay=0e+00
## + Fold5: size=1, decay=1e-01
## # weights: 17
## initial value 62736.151426
## iter 10 value 50053.161119
## iter 20 value 48988.212105
## iter 30 value 46875.079084
## iter 40 value 46333.865997
## final value 46291.834009
## converged
## - Fold5: size=1, decay=1e-01
## + Fold5: size=3, decay=1e-01
## # weights: 49
## initial value 55663.383945
## iter 10 value 48647.713538
## iter 20 value 45774.123804
## iter 30 value 40954.231622
## iter 40 value 38847.796837
## iter 50 value 37841.526984
## iter 60 value 37058.807629
## iter 70 value 36824.607795
## iter 80 value 36773.076288
## iter 90 value 36760.896370
## iter 100 value 36756.149190
## final value 36756.149190
## stopped after 100 iterations
## - Fold5: size=3, decay=1e-01
## + Fold5: size=5, decay=1e-01
## # weights: 81
## initial value 56290.846901
## iter 10 value 49826.626765
## iter 20 value 45351.148302
## iter 30 value 42980.174852
## iter 40 value 40861.028977
## iter 50 value 39118.457095
## iter 60 value 37700.367851
## iter 70 value 36925.310643
## iter 80 value 36267.425832
## iter 90 value 35988.734938
## iter 100 value 35883.080106
## final value 35883.080106
## stopped after 100 iterations
## - Fold5: size=5, decay=1e-01
## + Fold5: size=1, decay=1e-04
## # weights: 17

```

```

## initial value 58158.124045
## iter 10 value 49978.049923
## iter 20 value 46615.344028
## iter 30 value 45201.398041
## iter 40 value 45151.716018
## final value 45151.713269
## converged
## - Fold5: size=1, decay=1e-04
## + Fold5: size=3, decay=1e-04
## # weights: 49
## initial value 54051.165621
## iter 10 value 49607.273321
## iter 20 value 46684.249394
## iter 30 value 45957.429341
## iter 40 value 45861.923057
## iter 50 value 45445.883705
## iter 60 value 43431.377330
## iter 70 value 38715.037591
## iter 80 value 37242.400335
## iter 90 value 36827.314219
## iter 100 value 36685.831158
## final value 36685.831158
## stopped after 100 iterations
## - Fold5: size=3, decay=1e-04
## + Fold5: size=5, decay=1e-04
## # weights: 81
## initial value 58332.383724
## iter 10 value 50285.942458
## iter 20 value 47839.298600
## iter 30 value 44969.878967
## iter 40 value 41766.915252
## iter 50 value 38519.316076
## iter 60 value 36772.429713
## iter 70 value 36197.910195
## iter 80 value 35909.105950
## iter 90 value 35796.993118
## iter 100 value 35760.528679
## final value 35760.528679
## stopped after 100 iterations
## - Fold5: size=5, decay=1e-04
## Aggregating results
## Selecting tuning parameters
## Fitting size = 5, decay = 0 on full training set
## # weights: 81
## initial value 69713.788641
## iter 10 value 61580.725062
## iter 20 value 57873.768598
## iter 30 value 54766.696034
## iter 40 value 52378.408958
## iter 50 value 49580.156318
## iter 60 value 46982.284891
## iter 70 value 45715.274037
## iter 80 value 45386.249225
## iter 90 value 45244.540655

```

```

## iter 100 value 44869.071941
## final value 44869.071941
## stopped after 100 iterations

rnnPrediccion <- predict(rnn, test)
rnnMatrizConfusion <- confusionMatrix(table(rnnPrediccion, test[["Label"]]))
rnnMatrizConfusion

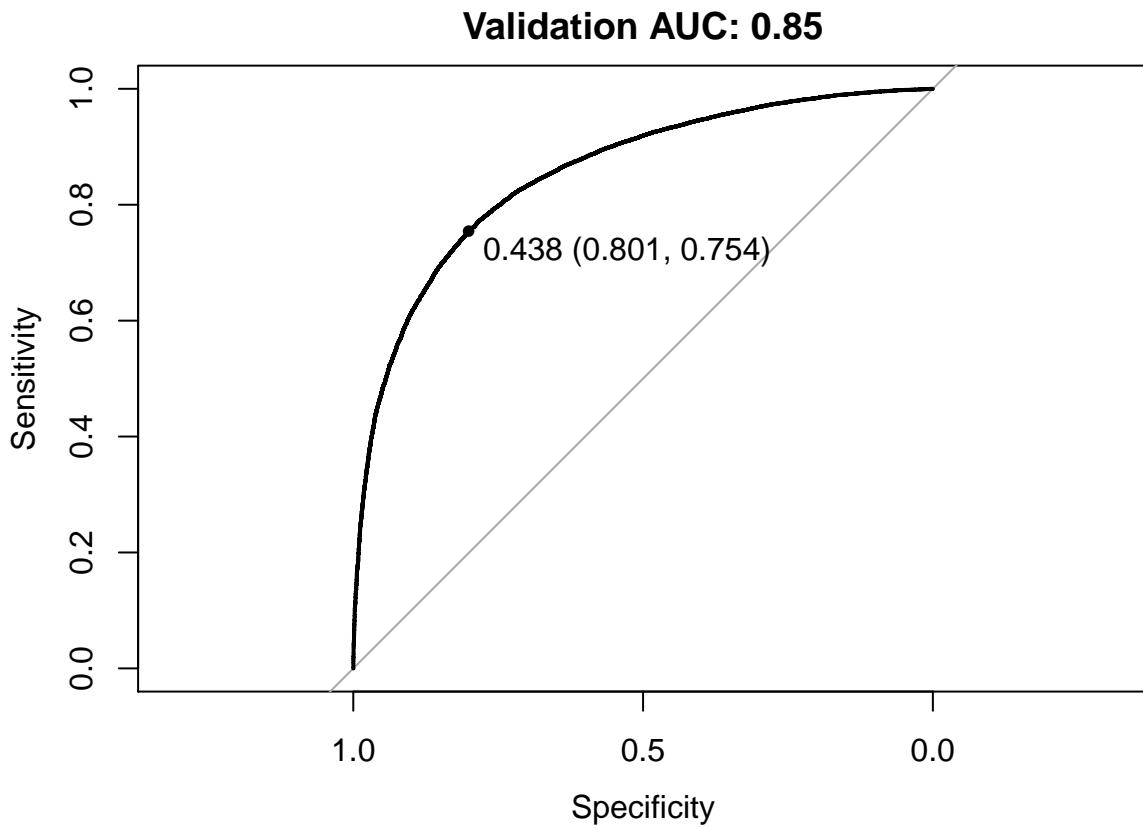
## Confusion Matrix and Statistics
##
##
## rnnPrediccion      b      s
##           b 18953  4403
##           s  4806 13266
##
##                 Accuracy : 0.7777
##                 95% CI : (0.7737, 0.7817)
##     No Information Rate : 0.5735
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                 Kappa : 0.5469
##
##     Mcnemar's Test P-Value : 2.801e-05
##
##                 Sensitivity : 0.7977
##                 Specificity : 0.7508
##     Pos Pred Value : 0.8115
##     Neg Pred Value : 0.7341
##                 Prevalence : 0.5735
##                 Detection Rate : 0.4575
##     Detection Prevalence : 0.5638
##     Balanced Accuracy : 0.7743
##
##     'Positive' Class : b
##

```

Como podemos observar, el modelo de red neuronal obtiene una precisión del 78% con una precisión del balanceo del 77%. Aunque son mejores parámetros que en el caso del árbol de decisión, son algo menores al modelo basado en *random forest*.

```

rnnAUC <- predict(rnn, test, type = "prob")
rnnROC <- roc(test$Label, rnnAUC[["s"]], levels = unique(test[["Label"]]))
rnnROCplot <- plot.roc(rnnROC, ylim=c(0,1), type = "S" , print.thres = T, main=paste('Validation AUC:',
```



Para el área bajo la curva, obtenemos un valor de 0.86, mucho mejor que el del árbol de decisión, pero peor que el del *random forest*.

5.6 Comparativa

```
columnas <- c("Random Forest", "Árbol de decisión", "KNN", "Redes Neuronales Artificiales")
precision <- c("79%", "69%","70%", "78%")
auc <- c("0.87", "0.71", "0.76", "0.86")
balanceo <- c("0.78","0.67","0.69", "0.77")

comparativa <- data.frame(columnas, precision, auc, balanceo)
knitr::kable(comparativa, "pipe")
```

columnas	precision	auc	balanceo
Random Forest	79%	0.87	0.78
Árbol de decisión	69%	0.71	0.67
KNN	70%	0.76	0.69
Redes Neuronales Artificiales	78%	0.86	0.77

6 Balanceamiento

Como hemos podido observar en la anterior comparativa, todos los modelos pecaban de un desbalanceo muy alto. Esta cuestión ya se podía ver de venir desde el análisis exploratorio de datos, pero aún así, decidimos continuar para ver qué podía ocurrir en forma de aprendizaje (nuestro, como estudiante).

Por ello, vamos a tratar de realizar un proceso de reducción de las observaciones de tal forma que se igualen

(se balanceen) las clases y repetiremos de nuevo la ejecución de los algoritmos de aprendizaje automático.

```
set.seed(0)

entrenamiento$Label <- as.factor(entrenamiento$Label)
entrenamientoBalanceadoReducido <- downSample(x = entrenamiento, y = entrenamiento$Label)
#entrenamientoBalanceadoReducido<- NULL

indiceBalanceado <- createDataPartition(entrenamientoBalanceadoReducido$Label, p = .7, list = FALSE)
entrenamientoBalanceadoDatos <- entrenamientoBalanceadoReducido[indiceBalanceado, ]
testBalanceadoDatos <- entrenamientoBalanceadoReducido[-indiceBalanceado, ]

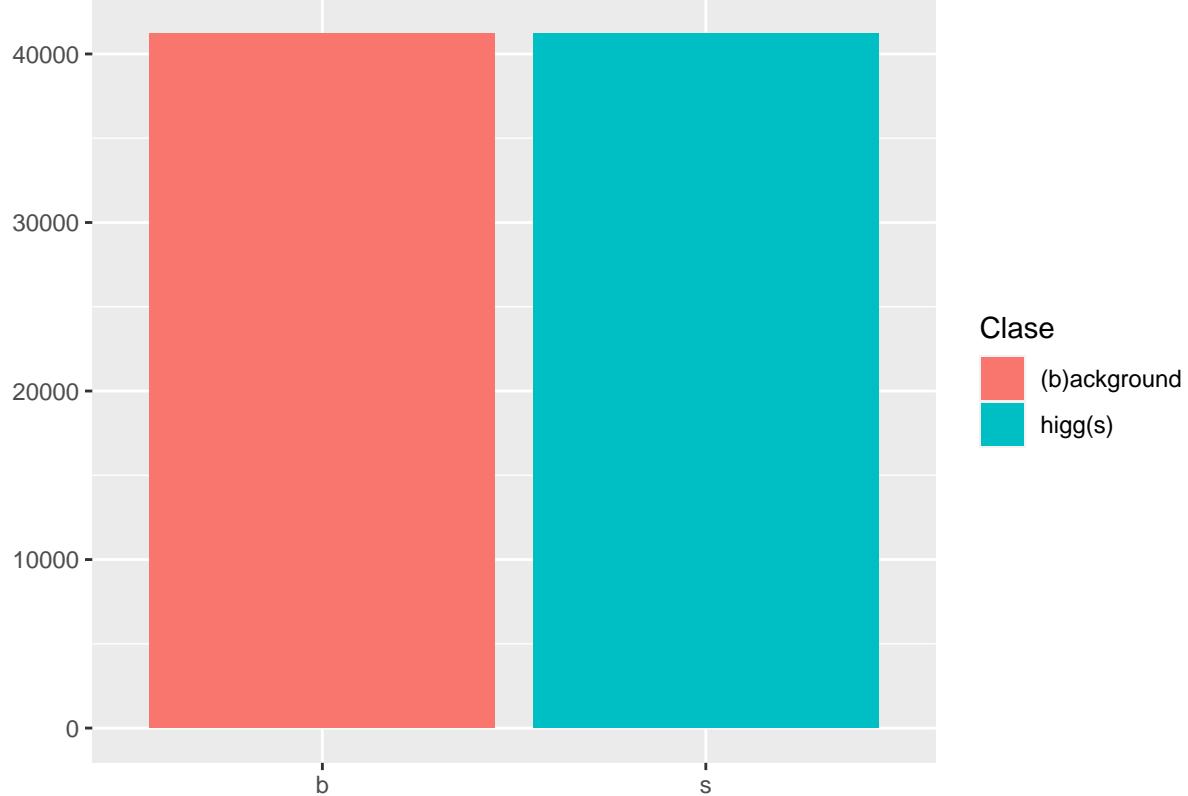
entrenamientoBalanceadoDatos['Class']<-NULL

#cvBalanceado <- trainControl(verboseIter = TRUE, classProbs = TRUE, summaryFunction = twoClassSummary,
```

A continuación podemos ver como el conjunto de datos se encuentra balanceado con el mismo número de elementos de clase bosón y ruido de fondo.

```
ggplot(entrenamientoBalanceadoReducido) +
geom_histogram(
  aes(x = Label, fill = Label), stat = "count") +
  labs(x = "", y = "") +
  scale_fill_discrete(name ="Clase", labels=c("(b)ackground", "higg(s)"))
)

## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



6.1 Random Forest

```
randomForestBalanceado <- train(Label ~ ., data = entranamientoBalanceadoDatos, metric = "ROC", method = "RF", tuneLength = 10, ntree = 1000, importance = TRUE, nfold = 5, trControl = trainControl(method = "cv", number = 5), tuneGrid = grid, preProcess = c("center", "scale"))  
  
## + Fold1: mtry= 2  
## - Fold1: mtry= 2  
## + Fold1: mtry= 8  
## - Fold1: mtry= 8  
## + Fold1: mtry=14  
## - Fold1: mtry=14  
## + Fold2: mtry= 2  
## - Fold2: mtry= 2  
## + Fold2: mtry= 8  
## - Fold2: mtry= 8  
## + Fold2: mtry=14  
## - Fold2: mtry=14  
## + Fold3: mtry= 2  
## - Fold3: mtry= 2  
## + Fold3: mtry= 8  
## - Fold3: mtry= 8  
## + Fold3: mtry=14  
## - Fold3: mtry=14  
## + Fold4: mtry= 2  
## - Fold4: mtry= 2  
## + Fold4: mtry= 8  
## - Fold4: mtry= 8  
## + Fold4: mtry=14  
## - Fold4: mtry=14  
## + Fold5: mtry= 2  
## - Fold5: mtry= 2  
## + Fold5: mtry= 8  
## - Fold5: mtry= 8  
## + Fold5: mtry=14  
## - Fold5: mtry=14  
## Aggregating results  
## Selecting tuning parameters  
## Fitting mtry = 8 on full training set  
randomForestBalanceadoPrediccion <- predict(randomForestBalanceado, testBalanceadoDatos)  
randomForestBalanceadoMatrizConfusion <- confusionMatrix(table(randomForestBalanceadoPrediccion, testBalanceadoDatos))  
  
## Confusion Matrix and Statistics  
##  
## randomForestBalanceadoPrediccion      b      s  
##                               b 9637 2398  
##                               s 2732 9971  
##  
##           Accuracy : 0.7926  
##           95% CI : (0.7875, 0.7977)  
##   No Information Rate : 0.5  
##   P-Value [Acc > NIR] : < 2.2e-16  
##  
##           Kappa : 0.5853
```

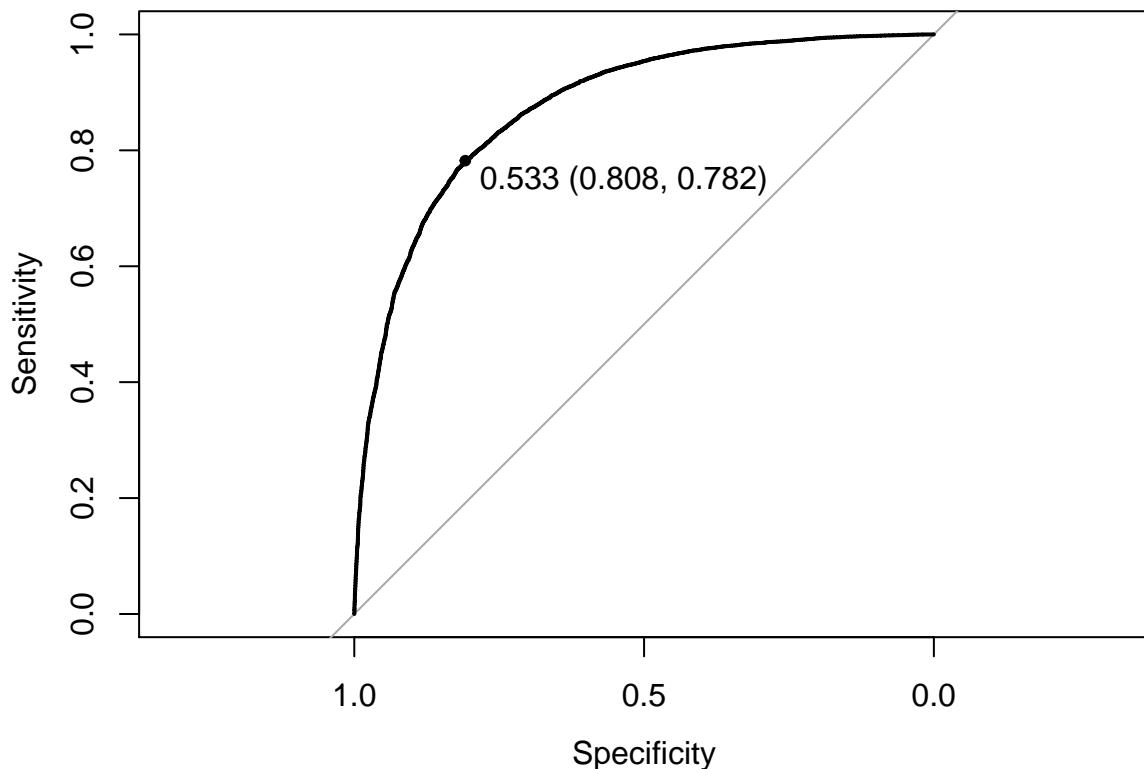
```

## 
##  Mcnemar's Test P-Value : 3.331e-06
## 
##          Sensitivity : 0.7791
##          Specificity : 0.8061
##          Pos Pred Value : 0.8007
##          Neg Pred Value : 0.7849
##          Prevalence : 0.5000
##          Detection Rate : 0.3896
##          Detection Prevalence : 0.4865
##          Balanced Accuracy : 0.7926
## 
##          'Positive' Class : b
## 

randomForestBalanceadoAUC <- predict(randomForestBalanceado, testBalanceadoDatos, type = "prob")
randomForestBalanceadoROC <- roc(testBalanceadoDatos$Label, randomForestBalanceadoAUC[["s"]], levels = c(0,1))
randomForestBalanceadoROCplot <- plot.roc(randomForestBalanceadoROC, ylim=c(0,1), type = "S" , print.th=0)

```

Validation AUC: 0.88



Podemos ver como la curva ROC ha mejorado algo, apuntándose ahora a 0.88

6.2 KNN

De nuevo, vamos a hacer lo propio con el modelo KNN creando un nuevo modelo a partir del conjunto de datos resultantes del balanceo.

```
knnBalanceado <- train(Label ~ ., data = entrainmentBalanceadoDatos, metric = "ROC", method = "knn", ...)
```

```
## + Fold1: k=5
```

```

## - Fold1: k=5
## + Fold1: k=7
## - Fold1: k=7
## + Fold1: k=9
## - Fold1: k=9
## + Fold2: k=5
## - Fold2: k=5
## + Fold2: k=7
## - Fold2: k=7
## + Fold2: k=9
## - Fold2: k=9
## + Fold3: k=5
## - Fold3: k=5
## + Fold3: k=7
## - Fold3: k=7
## + Fold3: k=9
## - Fold3: k=9
## + Fold4: k=5
## - Fold4: k=5
## + Fold4: k=7
## - Fold4: k=7
## + Fold4: k=9
## - Fold4: k=9
## + Fold5: k=5
## - Fold5: k=5
## + Fold5: k=7
## - Fold5: k=7
## + Fold5: k=9
## - Fold5: k=9
## Aggregating results
## Selecting tuning parameters
## Fitting k = 9 on full training set
knnPrediccionBalanceado <- predict(knnBalanceado, test)
knnMatrizConfusionBalanceado <- confusionMatrix(table(knnPrediccionBalanceado, test[["Label"]]))
knnMatrizConfusionBalanceado

## Confusion Matrix and Statistics
##
## 
## knnPrediccionBalanceado      b      s
##                         b 15102  4581
##                         s  8657 13088
##
##                               Accuracy : 0.6805
##                               95% CI : (0.6759, 0.6849)
## No Information Rate : 0.5735
## P-Value [Acc > NIR] : < 2.2e-16
##
##                               Kappa : 0.3656
##
## Mcnemar's Test P-Value : < 2.2e-16
##
##                               Sensitivity : 0.6356
##                               Specificity : 0.7407

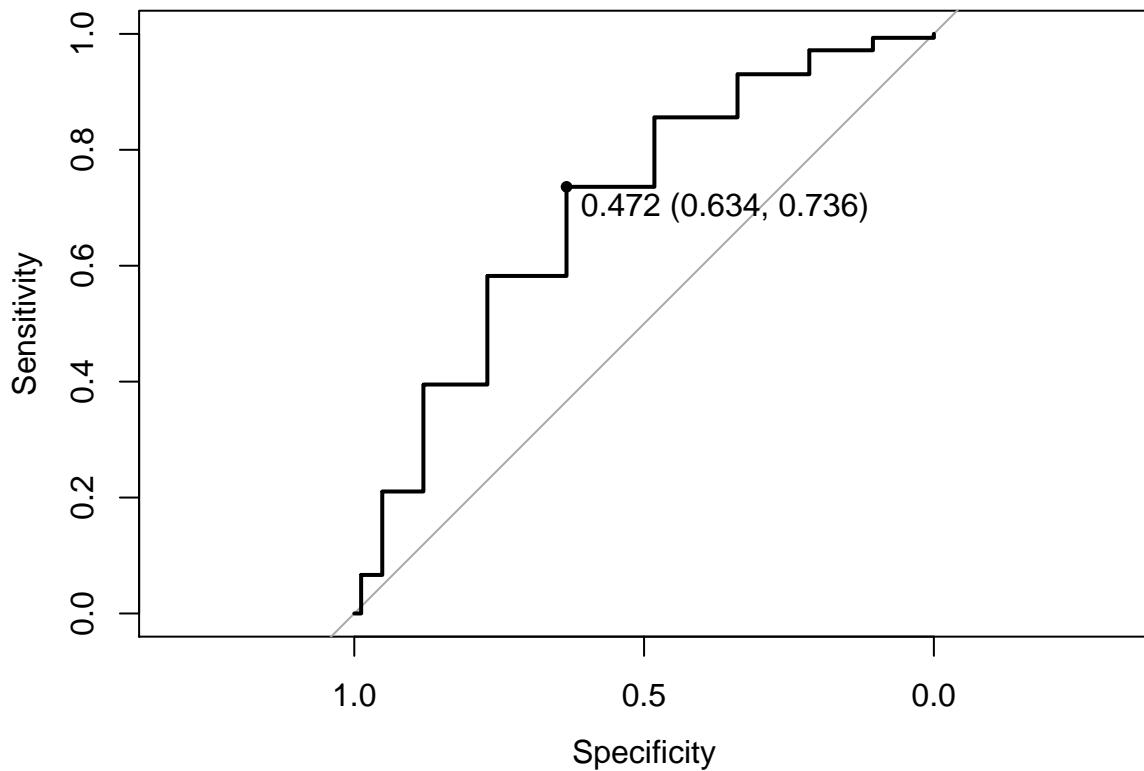
```

```

##           Pos Pred Value : 0.7673
##           Neg Pred Value : 0.6019
##           Prevalence : 0.5735
##           Detection Rate : 0.3645
##   Detection Prevalence : 0.4751
##           Balanced Accuracy : 0.6882
##
##           'Positive' Class : b
##
knnBalanceadoAUC <- predict(knnBalanceado, testBalanceadoDatos, type = "prob")
knnBalanceadoROC <- roc(testBalanceadoDatos$Label, knnBalanceadoAUC[["s"]], levels = unique(testBalanceadoDatos$Label))
knnBalanceadoROCplot <- plot.roc(knnBalanceadoROC, ylim=c(0,1),
                                    type = "S" , print.thres = T, main=paste('Validation AUC:', round(knnBalanceadoAUC, 3)))

```

Validation AUC: 0.75



6.3 Redes Neuronales Artificiales

Vamos a volver a las redes neuronales para ver qué tal se comportan con este balanceo.

```

rnnBalanceado <- train(Label ~ ., data = entrancimientoBalanceadoDatos, metric = "ROC", method = "nnet",
## + Fold1: size=1, decay=0e+00
## # weights: 17
## initial value 31990.218113
## iter 10 value 30577.035233
## iter 20 value 28514.758252
## iter 30 value 28177.655981
## iter 40 value 27742.389129
## iter 50 value 27713.545551

```

```

## iter 60 value 27691.224661
## final value 27690.340532
## converged
## - Fold1: size=1, decay=0e+00
## + Fold1: size=3, decay=0e+00
## # weights: 49
## initial value 35612.671587
## iter 10 value 30856.673033
## iter 20 value 28965.302971
## iter 30 value 26577.807606
## iter 40 value 24786.061182
## iter 50 value 23656.976821
## iter 60 value 22714.078758
## iter 70 value 22173.535063
## iter 80 value 22094.652340
## iter 90 value 22030.533069
## iter 100 value 22009.435590
## final value 22009.435590
## stopped after 100 iterations
## - Fold1: size=3, decay=0e+00
## + Fold1: size=5, decay=0e+00
## # weights: 81
## initial value 32438.782271
## iter 10 value 30280.253971
## iter 20 value 28117.160265
## iter 30 value 26399.222389
## iter 40 value 24629.506237
## iter 50 value 22581.443532
## iter 60 value 21674.897232
## iter 70 value 21505.450667
## iter 80 value 21410.845792
## iter 90 value 21328.453319
## iter 100 value 21302.940383
## final value 21302.940383
## stopped after 100 iterations
## - Fold1: size=5, decay=0e+00
## + Fold1: size=1, decay=1e-01
## # weights: 17
## initial value 32028.568879
## iter 10 value 30675.752382
## iter 20 value 29087.527943
## iter 30 value 28121.435337
## iter 40 value 27963.841173
## iter 50 value 27945.730927
## iter 60 value 27938.346452
## final value 27938.325860
## converged
## - Fold1: size=1, decay=1e-01
## + Fold1: size=3, decay=1e-01
## # weights: 49
## initial value 32803.788176
## iter 10 value 30138.750962
## iter 20 value 29000.580978
## iter 30 value 26237.440374

```

```

## iter 40 value 24510.781916
## iter 50 value 23999.585047
## iter 60 value 23098.698250
## iter 70 value 22697.738914
## iter 80 value 22572.292255
## iter 90 value 22549.782101
## iter 100 value 22540.281853
## final value 22540.281853
## stopped after 100 iterations
## - Fold1: size=3, decay=1e-01
## + Fold1: size=5, decay=1e-01
## # weights: 81
## initial value 32472.236139
## iter 10 value 29916.026078
## iter 20 value 28606.922015
## iter 30 value 26776.855318
## iter 40 value 24876.450419
## iter 50 value 23778.424879
## iter 60 value 22915.391895
## iter 70 value 22619.369678
## iter 80 value 22476.591825
## iter 90 value 22425.563162
## iter 100 value 22369.895194
## final value 22369.895194
## stopped after 100 iterations
## - Fold1: size=5, decay=1e-01
## + Fold1: size=1, decay=1e-04
## # weights: 17
## initial value 32041.865344
## iter 10 value 31121.966682
## iter 20 value 28606.747945
## iter 30 value 27726.965421
## iter 40 value 27695.061611
## iter 50 value 27691.539471
## final value 27690.887112
## converged
## - Fold1: size=1, decay=1e-04
## + Fold1: size=3, decay=1e-04
## # weights: 49
## initial value 32400.536794
## iter 10 value 30359.330509
## iter 20 value 28740.946142
## iter 30 value 25391.704278
## iter 40 value 23340.610856
## iter 50 value 22266.108601
## iter 60 value 21994.607251
## iter 70 value 21888.908805
## iter 80 value 21804.410591
## iter 90 value 21762.956412
## iter 100 value 21743.496215
## final value 21743.496215
## stopped after 100 iterations
## - Fold1: size=3, decay=1e-04
## + Fold1: size=5, decay=1e-04

```

```

## # weights:  81
## initial  value 33033.571353
## iter   10 value 30022.152316
## iter   20 value 28167.717885
## iter   30 value 26432.933986
## iter   40 value 26200.033451
## iter   50 value 25702.993938
## iter   60 value 25030.576316
## iter   70 value 23688.999296
## iter   80 value 22290.641435
## iter   90 value 21977.850064
## iter  100 value 21883.376969
## final   value 21883.376969
## stopped after 100 iterations
## - Fold1: size=5, decay=1e-04
## + Fold2: size=1, decay=0e+00
## # weights:  17
## initial  value 33292.411153
## iter   10 value 30340.965751
## iter   20 value 27818.945307
## iter   30 value 27413.283320
## iter   40 value 27211.365578
## iter   50 value 27211.222522
## final   value 27211.175980
## converged
## - Fold2: size=1, decay=0e+00
## + Fold2: size=3, decay=0e+00
## # weights:  49
## initial  value 33135.538023
## iter   10 value 30520.754838
## iter   20 value 28662.797764
## iter   30 value 25986.494482
## iter   40 value 23045.211642
## iter   50 value 22342.184437
## iter   60 value 22201.070463
## iter   70 value 22171.168659
## iter   80 value 22153.446627
## iter   90 value 22142.800631
## iter  100 value 22136.107136
## final   value 22136.107136
## stopped after 100 iterations
## - Fold2: size=3, decay=0e+00
## + Fold2: size=5, decay=0e+00
## # weights:  81
## initial  value 32096.456078
## iter   10 value 29812.630344
## iter   20 value 27940.917183
## iter   30 value 27124.100083
## iter   40 value 25584.938359
## iter   50 value 22390.502384
## iter   60 value 21660.700123
## iter   70 value 21388.813741
## iter   80 value 21222.039322
## iter   90 value 21164.350234

```

```

## iter 100 value 21128.265390
## final value 21128.265390
## stopped after 100 iterations
## - Fold2: size=5, decay=0e+00
## + Fold2: size=1, decay=1e-01
## # weights: 17
## initial value 32136.238248
## iter 10 value 30240.611647
## iter 20 value 28322.913565
## iter 30 value 27955.414225
## iter 40 value 27882.716761
## iter 50 value 27860.680633
## iter 60 value 27840.473929
## final value 27840.391301
## converged
## - Fold2: size=1, decay=1e-01
## + Fold2: size=3, decay=1e-01
## # weights: 49
## initial value 32194.460845
## iter 10 value 30115.955733
## iter 20 value 29111.871807
## iter 30 value 26230.481441
## iter 40 value 24211.618516
## iter 50 value 23280.717222
## iter 60 value 22706.308774
## iter 70 value 22505.229229
## iter 80 value 22432.137927
## iter 90 value 22429.107427
## iter 100 value 22426.300180
## final value 22426.300180
## stopped after 100 iterations
## - Fold2: size=3, decay=1e-01
## + Fold2: size=5, decay=1e-01
## # weights: 81
## initial value 32192.220568
## iter 10 value 30419.130362
## iter 20 value 29065.563594
## iter 30 value 28043.816200
## iter 40 value 25287.069096
## iter 50 value 23306.759124
## iter 60 value 22879.603577
## iter 70 value 22611.002273
## iter 80 value 22367.943891
## iter 90 value 22246.994941
## iter 100 value 22188.244457
## final value 22188.244457
## stopped after 100 iterations
## - Fold2: size=5, decay=1e-01
## + Fold2: size=1, decay=1e-04
## # weights: 17
## initial value 32670.674906
## iter 10 value 30436.922685
## iter 20 value 28015.887879
## iter 30 value 27616.423098

```

```

## iter 40 value 27611.024381
## iter 50 value 27609.403156
## final value 27608.362734
## converged
## - Fold2: size=1, decay=1e-04
## + Fold2: size=3, decay=1e-04
## # weights: 49
## initial value 32620.681095
## iter 10 value 30187.837939
## iter 20 value 28859.261377
## iter 30 value 27024.473779
## iter 40 value 23014.188067
## iter 50 value 22141.232172
## iter 60 value 21685.414622
## iter 70 value 21645.150778
## iter 80 value 21626.107163
## iter 90 value 21616.199261
## iter 100 value 21610.807418
## final value 21610.807418
## stopped after 100 iterations
## - Fold2: size=3, decay=1e-04
## + Fold2: size=5, decay=1e-04
## # weights: 81
## initial value 35510.940120
## iter 10 value 30177.760783
## iter 20 value 28778.280604
## iter 30 value 27318.134501
## iter 40 value 25613.033471
## iter 50 value 23427.435372
## iter 60 value 21812.534727
## iter 70 value 21602.227416
## iter 80 value 21542.041359
## iter 90 value 21501.385016
## iter 100 value 21478.734684
## final value 21478.734684
## stopped after 100 iterations
## - Fold2: size=5, decay=1e-04
## + Fold3: size=1, decay=0e+00
## # weights: 17
## initial value 32420.252058
## iter 10 value 30069.982602
## iter 20 value 27700.465363
## iter 30 value 27674.865906
## final value 27669.239629
## converged
## - Fold3: size=1, decay=0e+00
## + Fold3: size=3, decay=0e+00
## # weights: 49
## initial value 37785.274625
## iter 10 value 30476.338440
## iter 20 value 28715.602078
## iter 30 value 25897.386241
## iter 40 value 23504.671120
## iter 50 value 22156.683989

```

```

## iter 60 value 21935.677845
## iter 70 value 21867.290469
## iter 80 value 21824.472341
## iter 90 value 21803.097925
## iter 100 value 21774.126670
## final value 21774.126670
## stopped after 100 iterations
## - Fold3: size=3, decay=0e+00
## + Fold3: size=5, decay=0e+00
## # weights: 81
## initial value 32086.848231
## iter 10 value 29864.537968
## iter 20 value 28134.002637
## iter 30 value 26606.647063
## iter 40 value 23864.673415
## iter 50 value 22245.681925
## iter 60 value 21899.104400
## iter 70 value 21772.929606
## iter 80 value 21720.232748
## iter 90 value 21666.250126
## iter 100 value 21640.388670
## final value 21640.388670
## stopped after 100 iterations
## - Fold3: size=5, decay=0e+00
## + Fold3: size=1, decay=1e-01
## # weights: 17
## initial value 31987.901697
## iter 10 value 29787.716311
## iter 20 value 28159.004786
## iter 30 value 28060.526004
## final value 28060.253815
## converged
## - Fold3: size=1, decay=1e-01
## + Fold3: size=3, decay=1e-01
## # weights: 49
## initial value 33569.907971
## iter 10 value 29373.982492
## iter 20 value 27847.854533
## iter 30 value 25618.214302
## iter 40 value 24185.103470
## iter 50 value 23204.560985
## iter 60 value 22620.550454
## iter 70 value 22473.568016
## iter 80 value 22433.103889
## iter 90 value 22430.059848
## iter 100 value 22403.139056
## final value 22403.139056
## stopped after 100 iterations
## - Fold3: size=3, decay=1e-01
## + Fold3: size=5, decay=1e-01
## # weights: 81
## initial value 32110.028028
## iter 10 value 29299.895432
## iter 20 value 27085.550147

```

```

## iter 30 value 25716.039018
## iter 40 value 23666.360506
## iter 50 value 23003.009134
## iter 60 value 22598.059737
## iter 70 value 22461.777464
## iter 80 value 22323.714981
## iter 90 value 22164.119678
## iter 100 value 22062.060983
## final value 22062.060983
## stopped after 100 iterations
## - Fold3: size=5, decay=1e-01
## + Fold3: size=1, decay=1e-04
## # weights: 17
## initial value 33095.803044
## iter 10 value 30420.342625
## iter 20 value 28393.440982
## iter 30 value 27447.373546
## iter 40 value 27232.547044
## iter 50 value 27226.253232
## final value 27226.205067
## converged
## - Fold3: size=1, decay=1e-04
## + Fold3: size=3, decay=1e-04
## # weights: 49
## initial value 32190.921777
## iter 10 value 29797.511398
## iter 20 value 28220.049168
## iter 30 value 27525.701135
## iter 40 value 25100.098959
## iter 50 value 22446.501806
## iter 60 value 22035.170055
## iter 70 value 21853.499071
## iter 80 value 21801.895454
## iter 90 value 21722.465554
## iter 100 value 21695.645680
## final value 21695.645680
## stopped after 100 iterations
## - Fold3: size=3, decay=1e-04
## + Fold3: size=5, decay=1e-04
## # weights: 81
## initial value 32112.370134
## iter 10 value 29647.836429
## iter 20 value 27834.675210
## iter 30 value 26740.865046
## iter 40 value 25984.737729
## iter 50 value 24514.735916
## iter 60 value 23689.573811
## iter 70 value 22011.060059
## iter 80 value 21425.294212
## iter 90 value 21125.057276
## iter 100 value 21019.024079
## final value 21019.024079
## stopped after 100 iterations
## - Fold3: size=5, decay=1e-04

```

```

## + Fold4: size=1, decay=0e+00
## # weights: 17
## initial value 34061.744389
## iter 10 value 30275.483322
## iter 20 value 28026.122092
## iter 30 value 27740.682267
## iter 40 value 27738.497772
## iter 50 value 27734.014897
## final value 27733.182356
## converged
## - Fold4: size=1, decay=0e+00
## + Fold4: size=3, decay=0e+00
## # weights: 49
## initial value 32449.931930
## iter 10 value 29836.445651
## iter 20 value 28164.493168
## iter 30 value 26010.239882
## iter 40 value 23772.565248
## iter 50 value 22838.408366
## iter 60 value 22625.008130
## iter 70 value 22454.569441
## iter 80 value 22225.514769
## iter 90 value 22102.319493
## iter 100 value 22040.894298
## final value 22040.894298
## stopped after 100 iterations
## - Fold4: size=3, decay=0e+00
## + Fold4: size=5, decay=0e+00
## # weights: 81
## initial value 34883.586698
## iter 10 value 29720.887542
## iter 20 value 27778.039378
## iter 30 value 25656.919703
## iter 40 value 22965.800361
## iter 50 value 22204.198996
## iter 60 value 22037.909884
## iter 70 value 21931.613945
## iter 80 value 21685.205609
## iter 90 value 21458.020632
## iter 100 value 21348.672742
## final value 21348.672742
## stopped after 100 iterations
## - Fold4: size=5, decay=0e+00
## + Fold4: size=1, decay=1e-01
## # weights: 17
## initial value 34581.706876
## iter 10 value 31059.245244
## iter 20 value 29020.765335
## iter 30 value 28219.952918
## iter 40 value 28175.836504
## final value 28175.609401
## converged
## - Fold4: size=1, decay=1e-01
## + Fold4: size=3, decay=1e-01

```

```

## # weights: 49
## initial value 34249.389001
## iter 10 value 30539.087055
## iter 20 value 29941.644894
## iter 30 value 26389.611896
## iter 40 value 24030.497317
## iter 50 value 23597.889749
## iter 60 value 23234.746388
## iter 70 value 23085.070972
## iter 80 value 23037.424283
## iter 90 value 22966.103601
## iter 100 value 22768.571302
## final value 22768.571302
## stopped after 100 iterations
## - Fold4: size=3, decay=1e-01
## + Fold4: size=5, decay=1e-01
## # weights: 81
## initial value 35900.870246
## iter 10 value 29865.139974
## iter 20 value 28437.237125
## iter 30 value 26740.636113
## iter 40 value 24584.735399
## iter 50 value 23457.684111
## iter 60 value 22949.796077
## iter 70 value 22628.083047
## iter 80 value 22477.024990
## iter 90 value 22335.094754
## iter 100 value 22273.520923
## final value 22273.520923
## stopped after 100 iterations
## - Fold4: size=5, decay=1e-01
## + Fold4: size=1, decay=1e-04
## # weights: 17
## initial value 32472.030730
## iter 10 value 30377.408819
## iter 20 value 28141.797546
## iter 30 value 27754.886211
## iter 40 value 27753.662537
## iter 50 value 27738.376525
## iter 60 value 27733.689049
## iter 60 value 27733.688794
## iter 60 value 27733.688791
## final value 27733.688791
## converged
## - Fold4: size=1, decay=1e-04
## + Fold4: size=3, decay=1e-04
## # weights: 49
## initial value 35734.313660
## iter 10 value 29851.239074
## iter 20 value 27041.457575
## iter 30 value 24092.946019
## iter 40 value 22724.359628
## iter 50 value 22456.492782
## iter 60 value 22367.808257

```

```

## iter  70 value 22343.194802
## iter  80 value 22340.016817
## iter  90 value 22337.878383
## iter 100 value 22337.299821
## final  value 22337.299821
## stopped after 100 iterations
## - Fold4: size=3, decay=1e-04
## + Fold4: size=5, decay=1e-04
## # weights:  81
## initial  value 32004.254548
## iter   10 value 30162.648637
## iter   20 value 28688.454449
## iter   30 value 26430.941771
## iter   40 value 23652.062880
## iter   50 value 22403.841678
## iter   60 value 21921.182183
## iter   70 value 21643.588475
## iter   80 value 21459.494854
## iter   90 value 21392.178045
## iter 100 value 21319.708721
## final  value 21319.708721
## stopped after 100 iterations
## - Fold4: size=5, decay=1e-04
## + Fold5: size=1, decay=0e+00
## # weights:  17
## initial  value 33184.162250
## iter   10 value 30774.965634
## iter   20 value 29372.948471
## iter   30 value 27552.981931
## iter   40 value 27259.878646
## iter   50 value 27248.077191
## iter   60 value 27248.034469
## final  value 27248.024859
## converged
## - Fold5: size=1, decay=0e+00
## + Fold5: size=3, decay=0e+00
## # weights:  49
## initial  value 37883.210256
## iter   10 value 29787.952476
## iter   20 value 26694.161070
## iter   30 value 24751.484773
## iter   40 value 23045.755283
## iter   50 value 22372.600616
## iter   60 value 22247.057708
## iter   70 value 22059.487549
## iter   80 value 21898.555583
## iter   90 value 21803.358520
## iter 100 value 21781.483742
## final  value 21781.483742
## stopped after 100 iterations
## - Fold5: size=3, decay=0e+00
## + Fold5: size=5, decay=0e+00
## # weights:  81
## initial  value 33894.961380

```

```

## iter 10 value 30640.469123
## iter 20 value 29578.346903
## iter 30 value 27706.451678
## iter 40 value 26688.965541
## iter 50 value 25381.001862
## iter 60 value 23635.283838
## iter 70 value 21826.777302
## iter 80 value 21454.396714
## iter 90 value 21356.805984
## iter 100 value 21246.112729
## final value 21246.112729
## stopped after 100 iterations
## - Fold5: size=5, decay=0e+00
## + Fold5: size=1, decay=1e-01
## # weights: 17
## initial value 32177.951253
## iter 10 value 30706.677124
## iter 20 value 28514.836374
## iter 30 value 28068.250702
## iter 40 value 27877.577086
## iter 50 value 27869.430464
## iter 60 value 27867.055406
## final value 27867.054972
## converged
## - Fold5: size=1, decay=1e-01
## + Fold5: size=3, decay=1e-01
## # weights: 49
## initial value 32798.768977
## iter 10 value 29861.791805
## iter 20 value 28269.456396
## iter 30 value 26254.994606
## iter 40 value 24228.146500
## iter 50 value 23503.080225
## iter 60 value 23100.291048
## iter 70 value 22961.350693
## iter 80 value 22895.081083
## iter 90 value 22864.994193
## iter 100 value 22840.029997
## final value 22840.029997
## stopped after 100 iterations
## - Fold5: size=3, decay=1e-01
## + Fold5: size=5, decay=1e-01
## # weights: 81
## initial value 31733.577569
## iter 10 value 29544.919723
## iter 20 value 27402.965509
## iter 30 value 25461.906413
## iter 40 value 24052.537388
## iter 50 value 23078.353061
## iter 60 value 22430.997417
## iter 70 value 22153.374538
## iter 80 value 22021.700185
## iter 90 value 21928.350346
## iter 100 value 21851.072895

```

```

## final value 21851.072895
## stopped after 100 iterations
## - Fold5: size=5, decay=1e-01
## + Fold5: size=1, decay=1e-04
## # weights: 17
## initial value 34290.044297
## iter 10 value 30119.584692
## iter 20 value 28119.490761
## iter 30 value 27323.925381
## iter 40 value 27253.149816
## iter 50 value 27252.250793
## iter 60 value 27251.797177
## final value 27251.796315
## converged
## - Fold5: size=1, decay=1e-04
## + Fold5: size=3, decay=1e-04
## # weights: 49
## initial value 35734.188274
## iter 10 value 30235.752170
## iter 20 value 28371.255029
## iter 30 value 26093.562413
## iter 40 value 22769.608922
## iter 50 value 22325.310540
## iter 60 value 22223.333447
## iter 70 value 22182.260514
## iter 80 value 22170.267212
## iter 90 value 22166.377952
## iter 100 value 22166.050934
## final value 22166.050934
## stopped after 100 iterations
## - Fold5: size=3, decay=1e-04
## + Fold5: size=5, decay=1e-04
## # weights: 81
## initial value 32559.576801
## iter 10 value 29766.049605
## iter 20 value 28001.370242
## iter 30 value 27213.884643
## iter 40 value 25488.644793
## iter 50 value 23925.975603
## iter 60 value 22749.178553
## iter 70 value 21639.240640
## iter 80 value 21384.290747
## iter 90 value 21316.544989
## iter 100 value 21240.585389
## final value 21240.585389
## stopped after 100 iterations
## - Fold5: size=5, decay=1e-04
## Aggregating results
## Selecting tuning parameters
## Fitting size = 5, decay = 0 on full training set
## # weights: 81
## initial value 41804.884100
## iter 10 value 38067.035222
## iter 20 value 35310.073224

```

```

## iter 30 value 31770.450915
## iter 40 value 28701.370638
## iter 50 value 27595.093465
## iter 60 value 27217.008782
## iter 70 value 26959.242037
## iter 80 value 26638.051341
## iter 90 value 26480.055324
## iter 100 value 26412.935047
## final value 26412.935047
## stopped after 100 iterations

rnnBalanceadoPrediccion <- predict(rnnBalanceado, testBalanceadoDatos)
rnnBalanceadoMatrizConfusion <- confusionMatrix(table(rnnBalanceadoPrediccion, testBalanceadoDatos[["Label"]]))
rnnBalanceadoMatrizConfusion

## Confusion Matrix and Statistics
##
##
## rnnBalanceadoPrediccion      b      s
##                               b 9511 2431
##                               s 2858 9938
##
##                               Accuracy : 0.7862
##                               95% CI : (0.781, 0.7913)
##                               No Information Rate : 0.5
##                               P-Value [Acc > NIR] : < 2.2e-16
##
##                               Kappa : 0.5724
##
## Mcnemar's Test P-Value : 4.695e-09
##
##                               Sensitivity : 0.7689
##                               Specificity : 0.8035
##                               Pos Pred Value : 0.7964
##                               Neg Pred Value : 0.7766
##                               Prevalence : 0.5000
##                               Detection Rate : 0.3845
##                               Detection Prevalence : 0.4827
##                               Balanced Accuracy : 0.7862
##
##                               'Positive' Class : b
##

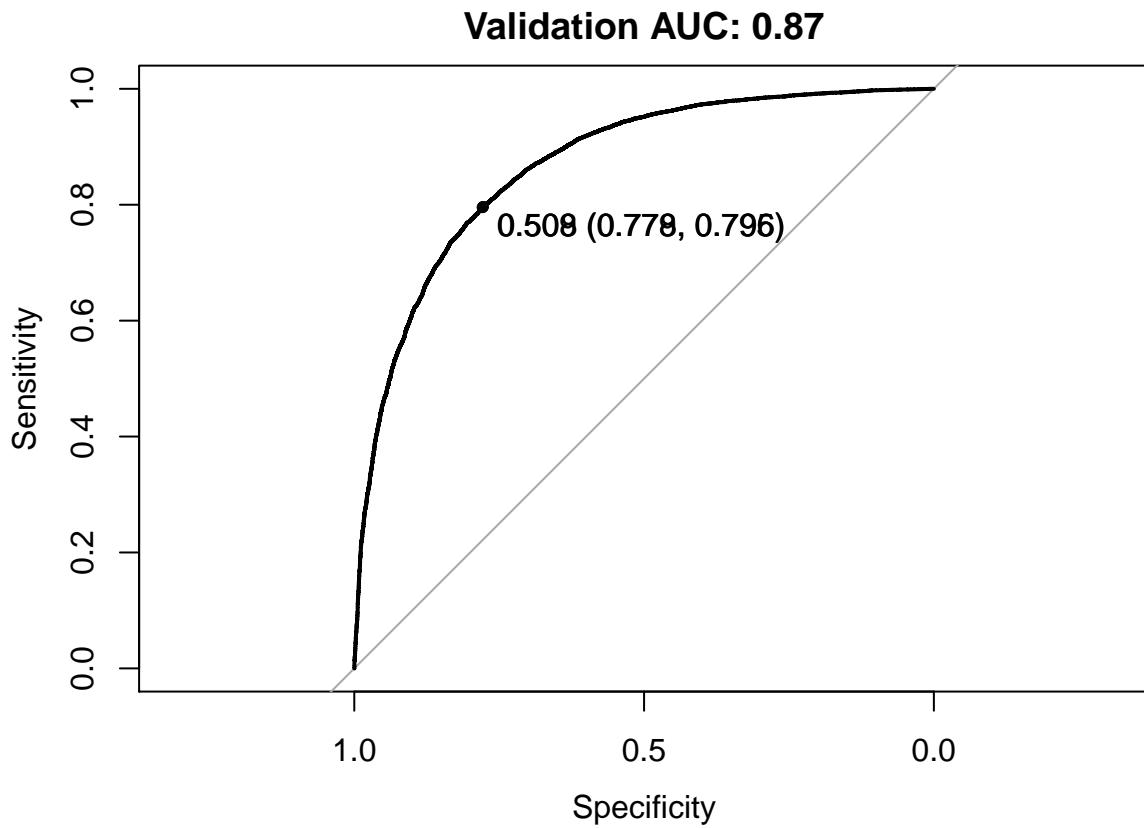
```

Podemos observar que ha empeorado el resultado, quedándose en un 77% de precisión.

```

rnnBalanceadoAUC <- predict(rnnBalanceado, testBalanceadoDatos, type = "prob")
rnnBalanceadoROC <- roc(testBalanceadoDatos$Label, rnnBalanceadoAUC[["s"]], levels = unique(testBalanceadoDatos$Label))
rnnBalanceadoROCPplot <- plot.roc(rnnBalanceadoROC, ylim=c(0,1), type = "S" , print.thres = T, main=paste("RNN", "Balanceado"))

```



Sin embargo el área bajo la curva, ha mejorado con un 0.85.

6.4 Comparativa

```
columnas <- c("Random Forest", "KNN", "Redes Neuronales Artificiales")
precision <- c("79%", "68%", "77%")
auc <- c("0.79", "0.68", "0.68")
balanceo <- c("0.88", "0.75", "0.85")

comparativa <- data.frame(columnas, precision, auc, balanceo)
knitr::kable(comparativa, "pipe")
```

columnas	precision	auc	balanceo
Random Forest	79%	0.79	0.88
KNN	68%	0.68	0.75
Redes Neuronales Artificiales	77%	0.68	0.85

7 Conclusiones

Respecto a los modelos de clasificación obtenidos con los algoritmos de aprendizaje supervisado que hemos empleado podemos decir que:

1. Ambos árboles de decisión no han llegado a obtener grandes resultados en su precisión. Quizás un ajuste en los parámetros de entrenamiento pueda variar algo este resultado, pero debido al rendimiento inicial, no creo que sean muy prósperos.
2. Los modelos de KNN sobre este problema alcanzan unos valores buenos de predicción, aunque no son

los mejores datos a obtener, sin embargo, su ejecución es algo más ligera luego ha merecido la pena realizarlo.

3. Los modelos basados en la red neuronal artificial obtiene también buenos valores de predicción. Quizás un ajuste en los parámetros de entrenamiento si puedan significar una mejora notable.
4. El mejor modelo alcanzado ha sido con **Random Forest**.

Respecto al ejercicio en sí: 1. El preprocesamiento es vital, ya que de éste depende el proceso de generalización y por tanto los modelos que obtengamos. 2. El balanceo de datos ha aportado una mejora prácticamente residual en este caso respecto a la precisión, pero sí que es cierto que hemos aumentado los valores bajo la curva, lo que nos indica que reducimos los falsos positivos y los falsos negativos, etiquetando mejor por tanto la clase final. 3. Quizás un conocimiento físico sobre el conjunto de datos pueda dar un mejor resultado al saber con conocimiento exporto, qué variables merecen más la pena, o si hay categorías que merezca la pena discretizar (siempre respetando la semántica de los datos).

8 Referencias

[1] Repositorio de la asignatura [2] Modelos y entrenamiento con caret [3] NNET con caret [4] Árboles de decisión [5] DownSampling [6] Normalize