



UNIVERSIDAD DE GRANADA



Exploración de datos del problema Higgs

Carlos Morales Aguilera

23/03/2021

Contents

1 Problema	2
1.1 Lectura de datos	3
1.2 Análisis exploratorio de los datos	3
1.3 Preprocesamiento de los datos	18
1.4 Modelos de clasificación	26
2 Alternativas	48
2.1 Utilización del conocimiento primitivo	48
2.2 Utilización del conocimiento derivado	49
2.3 Selección aleatoria de instancias	49
3 Comparativas de diferentes planteamientos	51
3.1 Planteamiento inicial:	52
3.2 Utilización de conocimiento primitivo	52
3.3 Utilización de conocimiento derivado	52
3.4 Selección aleatoria de instancias	53
3.5 Realizar un balanceo de datos	53
4 Conclusiones	53
5 Bibliografía	54

1 Problema

El problema propuesto pertenece a una competición de Kaggle sobre la identificación del bosón de Higgs con los experimento ATLAS aplicando técnicas de Machine Learning. ATLAS es un eperimento de física de partículas llevado a cabo en el Gran Colisionador de Hadrones del CERN. Este experimento busca nuevas partículas mediante la colisión frontal de protones de energía muy alta.

El objetivo del desafío propuesto es realizar un problema de aprendizaje automático en el que se examine la posibilidad de aplicar diferentes técnicas de Machine Learning para la predicción binaria de un bosón de Higgs o ruido a partir de una serie de eventos detectados por ATLAS.

Para este problema se realizaran una serie de tareas que se describen:

1. Lectura de datos.
2. Análisis exploratorio de los datos.
3. Preprocesamiento de los datos.
4. Clasificación con diversos modelos.
5. Análisis de resultados.
6. Alternativas de planteamiento del problema.
7. Conclusiones finales sobre el problema.

1.1 Lectura de datos

El código empleado tanto para la lectura de datos, como parte del código empleado para el análisis exploratorio pertenecen al profesor D. Juan Gómez Romero, de su repositorio de la asignatura de Sistemas Inteligentes para la Gestión de la empresa (SIGE) del Máster Profesional en Ingeniería Informática de la Universidad de Granada.

El primer paso es comprobar si se poseen los ficheros asociados a los datos del problema, en caso de no existir, se descargan:

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

Posteriormente se leen los datos de entrenamiento:

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

Antes de empezar a realizar el análisis exploratorio, se recodifican los valores perdidos como NA:

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

1.2 Análisis exploratorio de los datos

1.2.1 Resumen de los datos

El primer paso es realizar un análisis exploratorio inicial de los datos, para ello visualizamos un resumen inicial de los mismos:

```
summary(training_data_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   :177457      NA's   :177457
##
## 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
```

Se muestra ahora el estado de cada una de las variables del conjunto de datos:

```
kable(df_status(training_data_raw, FALSE))
```

variable	q_zeros	p_zeros	q_na	p_na	q_inf	p_inf	type	unique
EventId	0	0.00	0	0.00	0	0	numeric	250000
DER_mass_MMC	0	0.00	38114	15.25	0	0	numeric	108337
DER_mass_transverse_met_lep	3	0.00	0	0.00	0	0	numeric	101637
DER_mass_vis	0	0.00	0	0.00	0	0	numeric	100558
DER_pt_h	41	0.02	0	0.00	0	0	numeric	115563
DER_deltaeta_jet_jet	6	0.00	177457	70.98	0	0	numeric	7086
DER_mass_jet_jet	0	0.00	177457	70.98	0	0	numeric	68365
DER_prodeta_jet_jet	58	0.02	177457	70.98	0	0	numeric	16592
DER_deltar_tau_lep	0	0.00	0	0.00	0	0	numeric	4692
DER_pt_tot	39	0.02	0	0.00	0	0	numeric	59042
DER_sum_pt	0	0.00	0	0.00	0	0	numeric	156098
DER_pt_ratio_lep_tau	0	0.00	0	0.00	0	0	numeric	5931
DER_met_phi_centrality	53	0.02	0	0.00	0	0	numeric	2829
DER_lep_eta_centrality	15752	6.30	177457	70.98	0	0	numeric	1001
PRI_tau_pt	0	0.00	0	0.00	0	0	numeric	59639
PRI_tau_eta	0	0.00	0	0.00	0	0	numeric	4971
PRI_tau_phi	32	0.01	0	0.00	0	0	numeric	6285
PRI_lep_pt	0	0.00	0	0.00	0	0	numeric	61929
PRI_lep_eta	35	0.01	0	0.00	0	0	numeric	4987
PRI_lep_phi	33	0.01	0	0.00	0	0	numeric	6285
PRI_met	0	0.00	0	0.00	0	0	numeric	87836
PRI_met_phi	44	0.02	0	0.00	0	0	numeric	6285
PRI_met_sumet	0	0.00	0	0.00	0	0	numeric	179740
PRI_jet_num	99913	39.97	0	0.00	0	0	numeric	4
PRI_jet_leading_pt	0	0.00	99913	39.97	0	0	numeric	86589
PRI_jet_leading_eta	26	0.01	99913	39.97	0	0	numeric	8557
PRI_jet_leading_phi	19	0.01	99913	39.97	0	0	numeric	6284
PRI_jet_subleading_pt	0	0.00	177457	70.98	0	0	numeric	42463
PRI_jet_subleading_eta	9	0.00	177457	70.98	0	0	numeric	8627
PRI_jet_subleading_phi	10	0.00	177457	70.98	0	0	numeric	6285
PRI_jet_all_pt	99913	39.97	0	0.00	0	0	numeric	103559
Weight	0	0.00	0	0.00	0	0	numeric	104096
Label	0	0.00	0	0.00	0	0	character	2

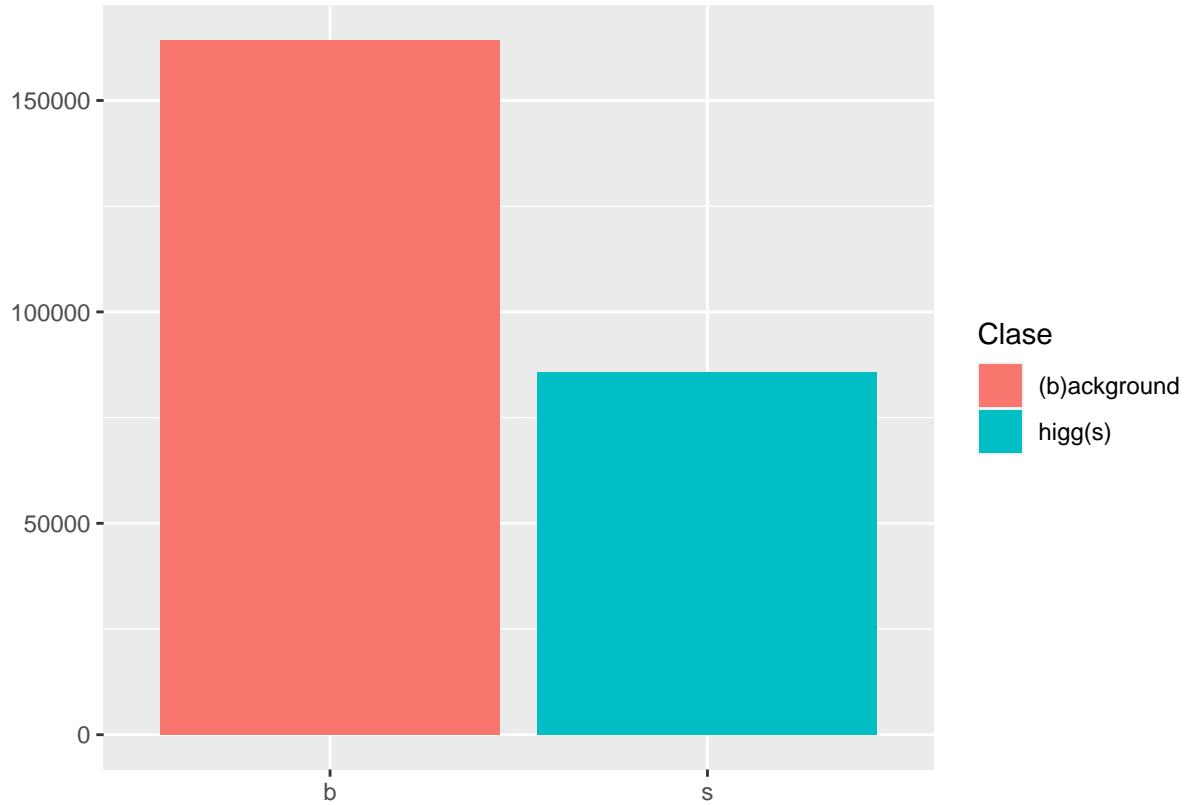
A continuación observaremos si el conjunto de datos se encuentra balanceado, observando los valores de cada clase:

```
table(training_data_raw$Label)
```

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

Se realiza una representación gráfica mediante un gráfico:

```
ggplot(training_data_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)"))
```

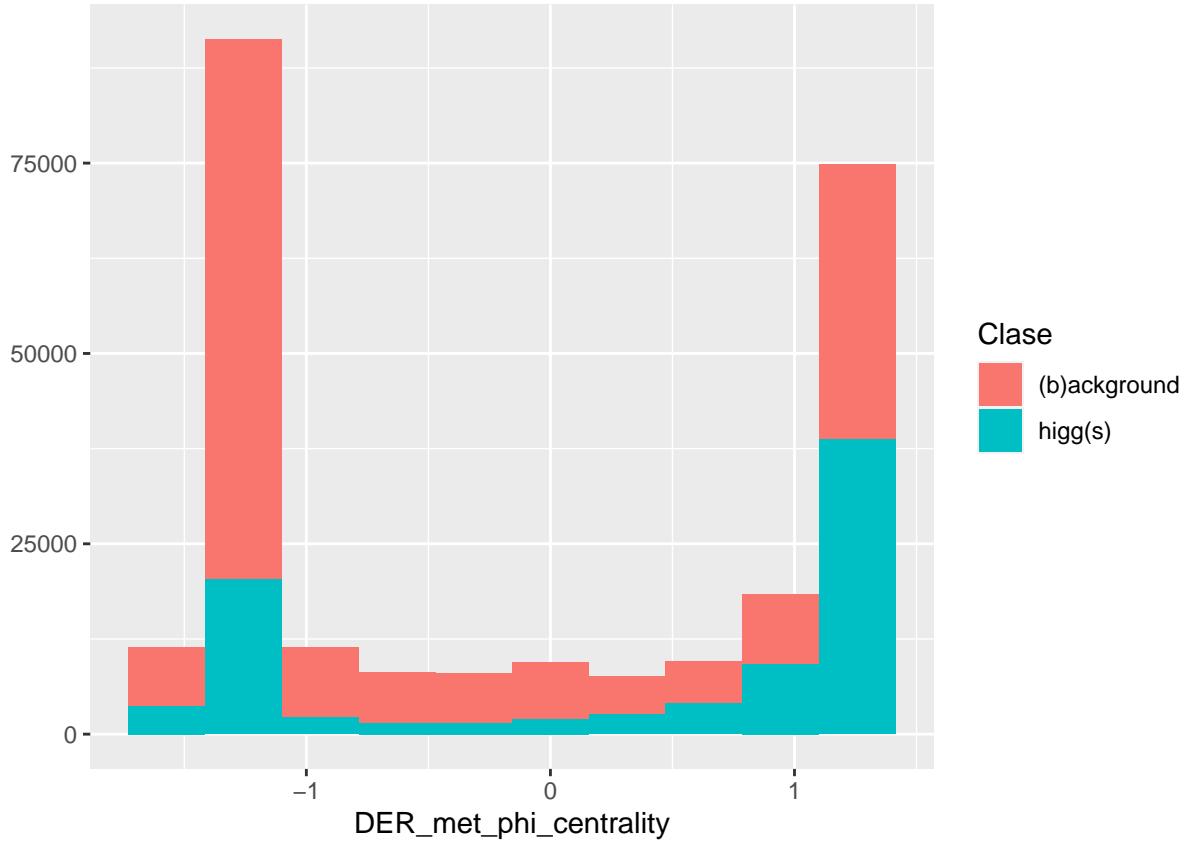


1.2.2 Estratificación

Se puede ver entonces que el conjunto de datos está claramente desbalanceado, ya que existe un mayor número de eventos asociados a ruido, lo cual es lógico con la naturaleza del proyecto ya que es más extraño encontrar un evento de un bosón de Higgs que ruido.

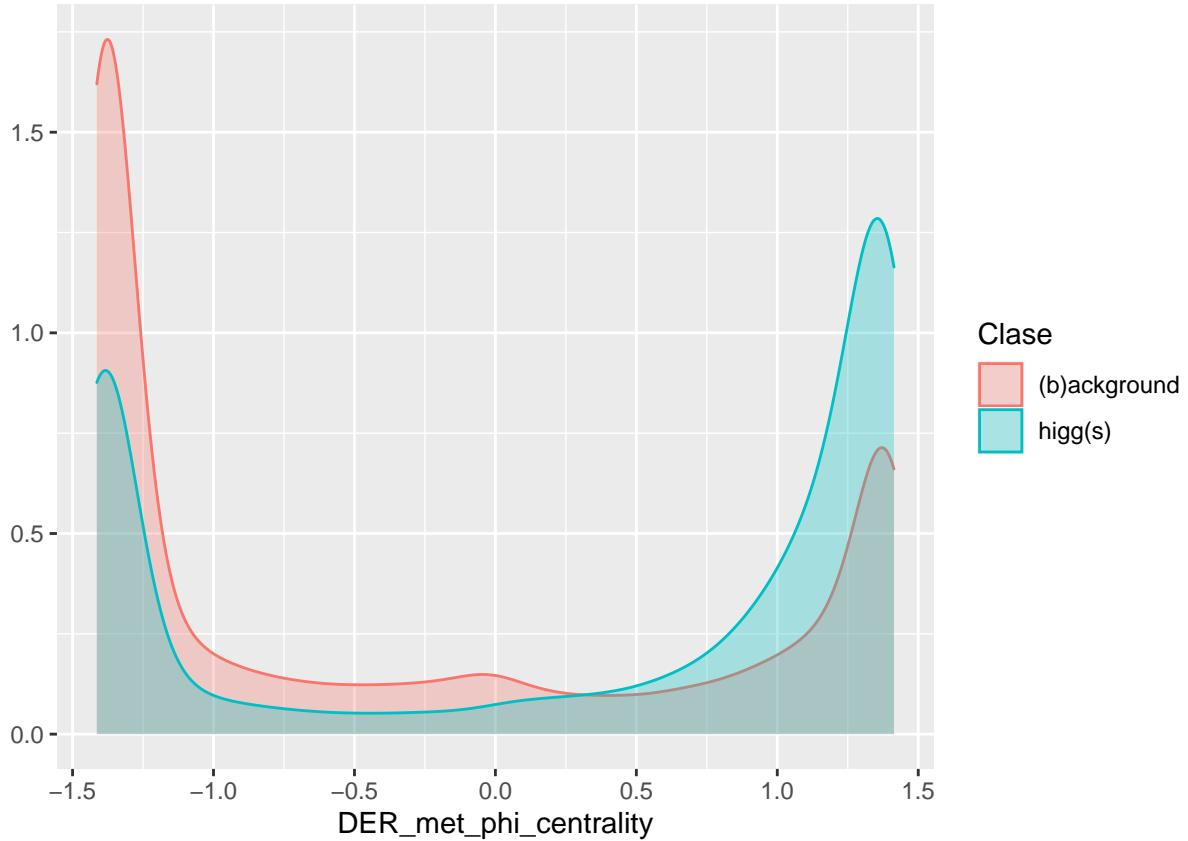
Por otro lado, también podemos observar la distribución de las clases para los diferentes valores de las distintas variables que componen el conjunto de datos, este procedimiento se conoce como *estratificación*, en este caso se analizarán las diferentes variables, pero solamente se mostrará la variable *DER_met_phi_centrality*:

```
ggplot(training_data_raw) +  
  geom_histogram(aes(x = DER_met_phi_centrality, fill = as.factor(Label)),  
                 bins = 10) +  
  labs(x = "DER_met_phi_centrality", y = "") +  
  scale_fill_discrete(name = "Clase", labels=c("(b)ackground", "higg(s)"))
```



Por otro lado se pueden ver pseudo-distribuciones de probabilidades de las diferentes variables, se examinarán todas, aunque se visualizará únicamente la variable indicada anteriormente:

```
ggplot(training_data_raw) +
  geom_density(aes(x = DER_met_phi_centrality, fill = Label, color = Label),
               alpha = 0.3) +
  labs(x = "DER_met_phi_centrality", y = "") +
  scale_fill_discrete(name = "Clase", labels=c("(b)ackground", "higg(s)")) +
  scale_color_discrete(name = "Clase", labels=c("(b)ackground", "higg(s)"))
```



A continuación se observa en las correlaciones de las diferentes variables con la clasificación final de los eventos que se estudian utilizando para ello Shiny, con todas las variables que no posean *NAs*:

Variable	Correlación
DER_mass_transverse_met_lep	-0.351427955861676
DER_mass_vis	-0.0140552737848525
DER_pt_h	0.192526328568748
DER_deltar_tau_lep	0.0122454812854829
DET_pt_tot	-0.0152874266877814
DER_sum_pt	0.153235932475814
DER_pt_ratio_lep_tau	-0.195397896182878
DER_met_phi_centrality	0.271751877051649
PRI_tau_pt	0.235237975878367
PRI_tau_eta	-0.00094325105821175
PRI_tau_phi	-0.00440253868638842
PRI_lep_pt	-0.0319475868053482
PRI_lep_eta	0.00151623537705973
PRI_lep_phi	0.00412544741152485
PRI_met	0.0224657515107859
PRI_met_phi	0.00747534218859026
PRI_met_sumet	0.135520261522685
PRI_jet_num	0.133549123081692
PRI_jet_all_pt	0.134295726669253

1.2.3 Análisis exploratorio con DataExplorer

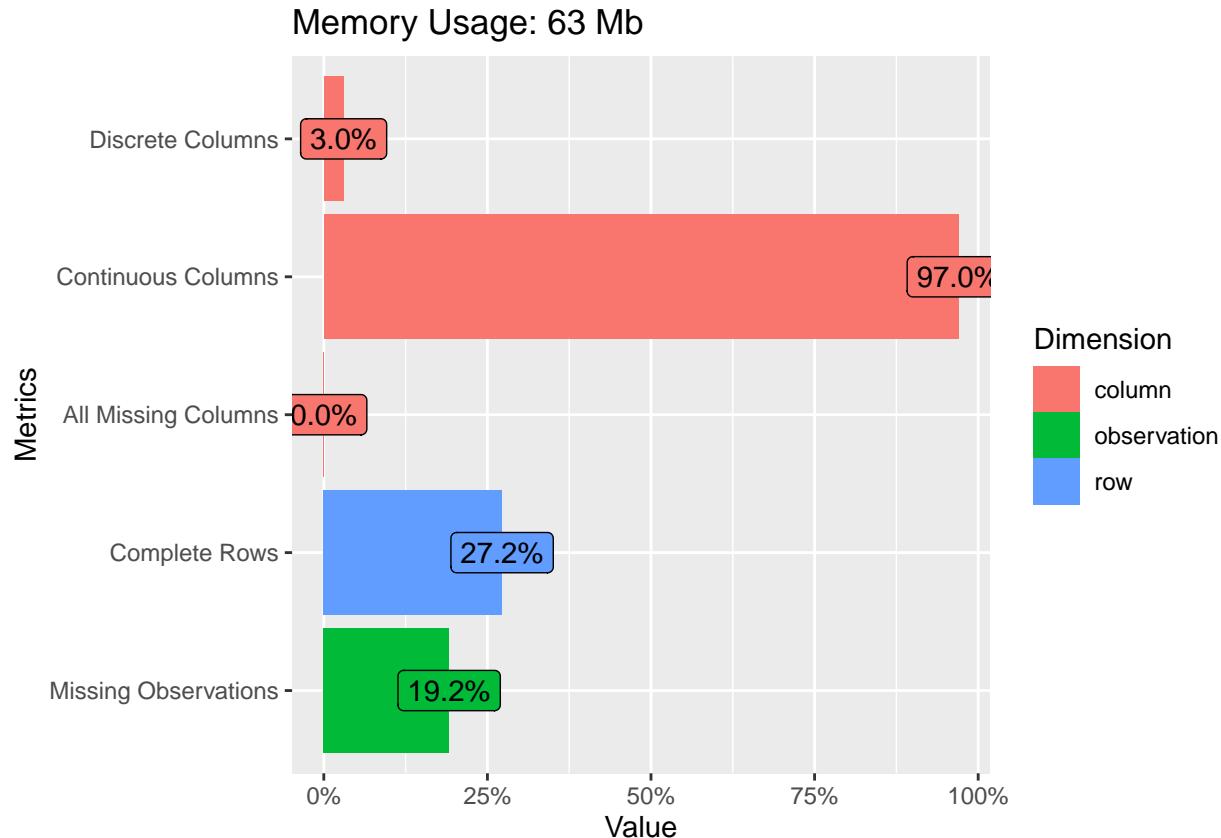
Por otro lado, se puede llevar a cabo un informe de un análisis exploratorio de los datos utilizando la librería DataExplorer con la que podemos observar diferentes gráficos que informan sobre el conjunto de datos:

Podemos observar por lo tanto una información inicial del conjunto de datos y los valores perdidos que se poseen:

Name	Value
Rows	250,000
Columns	33
Discrete columns	1
Continuous columns	32
All missing columns	0
Missing observations	1,580,052
Complete Rows	68,114
Total observations	8,250,000
Memory allocation	63 Mb

A continuación observamos dicha información de modo gráfico:

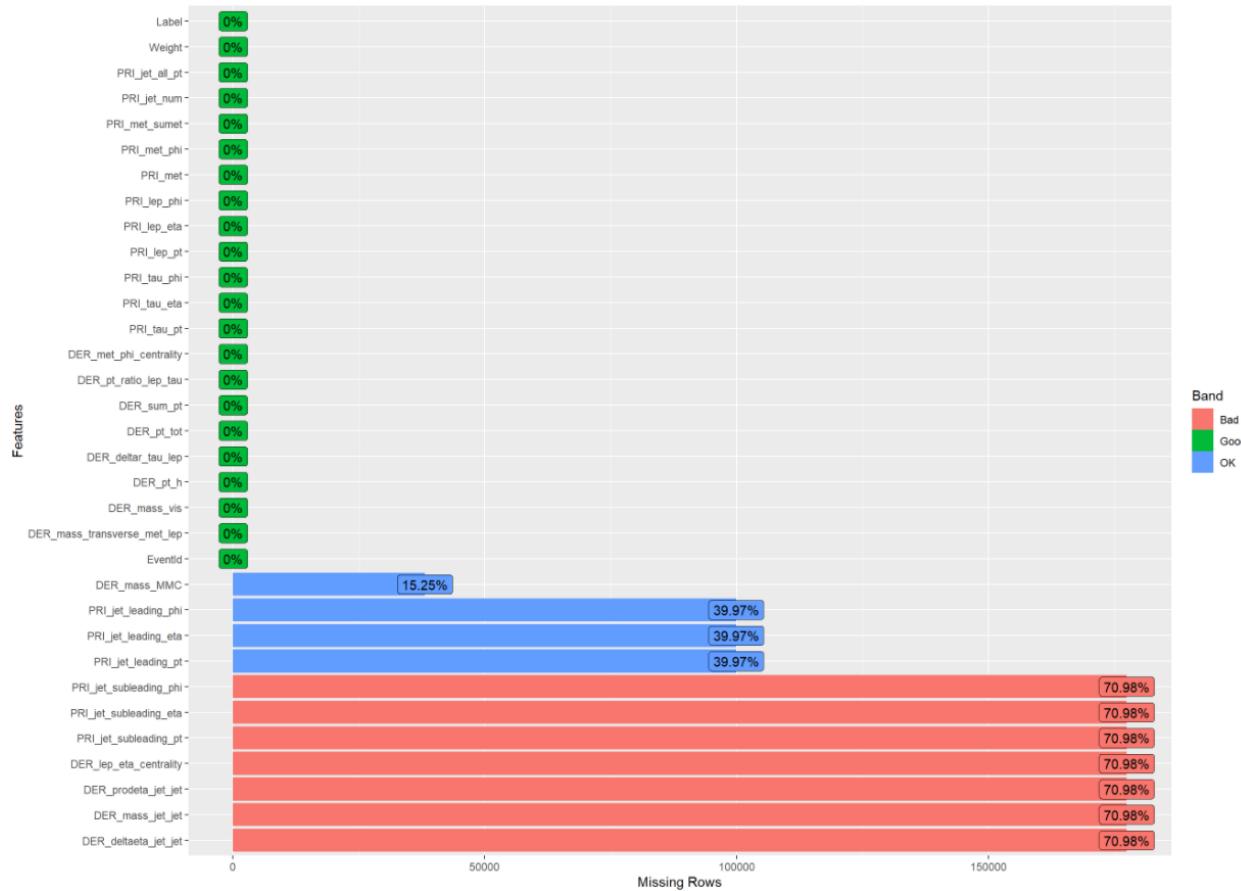
```
plot_intro(training_data_raw)
```



Como es lógico, no trabajaremos con los valores perdidos, por los que posteriormente todos los datos codificados como *NAs* serán descartados.

A continuación, para analizar dichos valores perdidos observamos en que columnas se encuentran y poder sacar posteriores conclusiones:

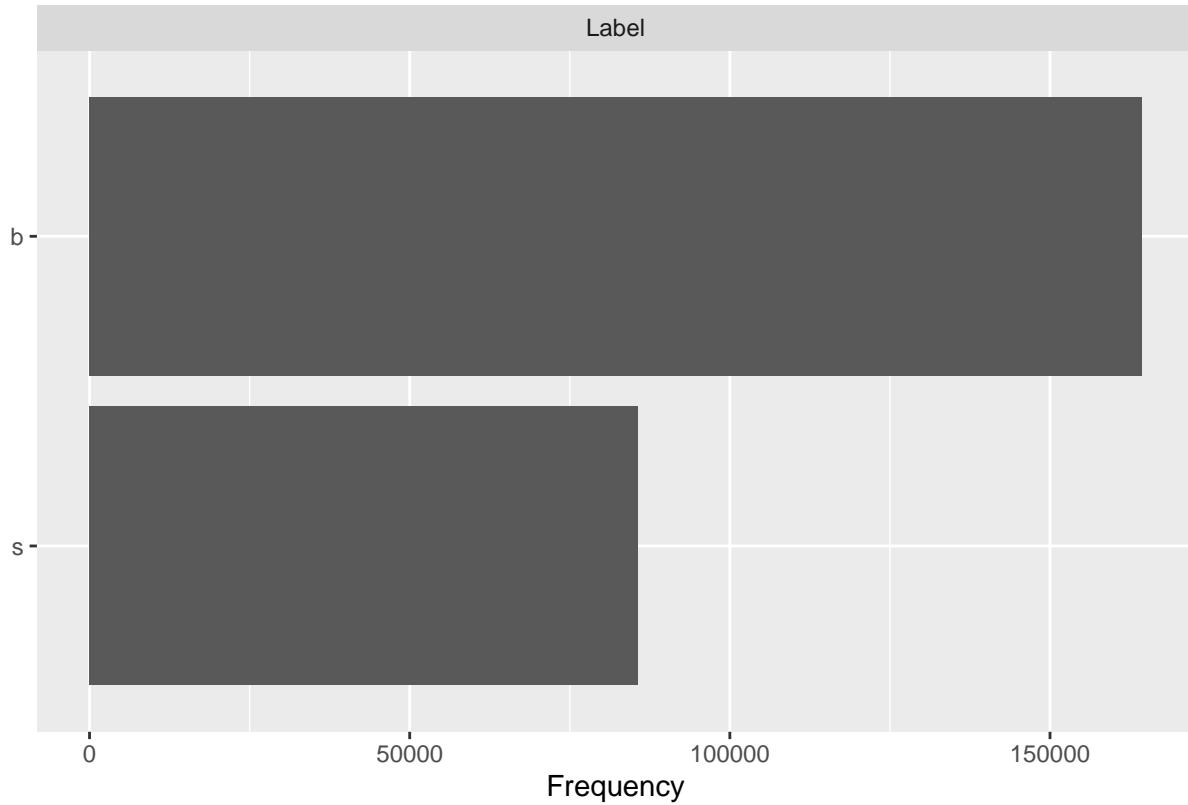
```
plot_missing(training_data_raw)
```



Se puede observar que se poseen más pérdidas en variables asociadas a los jets tras observar el gráfico y comprender dichas variables según la documentación proporcionada.

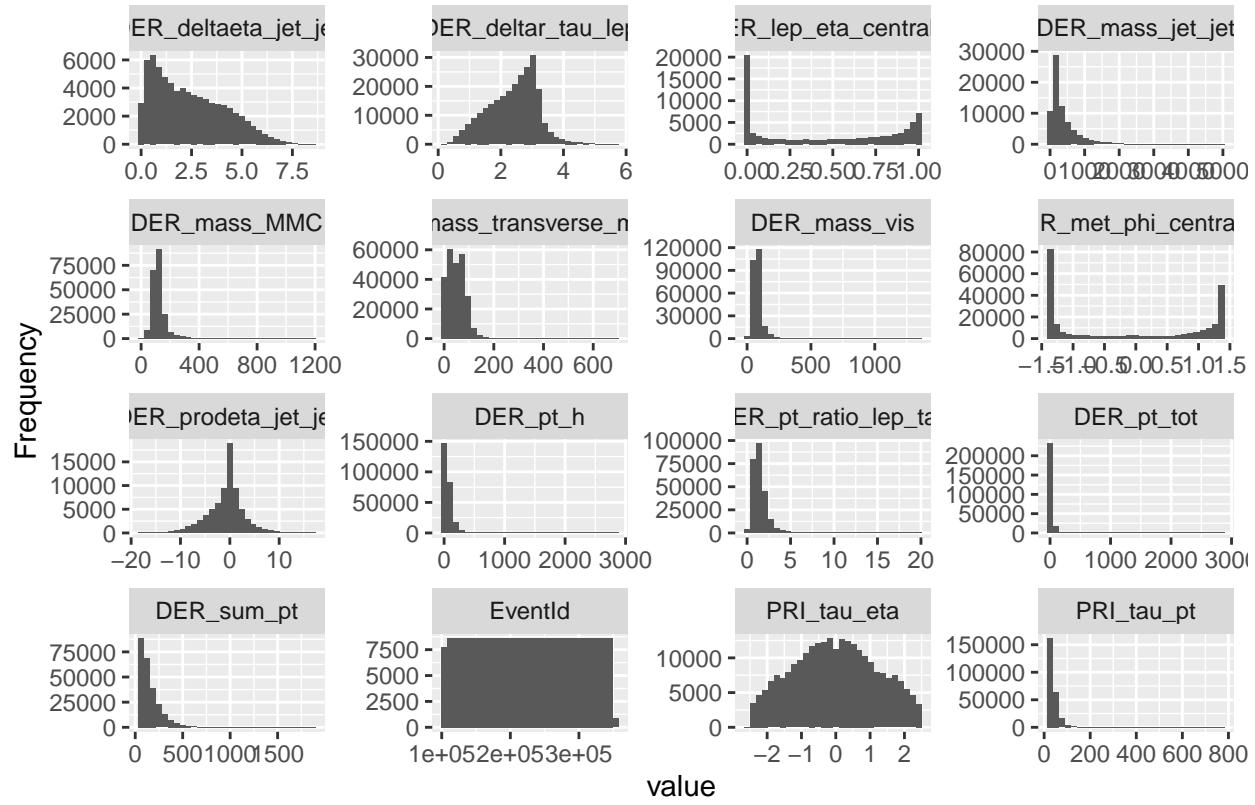
A continuación vemos de nuevo la proporción de datos, que como es lógico y hemos visto antes, existen más casos de ruido (background) que de bosones (Higgs).

```
plot_bar(training_data_raw)
```

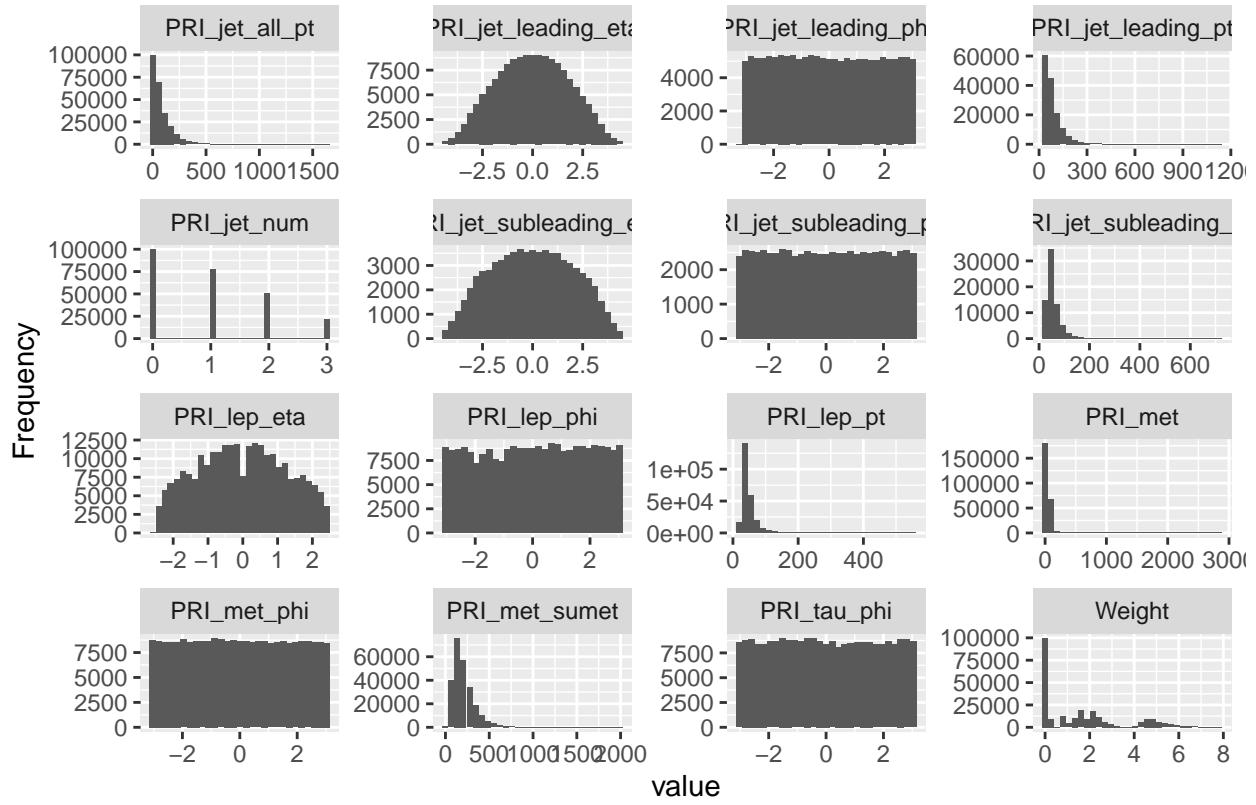


A continuación, analizando más en profundidad las distintas variables que posee el conjunto de datos podemos ver como se distribuyen utilizando para ello gráficos de histogramas por cada una de las variables disponibles:

```
plot_histogram(training_data_raw)
```



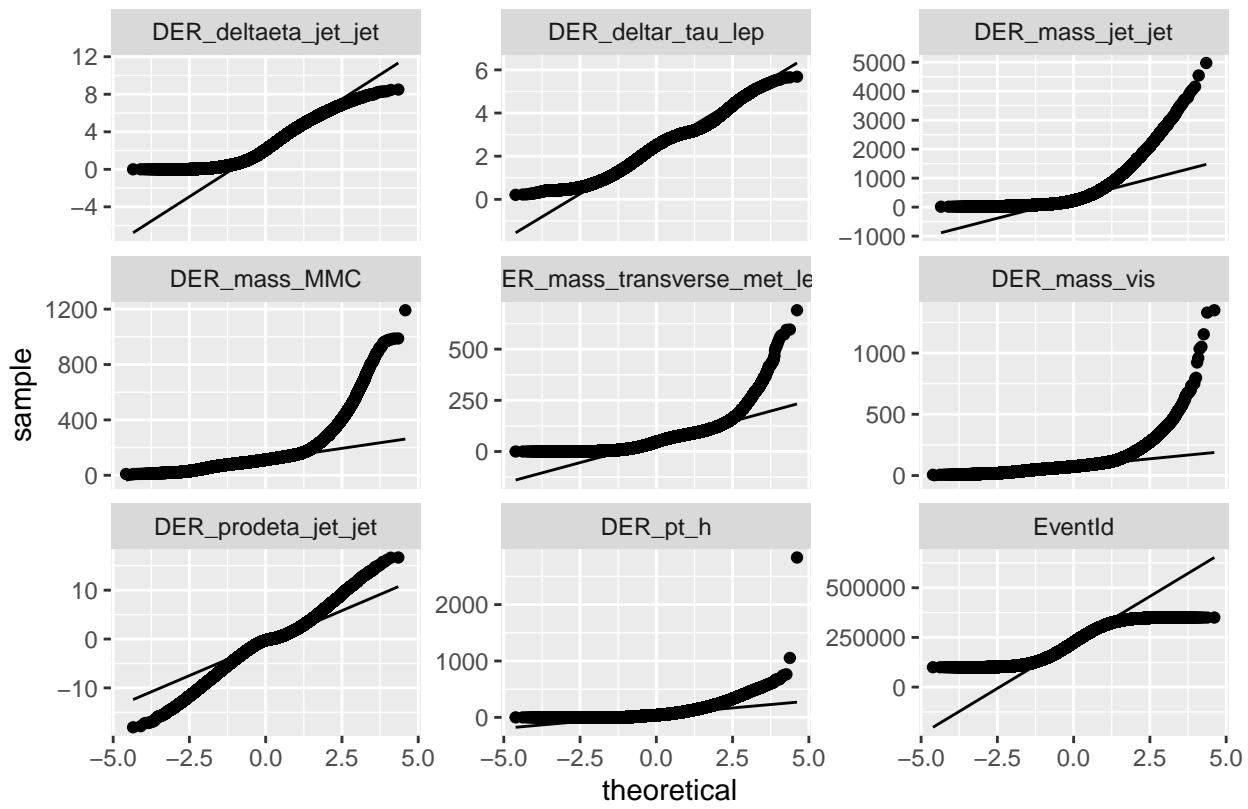
Page 1



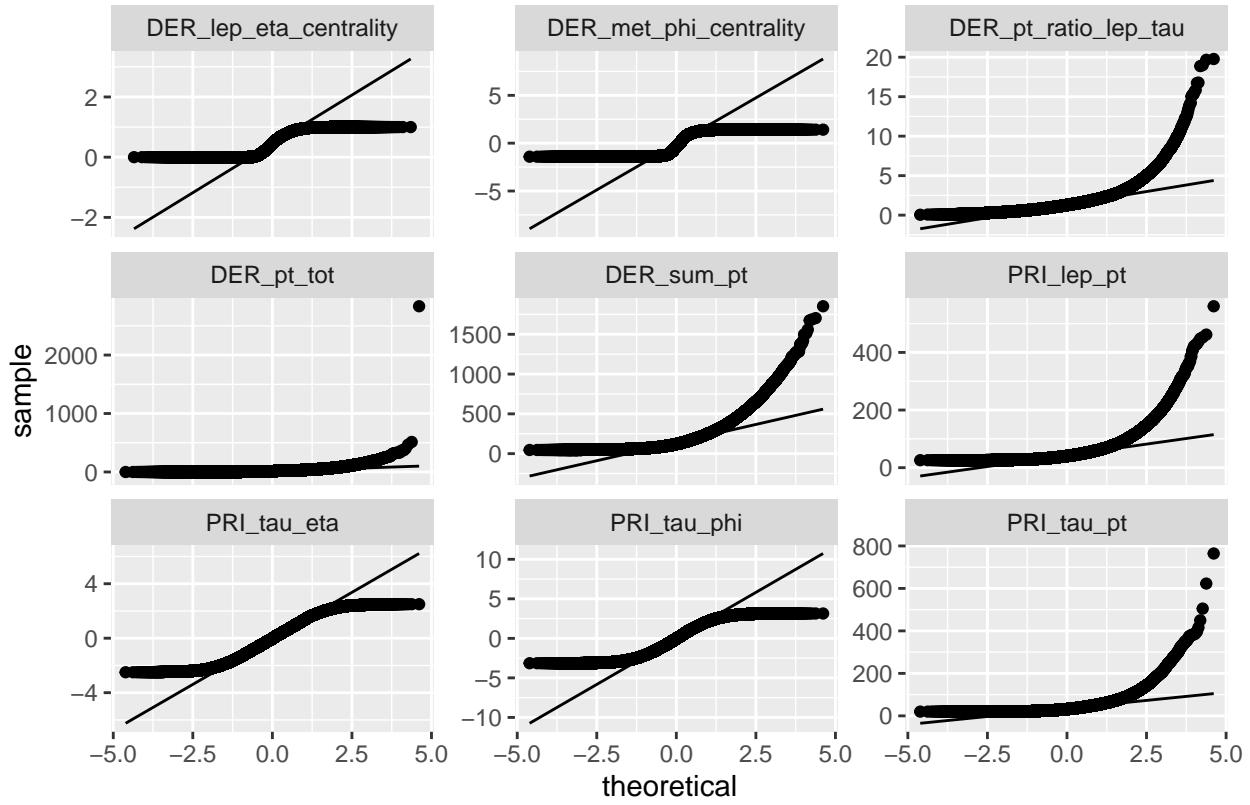
Page 2

Podemos ver que la distribución de los datos no es uniforme, cosa que podemos corroborar utilizando gráficos de quantiles:

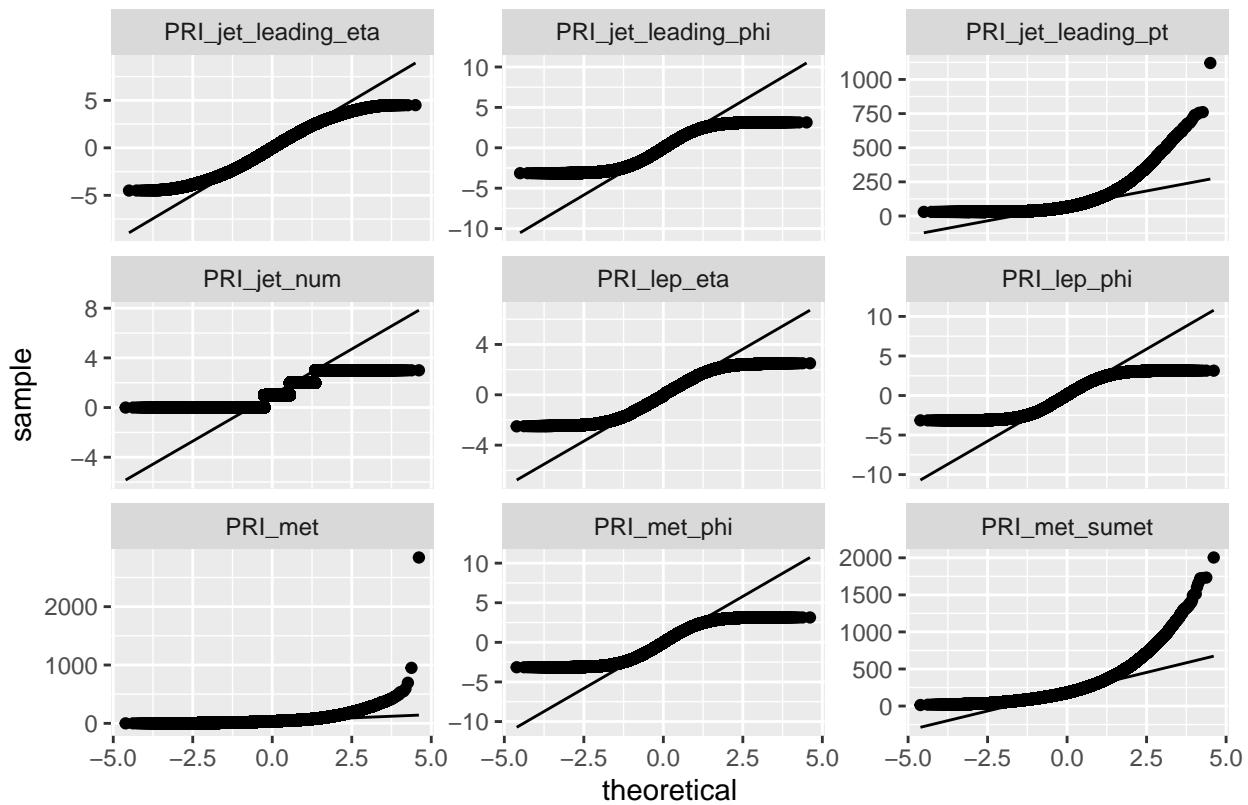
```
plot_qq(training_data_raw)
```



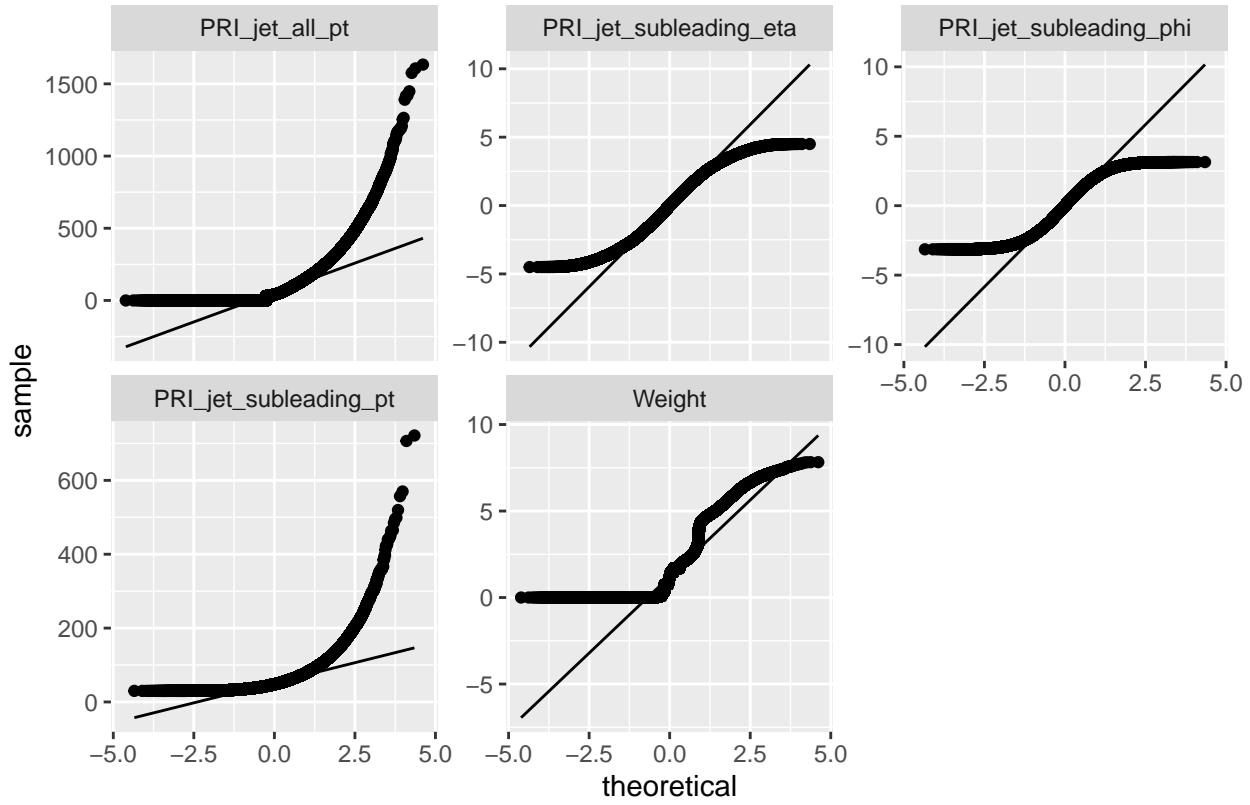
Page 1



Page 2



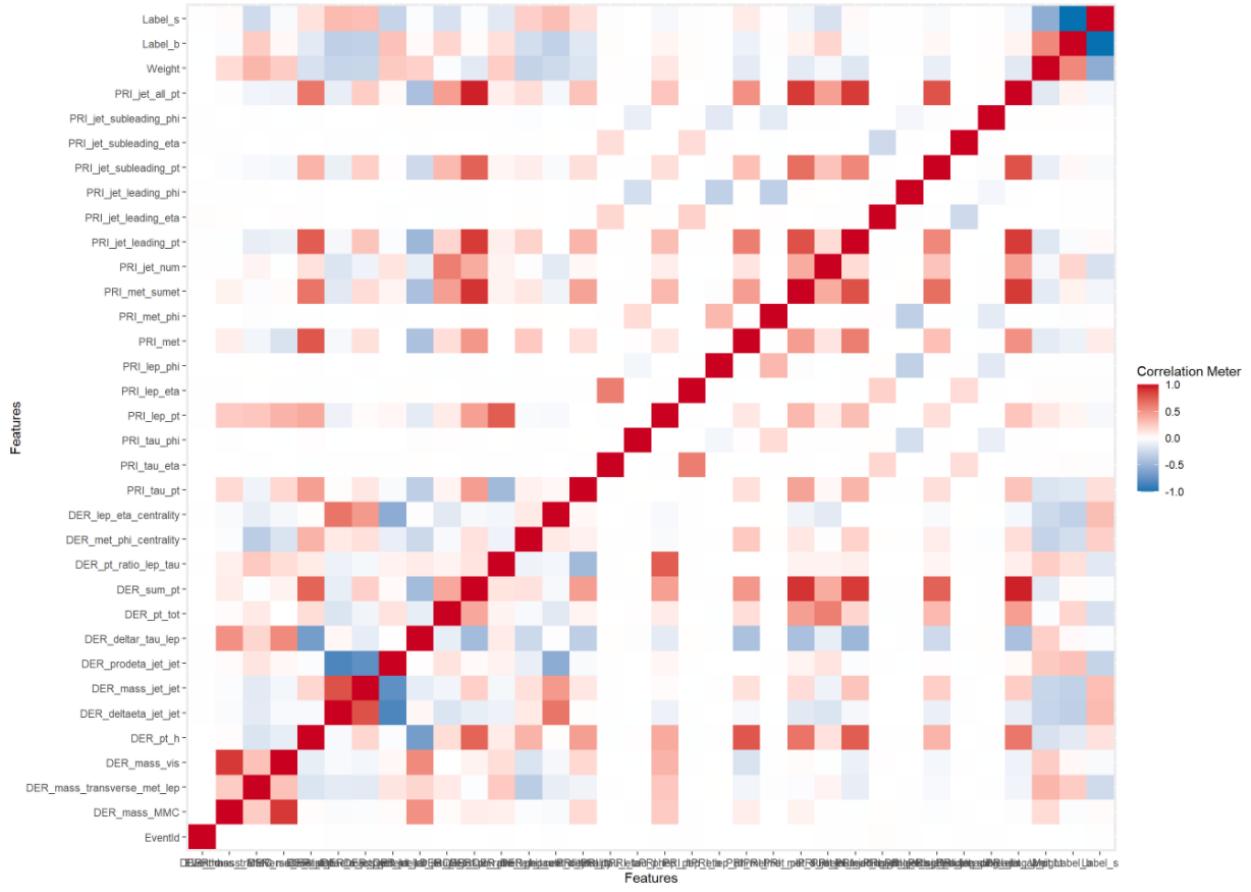
Page 3



Page 4

Por último, podemos observar la correlación de las diferentes variables utilizando una matriz de correlación que se visualice de forma gráfica:

```
plot_correlation(training_data_raw)
```



Como es lógico podemos observar que las variables derivadas están relacionadas con sus variables primitivas, y variables que operan sobre conceptos similares a su vez también poseen un gran grado de correlación.

El siguiente paso en este trabajo será realizar un preprocesamiento correcto en base a lo observado tras este análisis inicial, omitiendo valores perdidos, tratando de balancear clases y analizando las variables. Tras este primer análisis exploratorio se ha podido observar que las clases están desbalanceadas, que la distribución de los datos no es uniforme y que existen determinadas variables que poseen una mayor correlación con la clasificación final de los diferentes eventos. Estos serán los factores iniciales que determinen un camino de cara al preprocesamiento de los datos.

1.3 Preprocesamiento de los datos

1.3.1 Omisión de características inservibles

Tras leer la descripción del problema podemos comprobar que la característica *EventId* no es útil para el problema ya que solo representa el registro mediante identificador del evento. Por otro lado, la característica *Weight* no aporta información al problema de cara a la clasificación, por lo que también queda descartada:

```
training_data_raw[,"EventId"] <- NULL
training_data_raw[,"Weight"] <- NULL
```

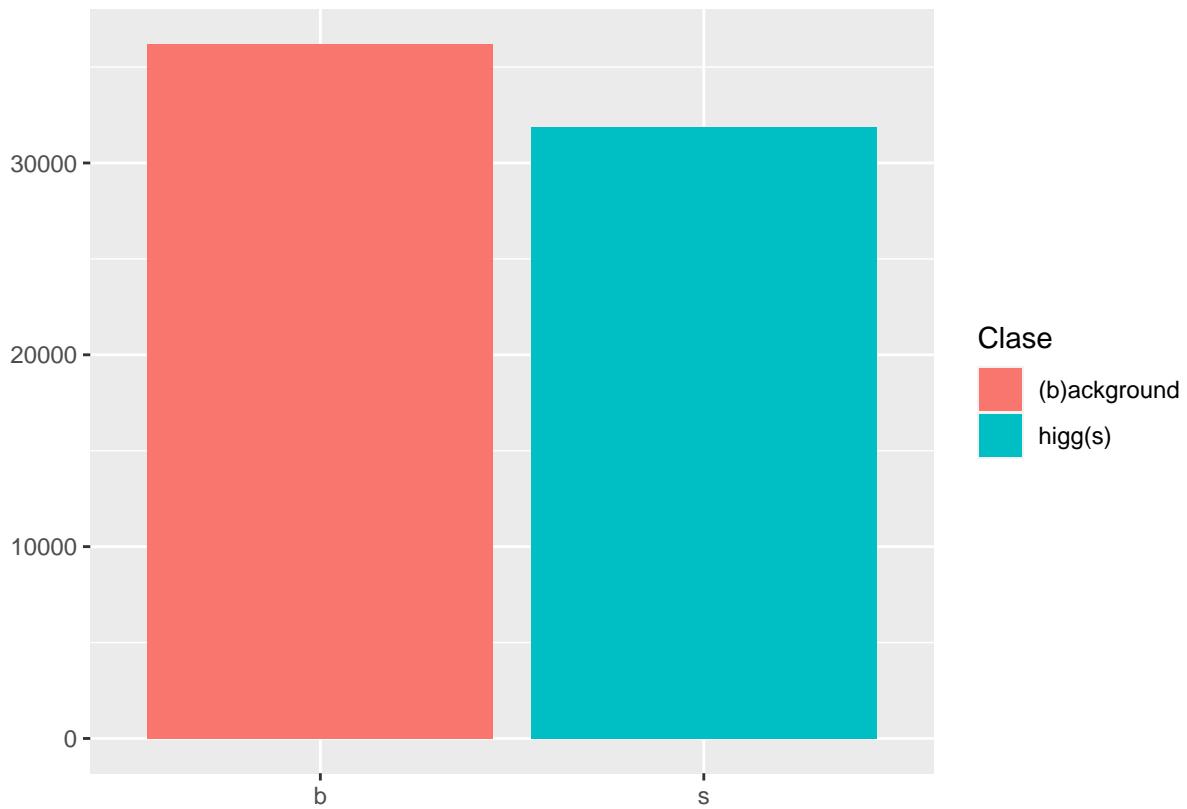
1.3.2 Omisión de valores perdidos

Uno de los preprocesamientos más comunes y prácticos es la omisión de valores perdidos, descartando por lo tanto eventos de los cuales no se conocen todos los datos, ya que al tratarse de datos parciales, no podemos obtener toda la información para poder realizar un modelo predictivo correcto en base a todos los datos aportados, por lo que para el estudio de modelos, descartaremos los datos con valores perdidos.

```
training_data_raw <- na.omit(training_data_raw)
```

Como podemos observar pasamos de un conjunto de datos de 250.000 eventos a un conjunto mucho más reducido pero suficientemente grande de 68.114, a continuación veremos el balanceo de las clases tras esta omisión de datos:

```
ggplot(training_data_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)"))
```



Podemos ver que pese a encontrarse desbalanceados, es mucho menor el desbalanceo de los datos obtenidos, lo cual es positivo de cada al análisis de los datos, pero a su vez por falta de información podría llevar a un modelo erróneo, pese a que los datos eliminados tuvieran valores perdidos.

1.3.3 Detección de outliers

A continuación, se procede a evaluar los posibles *outliers* de las variables numéricas, para ello se va a emplear la denominada *Distancia de Cook* (tal y como se ve en el tutorial <http://r-statistics.co/Outlier-Treatment.html>)

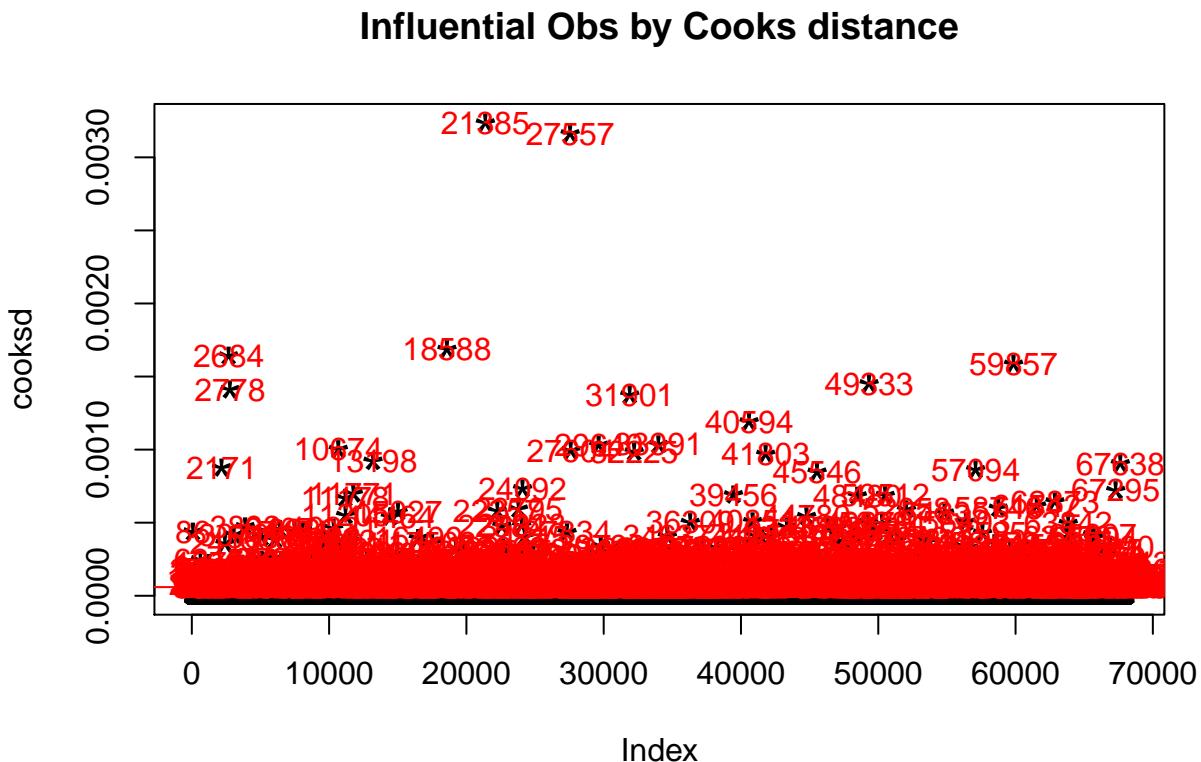
With-R.html). Para ello se utiliza un enfoque multivariable como el que se posee en el problema y generando un modelo de regresión, como puede ser un modelo lineal, calcula la importancia de cada uno de los puntos para este modelo con sus respectivas variables, esto permite detectar puntos demasiado importantes que por lo tanto consideraremos *outliers*.

```
training_data_raw$Label <- as.factor(training_data_raw$Label)

mod <- lm(as.numeric(Label)~., data=training_data_raw)

cooksrd <- cooks.distance(mod)

plot(cooksrd, pch="*", cex=2, main="Influential Obs by Cooks distance")
abline(h = 4*mean(cooksrd, na.rm=T), col="red")
text(x=1:length(cooksrd)+1, y=cooksrd,
      labels=ifelse(cooksrd>4*mean(cooksrd, na.rm=T),
                    names(cooksrd), ""), col="red")
```



```
influential <- as.numeric(names(cooksd)[(cooksd > 4*mean(cooksd, na.rm=T))])
training_data_raw[influential, ] # influential observations.
```

```

## # A tibble: 1,868 x 31
##   DER_mass_MMC DER_mass_transv~ DER_mass_vis DER_pt_h DER_deltaeta_j~
##       <dbl>           <dbl>           <dbl>      <dbl>           <dbl>
## 1     138.            38.0          99.0      65.0          1.81
## 2     119.            2.34          72.1      91.6          0.313

```

```

## 3      282.      47.9      225.      154.      5.76
## 4      99.6      10.4      81.4      436.      1.41
## 5     90.0      41.9      41.9      379.      2.80
## 6     115.      34.3      73.3      229.      1.00
## 7     114.      12.8      41.5      300.      0.765
## 8     342.      32.9      286.      334.      2.57
## 9     720.      47.6      209.      130.      0.584
## 10    394.      99.4      204.      113.      1.30
## # ... with 1,858 more rows, and 26 more variables: DER_mass_jet_jet <dbl>,
## #   DER_prodeta_jet_jet <dbl>, DER_deltar_tau_lep <dbl>, DER_pt_tot <dbl>,
## #   DER_sum_pt <dbl>, DER_pt_ratio_lep_tau <dbl>, DER_met_phi_centrality <dbl>,
## #   DER_lep_eta_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_subleading_pt <dbl>,
## #   PRI_jet_subleading_eta <dbl>, PRI_jet_subleading_phi <dbl>,
## #   PRI_jet_all_pt <dbl>, Label <fct>

```

```
training_data_raw <- training_data_raw[-influential,]
```

1.3.4 Discretización de variables

Una de las técnicas principales de preprocesamiento de datos consiste en la discretización de valores continuos, o discretización de valores categóricos como agrupación de estas categorías en otras que abarquen más categorías en una misma.

Sin embargo, en el problema que estamos tratando, todas las variables son numéricas, y realizar una discretización lógica conllevaría un mayor aprendizaje del problema para poder tomar decisiones coherentes en cuanto a la agrupación en diferentes categorías, por lo que se ha optado por descartar una discretización de las variables.

1.3.5 Tratamiento del ruido

Otro de los principales problemas son aquellas etiquetas que pueden suponer un aumento de la complejidad del problema por lo que se consideran ruido. Este ruido puede ser tratado mediante diferentes filtros que eliminan dichos eventos y que facilitan el posterior aprendizaje de los diferentes modelos de clasificación.

Para ello se utiliza la función edgeBoostFilter del paquete NoiseFilterR el cual permite definir una serie de filtros y limpia los datos del ruido existente en las etiquetas.

```
library(NoiseFiltersR)
training_data_raw <- edgeBoostFilter(Label~, training_data_raw)
```

```
## Warning in edgeBoostFilter.default(x = modFrame, ..., classColumn = 1): AdaBoost
## process stopped at iteration 4 because error exceeded 0.5
```

```
training_data_raw <- training_data_raw$cleanData
```

1.3.6 Normalización de variables

Otra de las técnicas principales consiste en la normalización de las variables numéricas dentro de unos límites más reducidos y trasladando sus valores. Este procedimiento se conoce como *normalización* y existen diferentes tipos. En este caso se empleará uno de los tipos más comunes, consistente en una normalización del tipo *min_max*, el cual establece entre límites de 0 y 1 todos los valores, siendo el menor de ellos el 0 y el mayor el 1.

Para ello utilizaremos una implementación de la función manual:

```
# Definition of min_max normalization function
min_max_norm <- function(x) {
  (x -min(x)) / (max(x) - min(x))
}
```

A continuación se normalizan todas las variables numéricas:

```
# Normalize variables
for(i in 1:length(training_data_raw[-1]))
  training_data_raw[,i] <- min_max_norm(training_data_raw[,i])
```

1.3.7 Selección de características

Una de las técnicas más útiles a la hora de reducir el tamaño del problema es la selección de características (variables), por lo que para ello se entrenará un *modelo lineal* y se observarán qué variables son necesarias para la construcción del modelo.

Se procede a realizar el **Modelo lineal**. Para ello primero entrenamos el modelo con un modelo lineal utilizando la función lm.

```
# Train the linear model
model <- lm(as.numeric(Label) ~ ., data = training_data_raw)
model

##
## Call:
## lm(formula = as.numeric(Label) ~ ., data = training_data_raw)
##
## Coefficients:
##              (Intercept)                  DER_mass_MMC
##                         2.144e+00                 2.235e-01
##              DER_mass_transverse_met_lep      DER_mass_vis
##                         -1.589e+00                -1.726e+00
##              DER_pt_h                      DER_deltaeta_jet_jet
##                         1.017e+00                 2.689e-02
##              DER_mass_jet_jet            DER_prodeta_jet_jet
##                         8.056e-01                -9.015e-02
##              DER_deltar_tau_lep          DER_pt_tot
##                         1.095e+00                 -5.065e-01
##              DER_sum_pt                   DER_pt_ratio_lep_tau
##                         4.806e+02                 -1.481e+00
##              DER_met_phi_centrality     DER_lep_eta_centrality
##                         1.925e-01                 2.258e-01
```

```

##          PRI_tau_pt             PRI_tau_eta
##          -1.077e+02            -1.054e-02
##          PRI_tau_phi           PRI_lep_pt
##          8.824e-04              -1.142e+02
##          PRI_lep_eta           PRI_lep_phi
##          -1.343e-02              7.473e-03
##          PRI_met                PRI_met_phi
##          9.232e-01              4.610e-03
##          PRI_met_sumet          PRI_jet_num
##          -9.311e-01             -8.855e-02
##          PRI_jet_leading_pt      PRI_jet_leading_eta
##          -9.288e-01              1.004e-02
##          PRI_jet_leading_phi     PRI_jet_subleading_pt
##          6.503e-03              3.687e-01
##          PRI_jet_subleading_eta  PRI_jet_subleading_phi
##          1.753e-02              -9.876e-03
##          PRI_jet_all_pt
##          -4.265e+02

```

A continuación construimos un modelo de regresión para establecer una variables predictoras añadiendo y eliminando predictores basandonos en el valor p, para ello utilizamos la función `ols_step_both_p` del paquete `olsrr`:

```

# Regression model
k <- ols_step_both_p(model)
summary(k$model)

##
## Call:
## lm(formula = paste(response, "~", paste(preds, collapse = " + ")),
##     data = 1)
##
## Residuals:
##       Min     1Q     Median     3Q    Max
## -1.05005 -0.32194 -0.02117  0.31850  1.08510
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.038411  0.020093 51.680 < 2e-16 ***
## DER_mass_transverse_met_lep -1.591177  0.040551 -39.239 < 2e-16 ***
## DER_pt_h                   1.017833  0.072928 13.957 < 2e-16 ***
## DER_mass_jet_jet           0.830115  0.025897 32.055 < 2e-16 ***
## DER_protdeta_jet_jet      -0.108936  0.027311 -3.989 6.65e-05 ***
## DER_pt_tot                  -0.497866  0.026415 -18.848 < 2e-16 ***
## DER_pt_ratio_lep_tau        -1.482579  0.056365 -26.303 < 2e-16 ***
## DER_met_phi_centrality      0.192632  0.006371 30.235 < 2e-16 ***
## DER_lep_eta_centrality      0.228636  0.004931 46.368 < 2e-16 ***
## PRI_tau_pt                  1.500227  0.049616 30.237 < 2e-16 ***
## PRI_jet_num                 -0.085625  0.004527 -18.916 < 2e-16 ***
## DER_deltar_tau_lep          1.096726  0.025707 42.663 < 2e-16 ***
## PRI_jet_leading_pt          -0.886210  0.034940 -25.364 < 2e-16 ***
## PRI_lep_pt                  2.172234  0.059745 36.358 < 2e-16 ***
## DER_mass_vis                 -1.727977  0.110032 -15.704 < 2e-16 ***

```

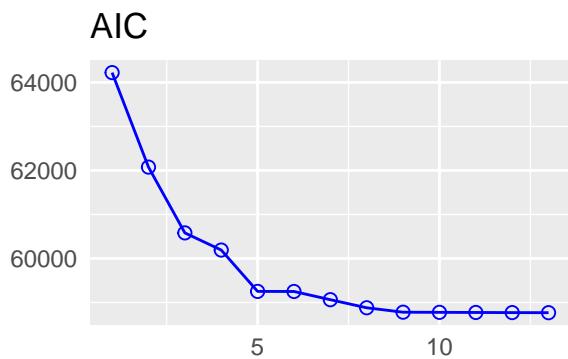
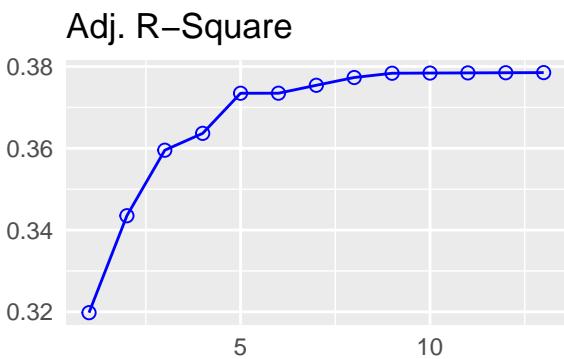
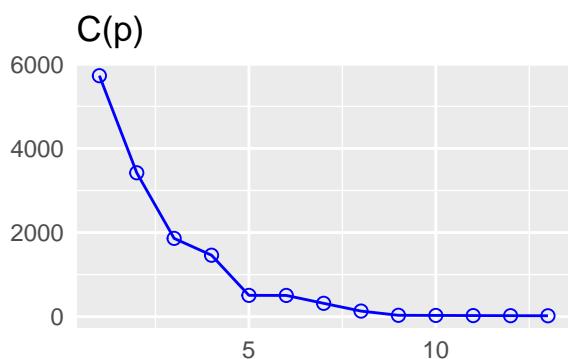
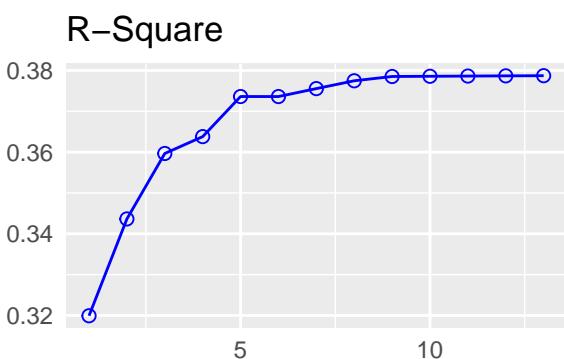
```

## PRI_met          0.921167  0.079807 11.542 < 2e-16 ***
## PRI_met_sumet   -0.921788 0.055471 -16.618 < 2e-16 ***
## PRI_jet_subleading_pt 0.401594  0.039837 10.081 < 2e-16 ***
## PRI_lep_eta     -0.017484 0.006715 -2.604 0.00922 **
## DER_mass_MMC    0.222415  0.098579  2.256 0.02406 *
## PRI_jet_subleading_phi -0.011987 0.005523 -2.170 0.02999 *
## PRI_jet_subleading_eta  0.014043  0.007095  1.979 0.04780 *
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3931 on 60522 degrees of freedom
## Multiple R-squared: 0.3787, Adjusted R-squared: 0.3785
## F-statistic: 1757 on 21 and 60522 DF, p-value: < 2.2e-16

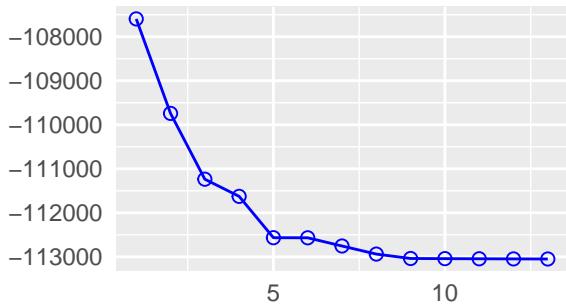
```

```
plot(k)
```

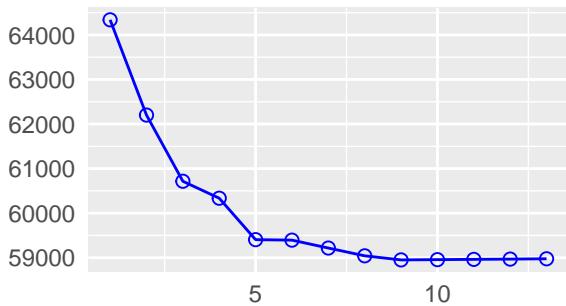
page 1 of 2



SBIC



SBC



Por lo tanto el modelo final contendrá las siguientes variables:

Variables
DER_mass_transverse_met_lep
DER_pt_h
DER_deltaeta_jet_jet
DER_mass_jet_jet
DER_pt_tot
DER_pt_ratio_lep_tau
DER_met_phi_centrality
DER_lep_eta_centrality
PRI_tau_pt
PRI_jet_num
DER_deltar_tau_lep
PRI_jet_leading_pt
DER_mass_vis
PRI_lep_pt
PRI_met_sumet
PRI_met
PRI_jet_subleading_pt
PRI_lep_eta
PRI_jet_subleading_phi
PRI_jet_subleading_eta
PRI_jet_leading_eta

Eliminamos por lo tanto las variables que no se utilizan:

```
training_data <- training_data_raw
training_data[,"DER_mass_MMC"] <- NULL
training_data[,"DER_prodeta_jet_jet"] <- NULL
training_data[,"DER_sum_pt"] <- NULL
training_data[,"PRI_tau_eta"] <- NULL
training_data[,"PRI_tau_phi"] <- NULL
training_data[,"PRI_lep_phi"] <- NULL
training_data[,"PRI_met_phi"] <- NULL
training_data[,"PRI_jet_leading_phi"] <- NULL
training_data[,"PRI_jet_all_pt"] <- NULL

# Save data for later purposes
training_data_saved <- training_data
```

1.4 Modelos de clasificación

caret incorpora métodos que permiten realizar validación cruzada mediante el controlador *trainControl*. Para ello se realizará primero una partición de los datos considerando una partición del 80% de los datos como entrenamiento y un 20% restante como validación.

```
set.seed(0)
trainIndex <- createDataPartition(training_data$Label, p = .8, list = FALSE)
training_set <- training_data[trainIndex, ]
validation_set <- training_data[-trainIndex, ]

trClassCtrl <- trainControl(classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             method = "cv", number = 10,
                             verboseIter=FALSE)
```

Para la realización de la práctica se ha decidido tomar diferentes modelos de clasificación y ver como estos son capaces de adaptarse a las condiciones del problema. Los modelos escogidos son:

- Árboles de decisión, con la función rpart. Se ha escogido este modelo ya que es uno de los modelos principales y más sencillos para ver como se comporta ante un problema tan complejo, además se realizará una variante en la que se explora todo el árbol.
- Random Forest, con la función ranger. Este modelo se ha escogido ya que utiliza un promedio de árboles de decisión y entrena un bosque, por lo que resulta interesante ver como se toman las diferentes decisiones y se median.
- Red Neuronal Artificial, con nnet. Como uno de los modelos más representativos de la Inteligencia Artificial, y además considerando la potencia de las redes neuronales, se ha escogido este método con una variación sobre el tamaño y el valor *decay* para comprobar también los diferentes comportamientos que puede tener una red neuronal y como se pueden adaptar al problema. Este tipo de modelo también resulta interesante debido a su adaptabilidad al problema.
- Naive Bayes, con naiveBayes. Con el fin de obtener un modelo probabilístico, se ha escogido este modelo, ya que se trata de uno de los más sencillos y a la vez más conocidos, por lo que poseer un enfoque diferente puede resultar interesante para la realización de este trabajo.
- K-Nearest Neighbour, con knn. Uno de los modelos más sencillos, aunque no se esperan a priori grandes resultados, es interesante ver como se distribuyen de forma espacial los datos, por lo que existe un importante interés en dicho modelo.

- Generalized Linear Model (glm). Finalmente se escoge también este modelo ya que utiliza conceptos como componentes de aleatoriedad, componentes sistemáticas que especifica las variables explicativas o predictoras y una función que define el valor esperado como combinación de las variables predictoras.

Tras observar los resultados obtenidos, se analizaran los resultados obtenidos, se obtendrán una serie de conclusiones sobre los modelos, y se obtendrá conocimiento suficiente para poder analizar modelos de predicción de bosón de Higgs.

Todos estos modelos a su vez serán comprobados mediante matrices de confusión con la función confusionMatrix de la librería caret y visualizandolas gráficamente, y con curvas ROC del paquete pROC.

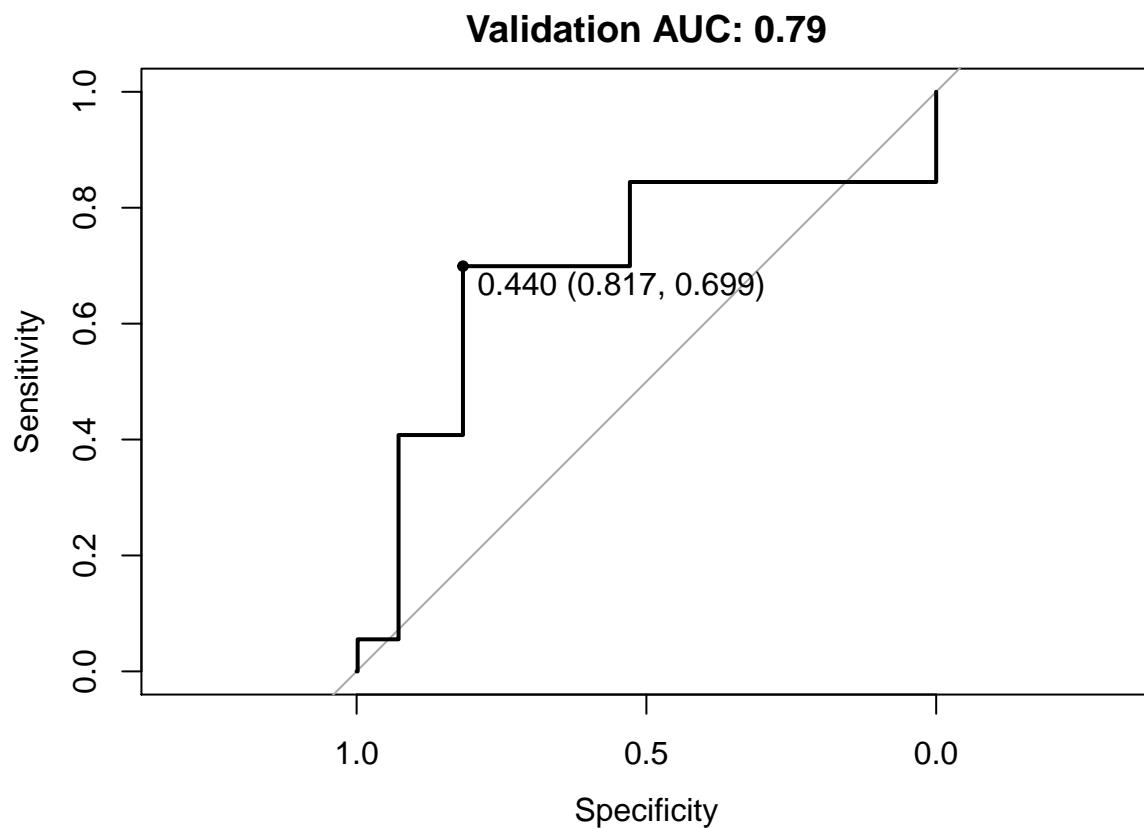
1.4.1 Árboles de decisión

El primer modelo de clasificación escogido es el árbol de decisión binario, el cual dado un conjunto de datos, divide este por características de forma que se tome una decisión o su opuesta, para clasificar. Se ha definido una función que solicita un conjunto de entrenamiento y realiza la clasificación en un árbol de decisión binario. Para ello se ha utilizado la librería rpart, concretamente haciendo uso tanto de la función rpart como de su representación mediante rpart.plot, perteneciente a la librería rpart.plot.

```
# Define model
modelo_rpart <- train(Label ~ .,
                        data = training_set,
                        method = "rpart",
                        metric = "ROC",
                        trControl = trClassCtrl)

# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_rpart, validation_set, type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_rpart, validation_set)

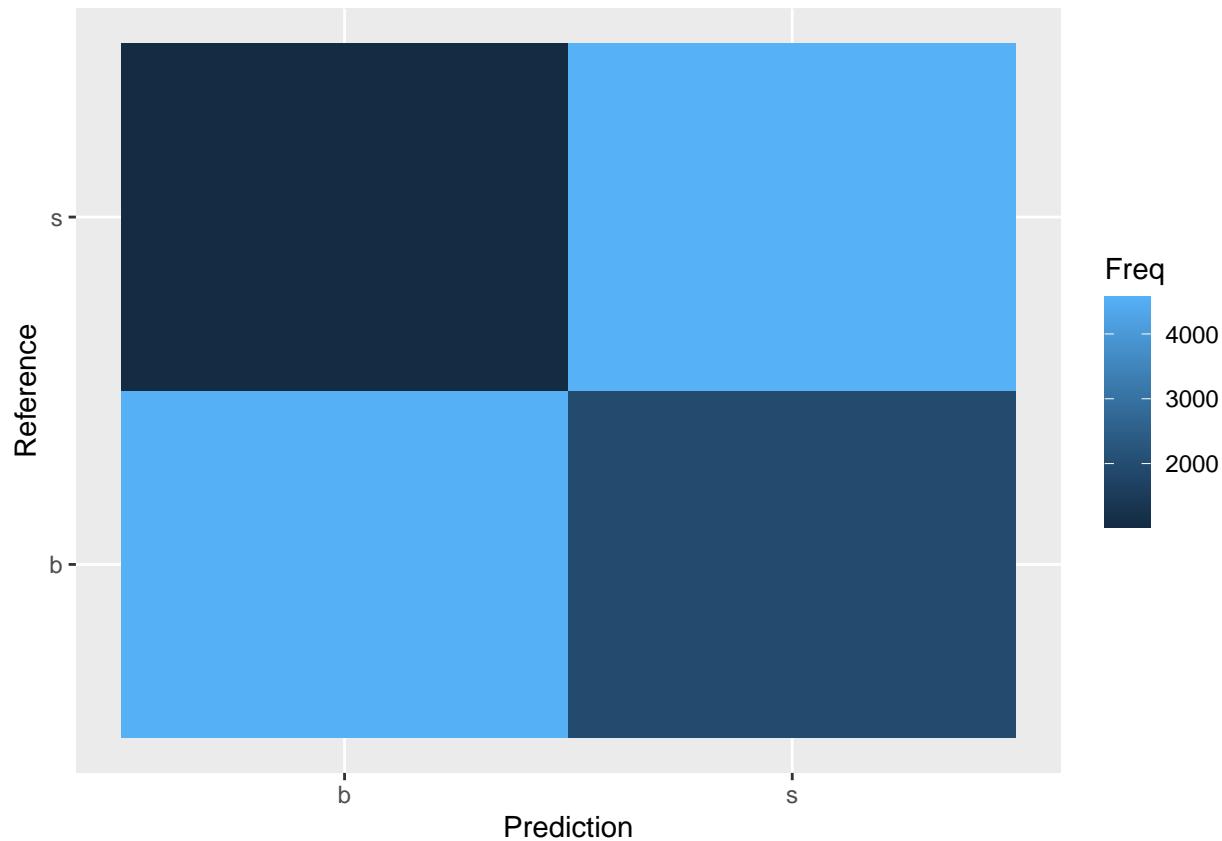
# Get AUC by ROC curve
auc_rpart <- roc(validation_set$Label, predictionValidationProb[["S"]],
                   levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_rpart, ylim=c(0,1), type = "S" , print.thres = T,
                           main=paste('Validation AUC:',
                                      round(auc_rpart$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_rpart <- confusionMatrix(predictionValidation, validation_set$Label)
acc_rpart$overall["Accuracy"]
```

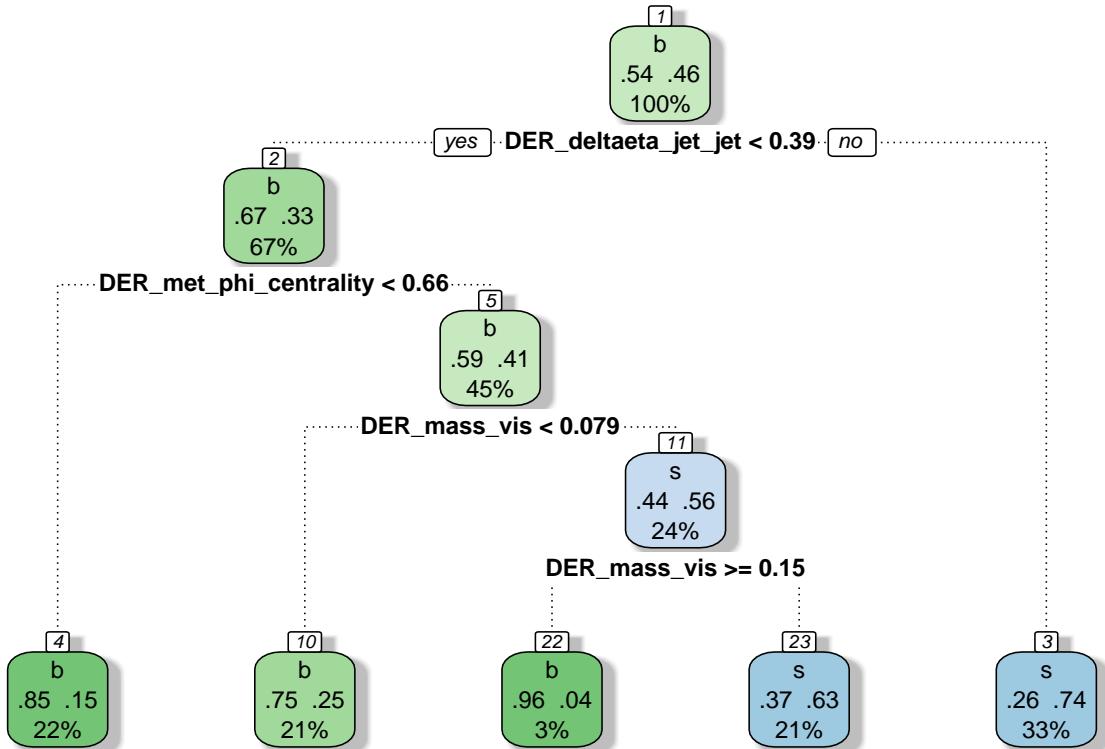
```
## Accuracy
## 0.7533862
```

```
# Plot confusion matrix
matrix_table <- data.frame(acc_rpart$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



Y dibujamos el árbol:

```
fancyRpartPlot(modelo_rpart$finalModel)
```



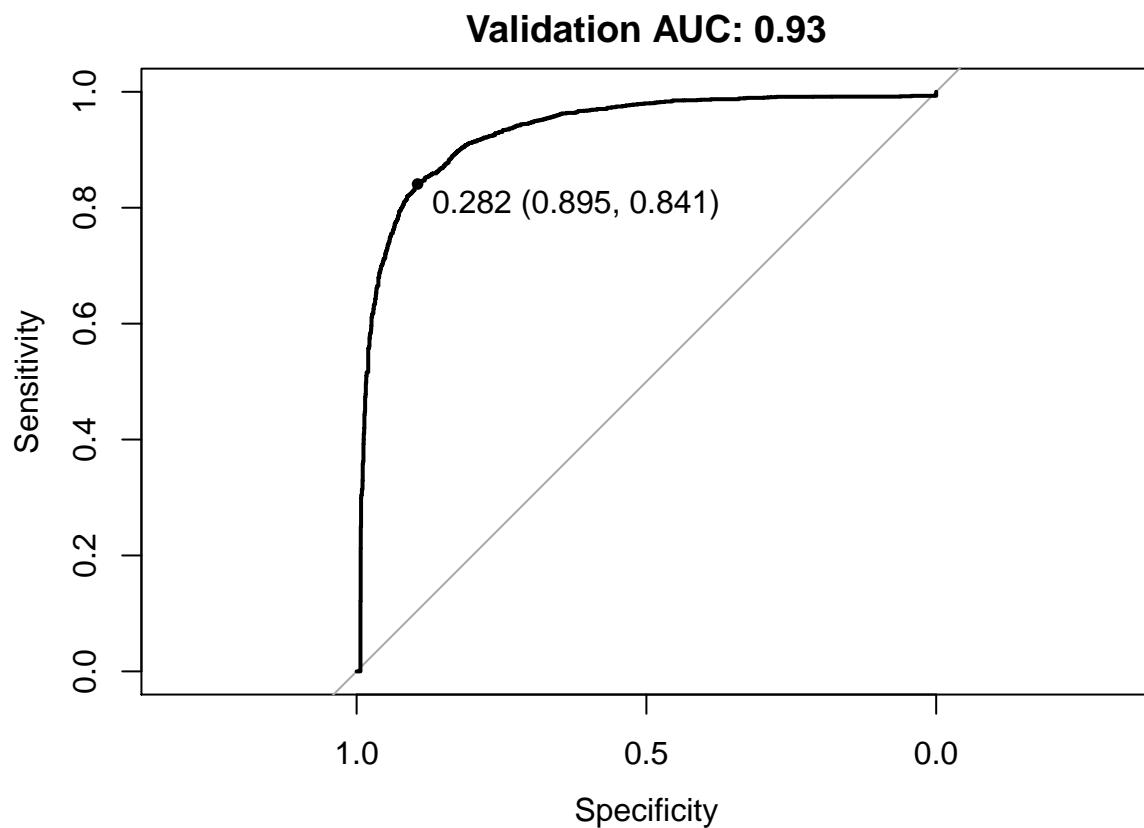
Rattle 2021–mar.–27 09:22:05 power

Además, se indica el parámetro `cp=-1` para que se explore el árbol entero, que aunque no se represente entero en el árbol dibujado, el modelo tendrá un mayor ajuste.

```
# Parameters Tune Grid
rpartParametersGrid <- expand.grid(.cp = -1)
# Define model
modelo_rpart_mejorado <- train(Label ~ .,
                                     data = training_set,
                                     method = "rpart",
                                     metric = "ROC",
                                     trControl = trClassCtrl,
                                     tuneGrid = rpartParametersGrid)

# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_rpart_mejorado, validation_set,
                                      type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_rpart_mejorado, validation_set)

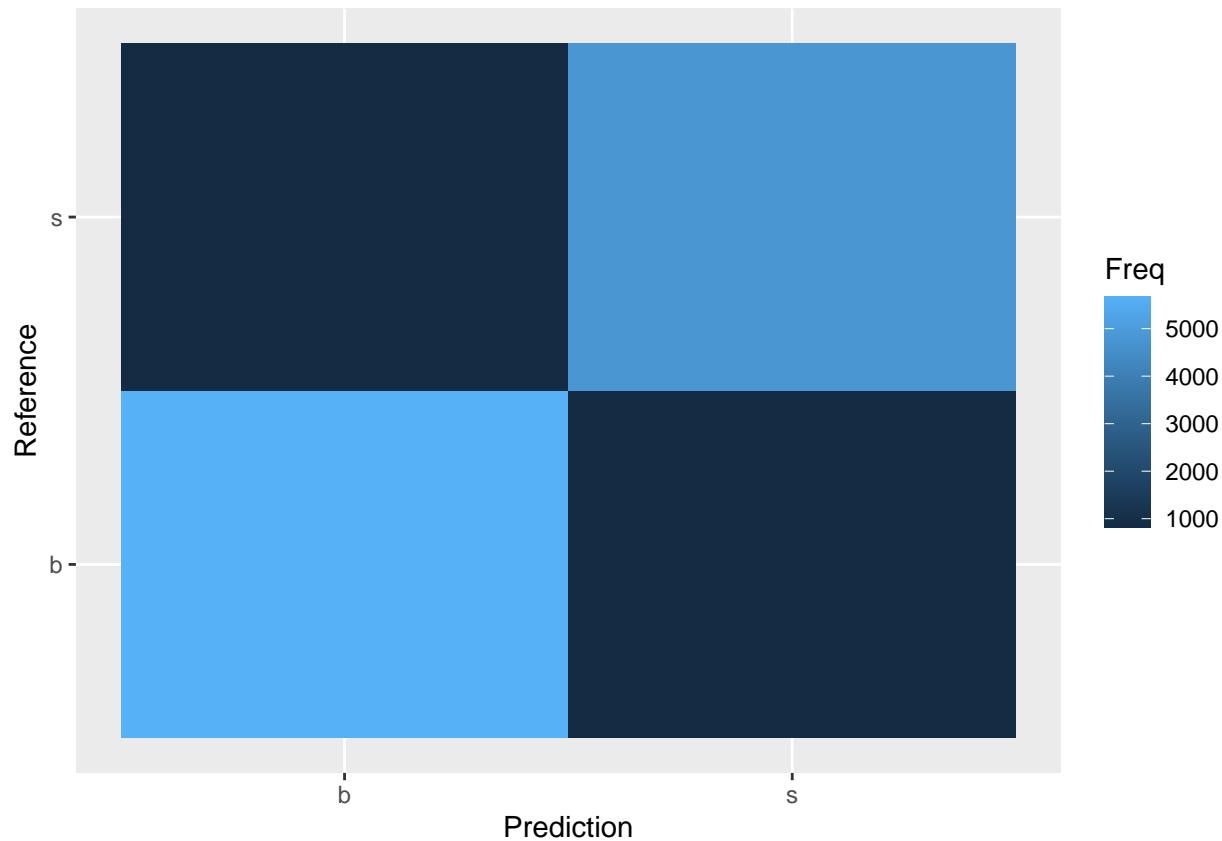
# Get AUC by ROC curve
auc_rpart_mejorado <- roc(validation_set$Label, predictionValidationProb[["s"]],
                            levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_rpart_mejorado, ylim=c(0,1), type = "S",
                           print.thres = T,
                           main=paste('Validation AUC: ',
                                      round(auc_rpart_mejorado$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_rpart_mejorado <- confusionMatrix(predictionValidation,
                                         validation_set$Label)
acc_rpart_mejorado$overall["Accuracy"]

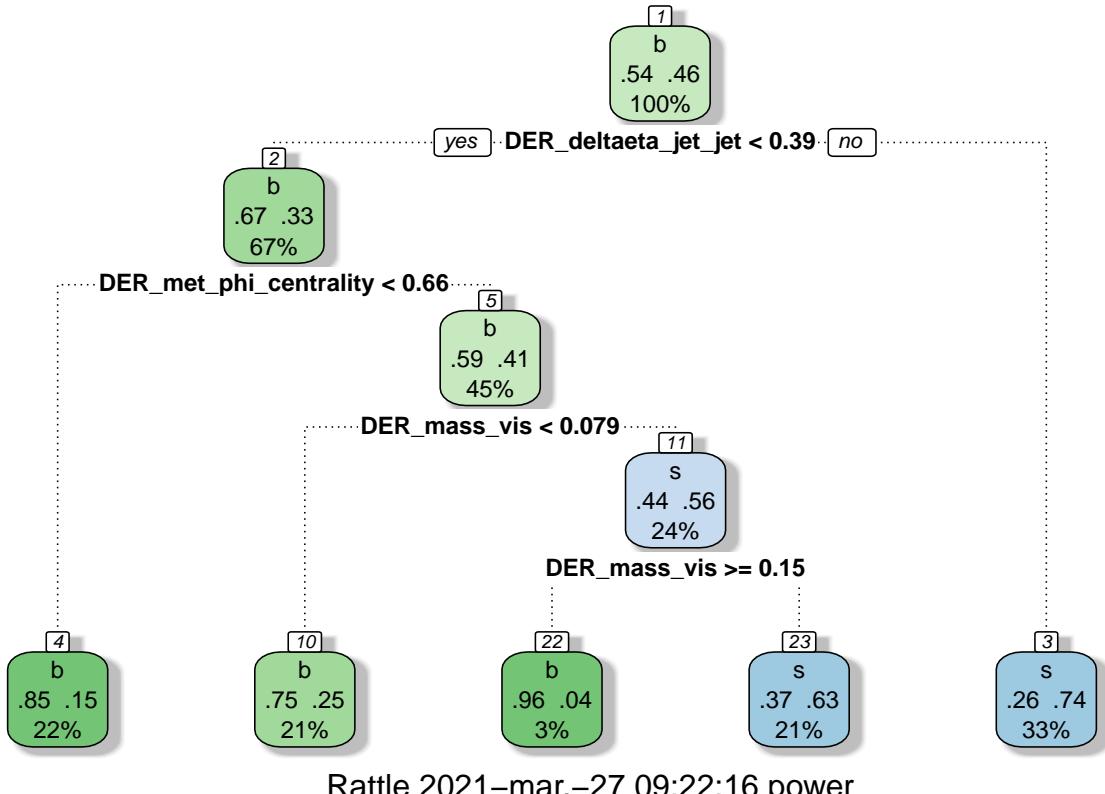
## Accuracy
## 0.8614139

# Plot confusion matrix
matrix_table <- data.frame(acc_rpart_mejorado$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



Y dibujamos el árbol:

```
fancyRpartPlot(modelo_rpart$finalModel)
```



1.4.2 Random Forest

El siguiente modelo a observar se trata de Random Forest, el cual dado un conjunto de datos, divide este por características de forma que se tome una decisión o su opuesta, para clasificar, formando varios árboles de decisión con decisiones diferentes. Tras crear una serie de árboles de decisión, el algoritmo se encarga de realizar una votación entre los distintos árboles para obtener un modelo final. Para ello se ha utilizado la librería ranger, utilizando para ello la función ranger ya que es más eficiente que el método *randomForest*.

Para la realización del modelo se escoge *sqrt(ncol(training_set))* como número de variables seleccionadas aleatoriamente para la división (*mtry*), como regla de partición *gini* ya que es la que existe por defecto para problemas de clasificación y un tamaño mínimo de nodos de 10 y 20 elementos tras estudiar y probar diferentes configuraciones del modelo.

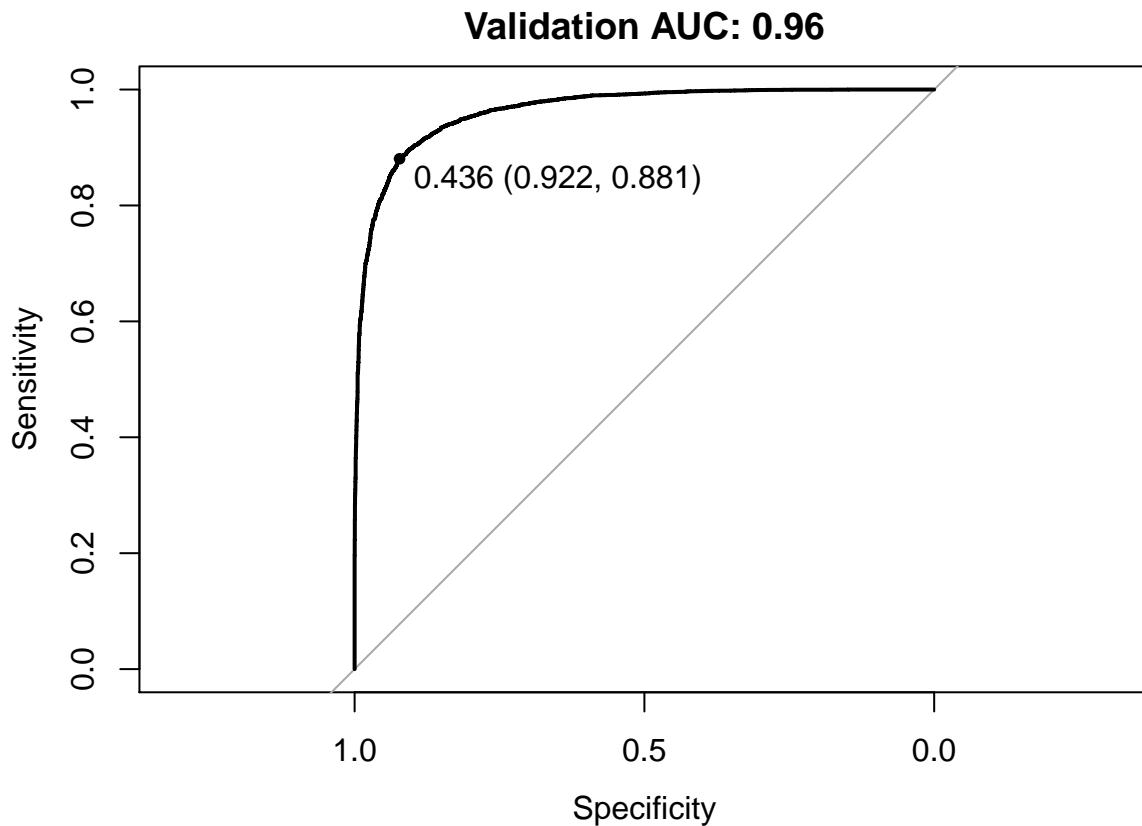
```
# Define tune grid
rfParametersGrid <- expand.grid(
  .mtry = sqrt(ncol(training_set)),
  .splitrule = "gini",
  .min.node.size = c(10, 20)
)
# Define model
modelo_rf <- train(Label ~ .,
  data = training_set,
  method = "ranger",
  metric = "ROC",
  trControl = trClassCtrl,
  tuneGrid = rfParametersGrid)
```

```

# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_rf, validation_set, type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_rf, validation_set)

# Get AUC by ROC curve
auc_rf <- roc(validation_set$Label, predictionValidationProb[["s"]],
               levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_rf, ylim=c(0,1), type = "S" , print.thres = T,
                           main=paste('Validation AUC: ',
                                      round(auc_rf$auc[[1]], 2)))

```



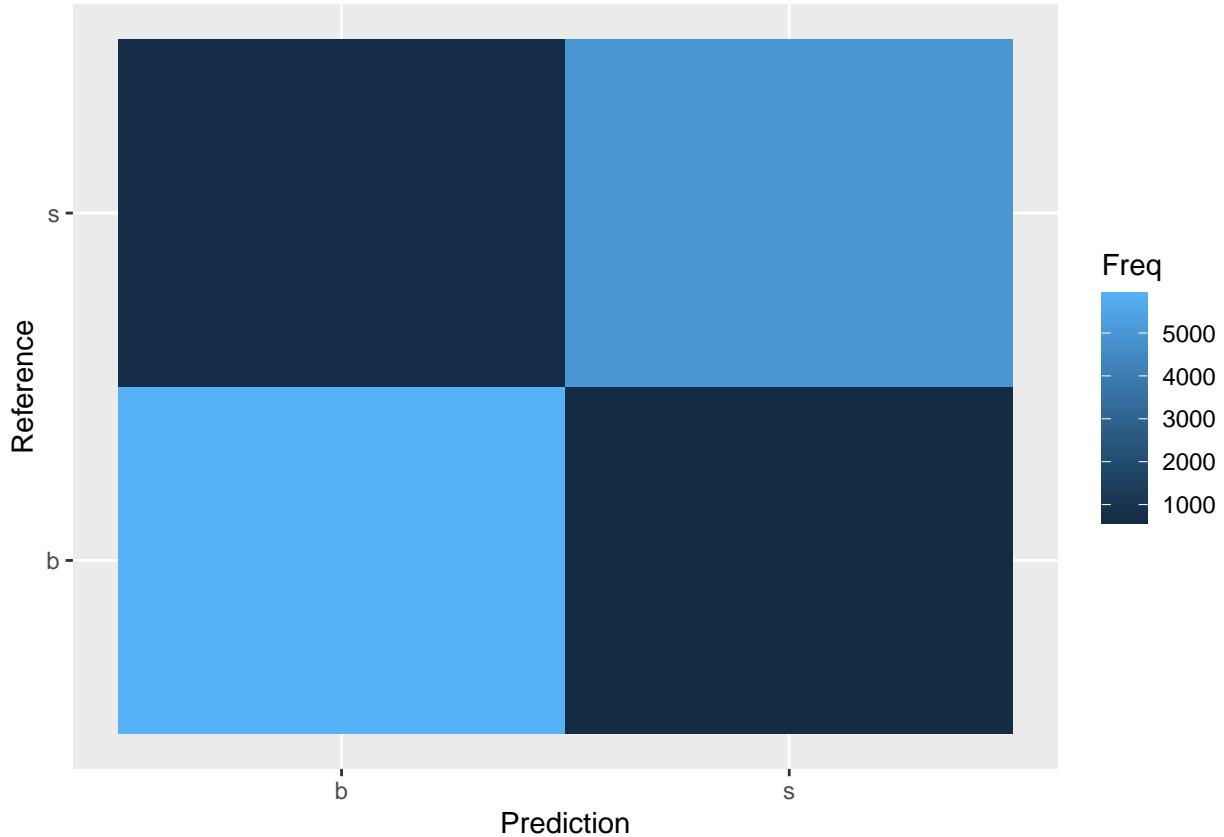
```

# Get accuracy by confusion matrix
acc_rf <- confusionMatrix(predictionValidation, validation_set$Label)
acc_rf$overall["Accuracy"]

## Accuracy
## 0.8987446

# Plot confusion matrix
matrix_table <- data.frame(acc_rf$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()

```



1.4.3 Redes Neuronales

El tercer modelo seleccionado es el de Red Neuronal Artificial, el cual recibe en las neuronas de entrada un conjunto de datos, que son procesados por las diferentes capas de neuronas ocultas (previamente configuradas) de la red, para obtener en la salida un modelo que a través del aprendizaje puede ajustarse mediante pesos al conjunto de datos.

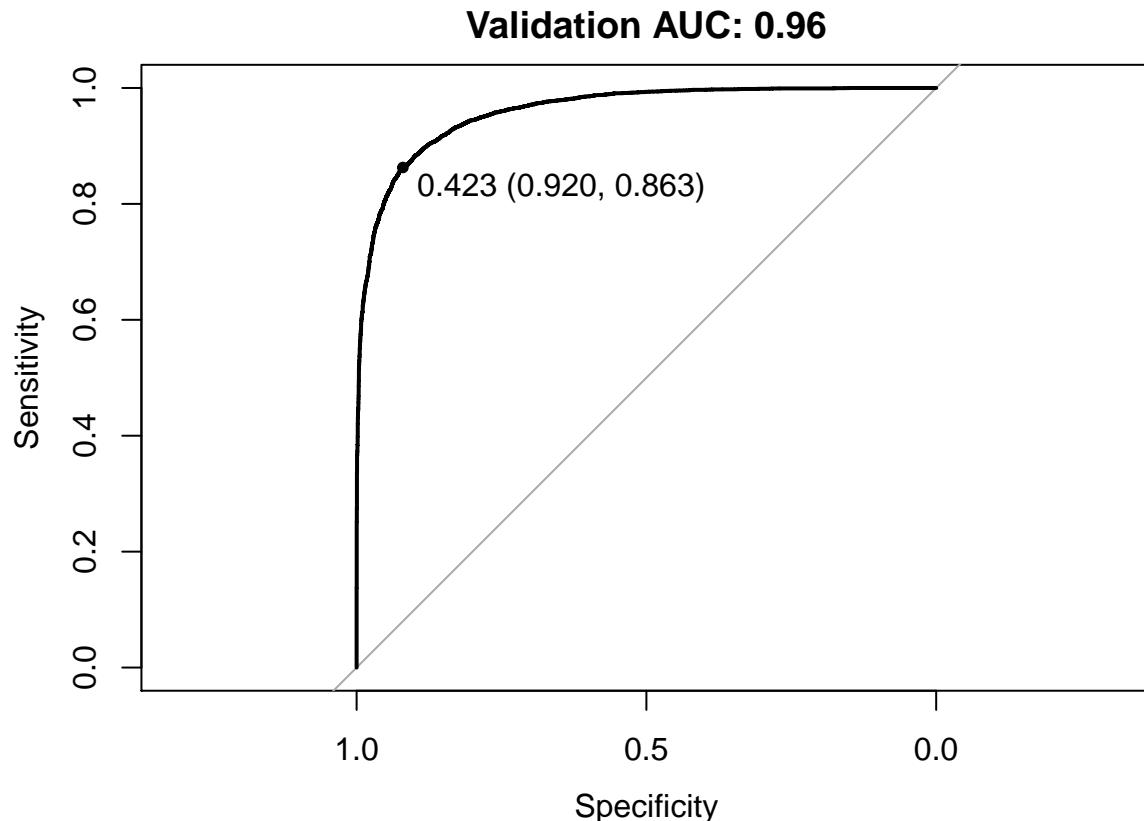
Para ello se ha utilizado la librería nnet, empleando la función nnet.

Para la realización del modelo se han utilizado una serie de configuraciones distintas, primero la configuración por defecto:

```
# Define model
# Use capture.output to silent output of iterations
silent <- capture.output(modelo_rna <- train(Label ~ .,
                                                data = training_set,
                                                method = "nnet",
                                                metric = "ROC",
                                                trControl = trClassCtrl))

# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_rna, validation_set, type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_rna, validation_set)
```

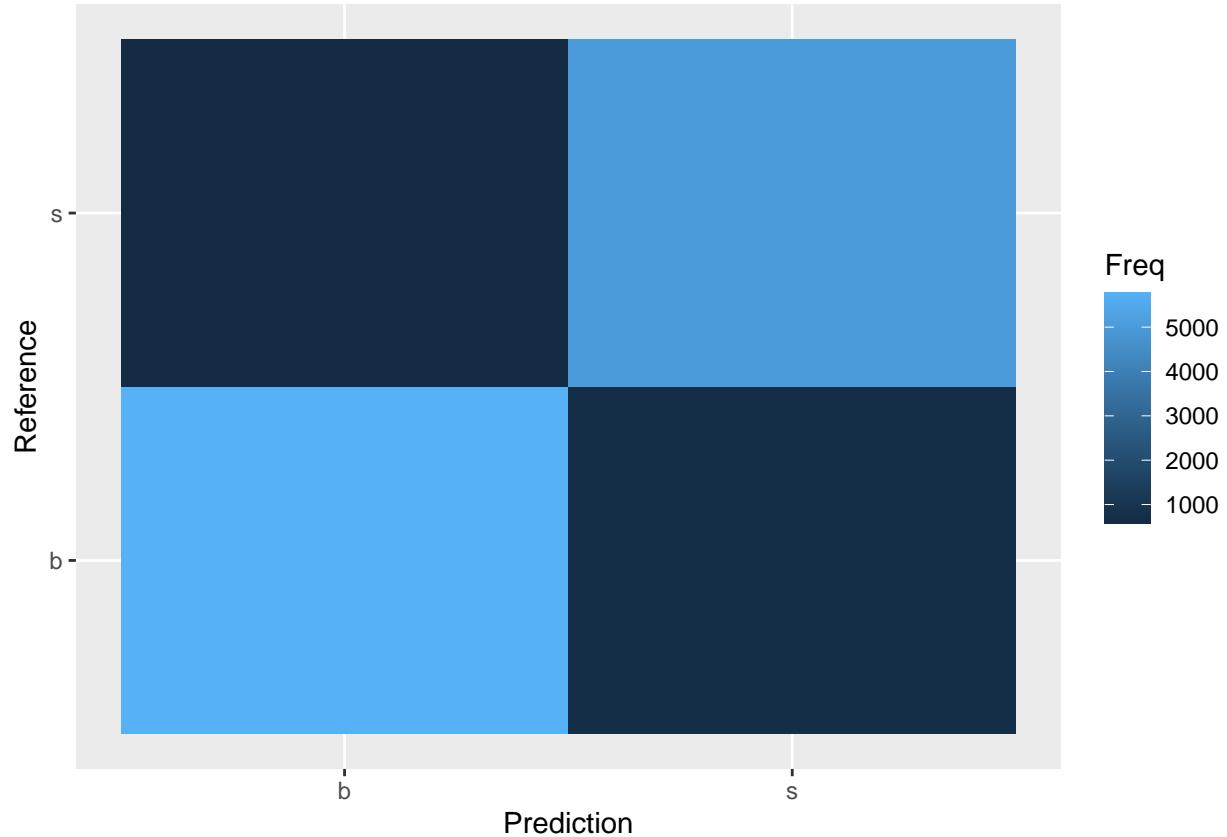
```
# Get AUC by ROC curve
auc_rna <- roc(validation_set$Label, predictionValidationProb[["S"]],
  levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_rna, ylim=c(0,1), type = "S" , print.thres = T,
  main=paste('Validation AUC: ',
  round(auc_rna$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_rna <- confusionMatrix(predictionValidation, validation_set$Label)
acc_rna$overall["Accuracy"]
```

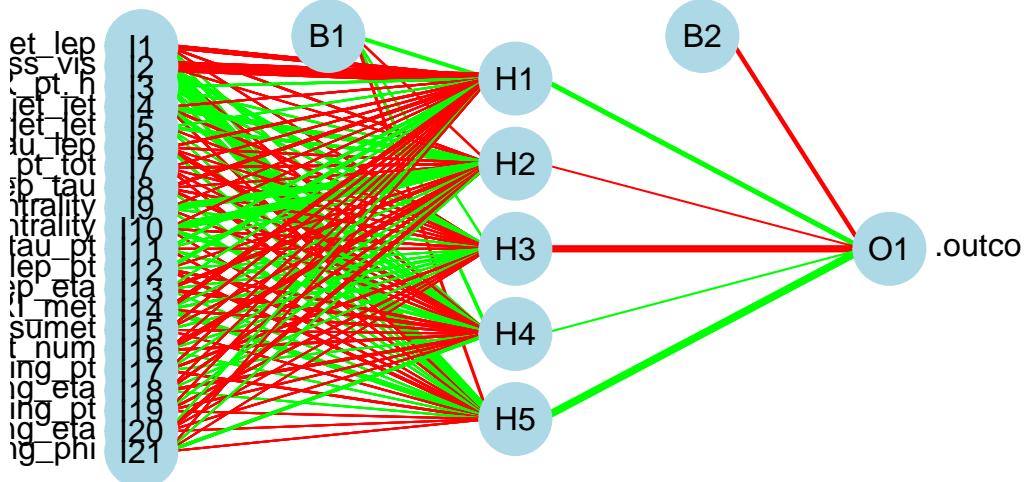
```
## Accuracy
## 0.8899075
```

```
# Plot confusion matrix
matrix_table <- data.frame(acc_rna$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



Y dibujamos la red neuronal:

```
plotnet(modelo_rna$finalModel, pos_col="green", neg_col="red", max_sp=TRUE)
```

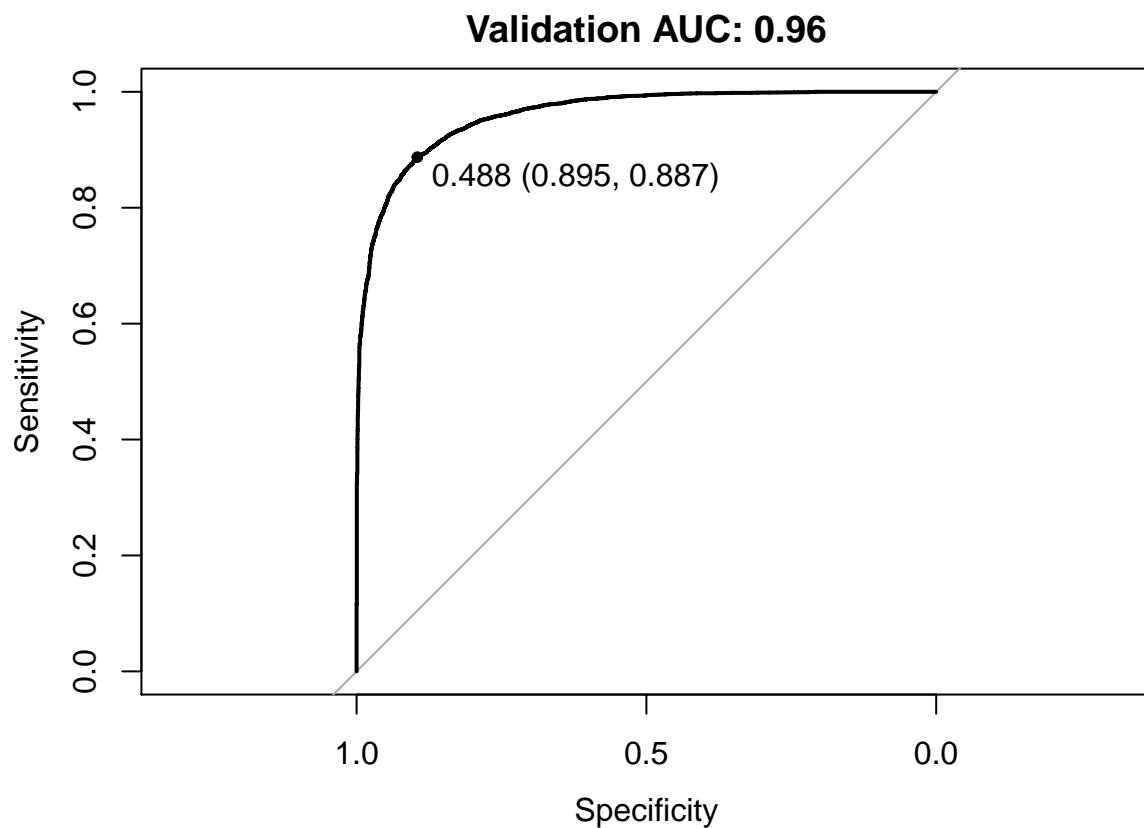


A continuación se realiza una configuración con variaciones de los parámetros *size* que representa el número de unidades en las capas ocultas y el parámetro *decay* que es el parámetro de regularización para evitar el *overfitting* o sobreajuste. Finalmente se opta por un modelo con *size* 5 y *decay* de 0.3.

```
# Define tune grid
nnetGrid <- expand.grid(size = 5, decay = 0.3)
# Train model
silent <- capture.output(modelo_rna_mejorado <- train(Label ~ .,
                                                       data = training_set,
                                                       method = "nnet",
                                                       metric = "ROC",
                                                       trControl = trClassCtrl,
                                                       tuneGrid = nnetGrid))

# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_rna_mejorado, validation_set,
                                      type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_rna_mejorado, validation_set)

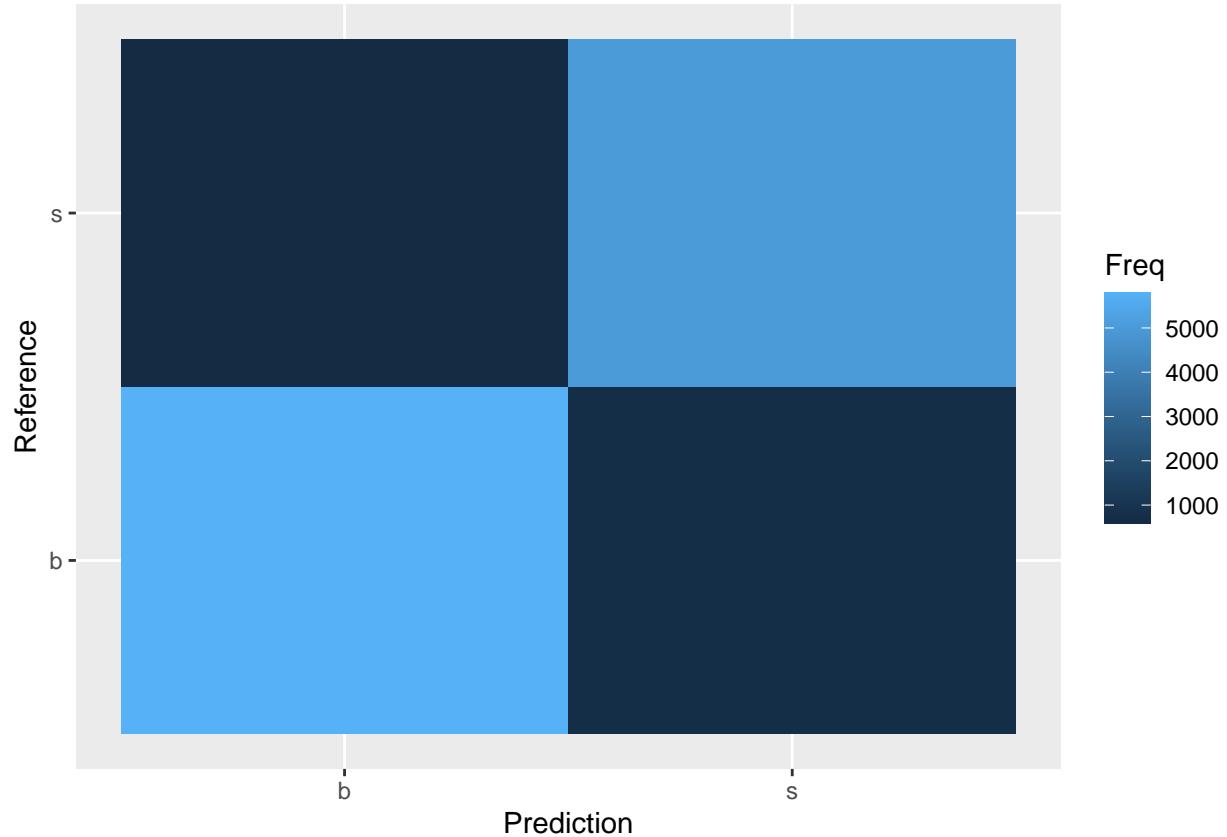
# Get AUC by ROC curve
auc_rna_mejorado <- roc(validation_set$Label, predictionValidationProb[["s"]],
                           levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_rna_mejorado, ylim=c(0,1), type = "S" ,
                           print.thres = T,
                           main=paste('Validation AUC: ',
                           round(auc_rna_mejorado$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_rna_mejorado <- confusionMatrix(predictionValidation, validation_set$Label)
acc_rna_mejorado$overall["Accuracy"]
```

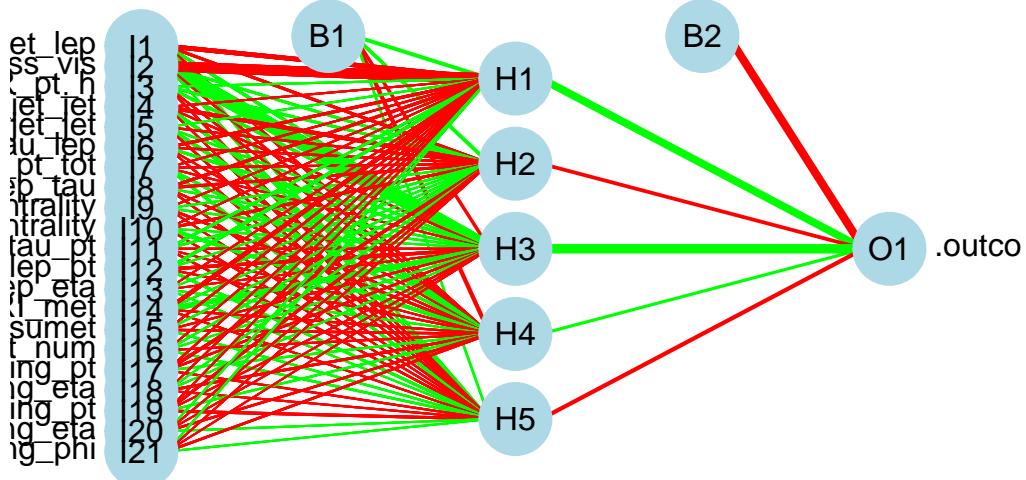
```
## Accuracy
## 0.8905682
```

```
# Plot confusion matrix
matrix_table <- data.frame(acc_rna_mejorado$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



Y dibujamos la red neuronal:

```
plotnet(modelo_rna_mejorado$finalModel, pos_col="green", neg_col="red",
        max_sp=TRUE)
```

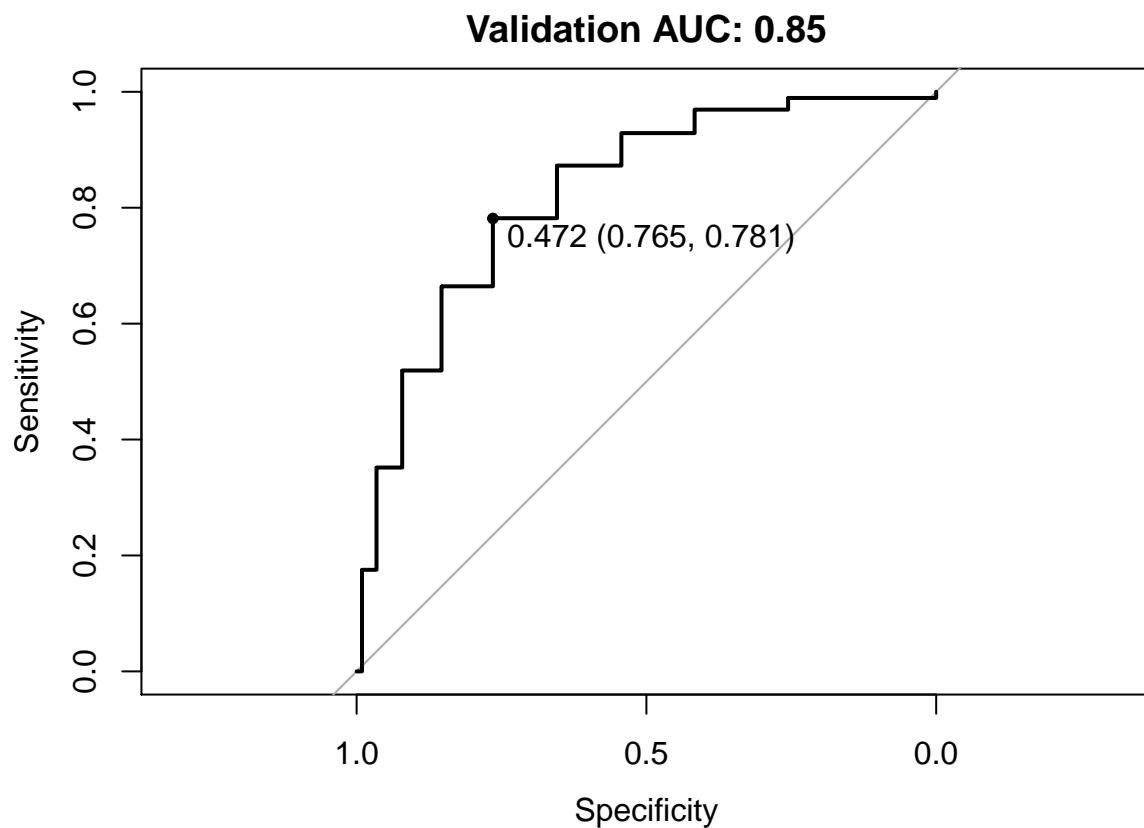


1.4.4 KNN

El cuarto modelo seleccionado es el de K-Nearest Neighbors, el cual dado un conjunto de entrada con diferentes clases, analiza si un elemento posee dentro de su rango más vecinos de una clase u otra, y clasifica en aquella clase en la que posea más vecinos dentro de un rango con k vecinos. Este modelo sigue una distribución espacial de los elementos.

```
# Define model
modelo_knn <- train(Label ~ ., data = training_set, method = "knn",
                      metric = "ROC", trControl = trClassCtrl)
# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_knn, validation_set, type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_knn, validation_set)

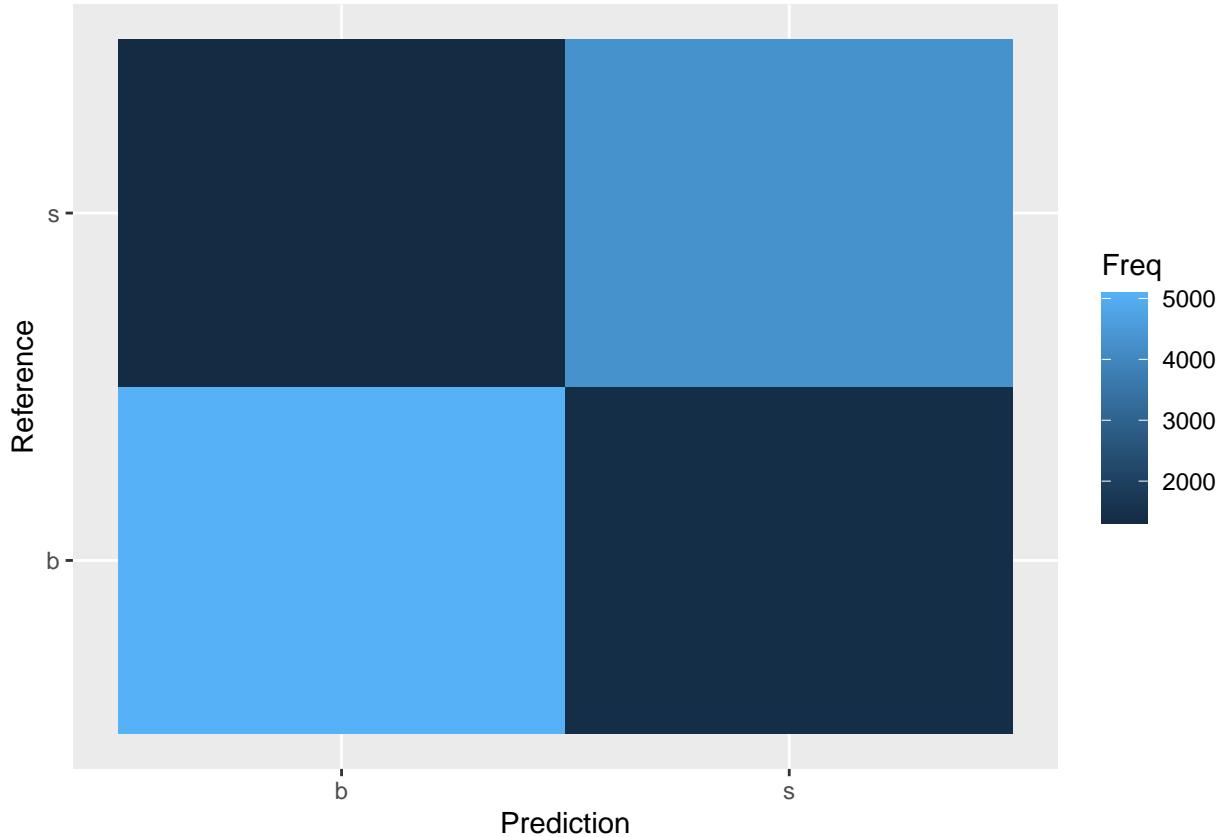
# Get AUC by ROC curve
auc_knn <- roc(validation_set$Label, predictionValidationProb[["s"]],
                 levels = unique(validation_set[[ "Label"]]))
roc_validation <- plot.roc(auc_knn, ylim=c(0,1), type = "S" , print.thres = T,
                           main=paste('Validation AUC: ',
                           round(auc_knn$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_knn <- confusionMatrix(predictionValidation, validation_set$Label)
acc_knn$overall["Accuracy"]

## Accuracy
## 0.7737859

# Plot confusion matrix
matrix_table <- data.frame(acc_knn$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



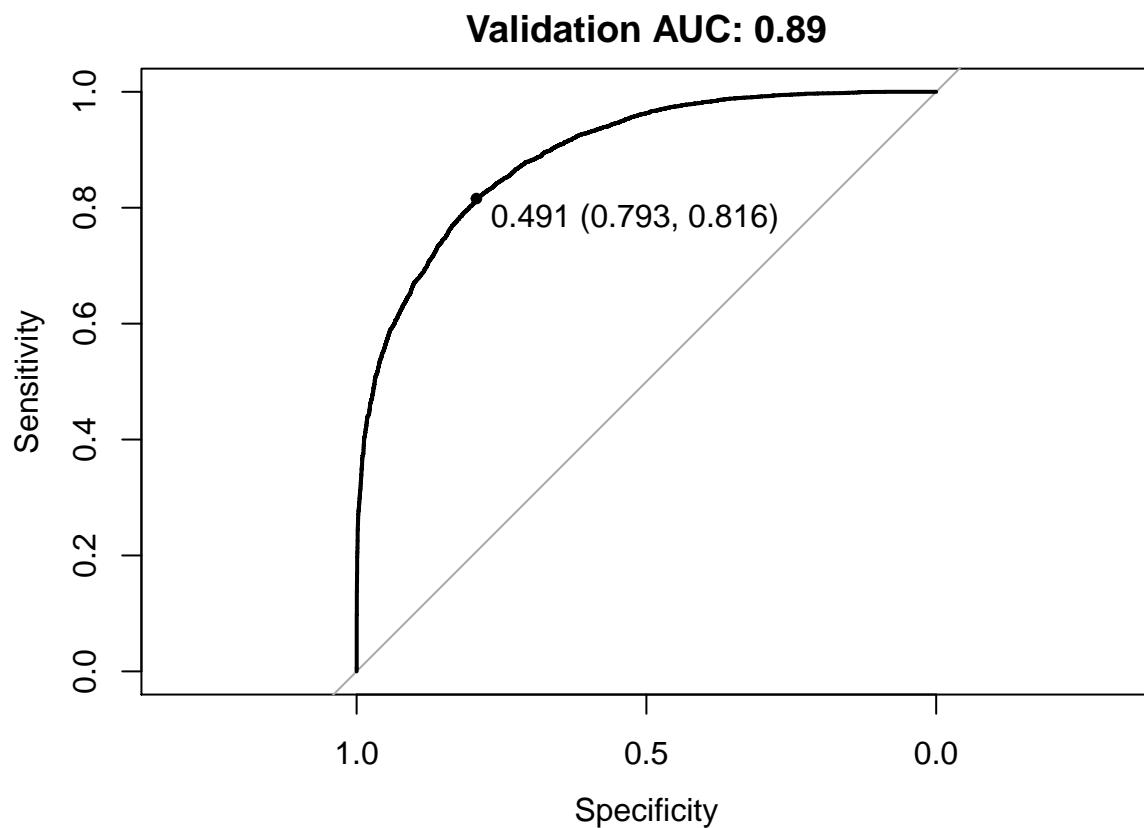
1.4.5 Naive Bayes

El siguiente modelo seleccionado es el de Naive Bayes, el cual asume que la presencia de una característica no es dependiente de la existencia de otra, permitiendo un entrenamiento del modelo mediante las frecuencias relativas de las características del conjunto de entrenamiento. Para ello crea un modelo que mezcla probabilidad y frecuencia, definiendo así las probabilidades de que se de una determinada clase a partir de sus características de forma independiente.

Para ello se ha utilizado la librería e1071, utilizando la función naiveBayes.

```
# Train model
modelo_nb <- train(Label ~ ., data = training_set, method = "naive_bayes",
                     metric = "ROC", trControl = trClassCtrl)
# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_nb, validation_set, type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_nb, validation_set)

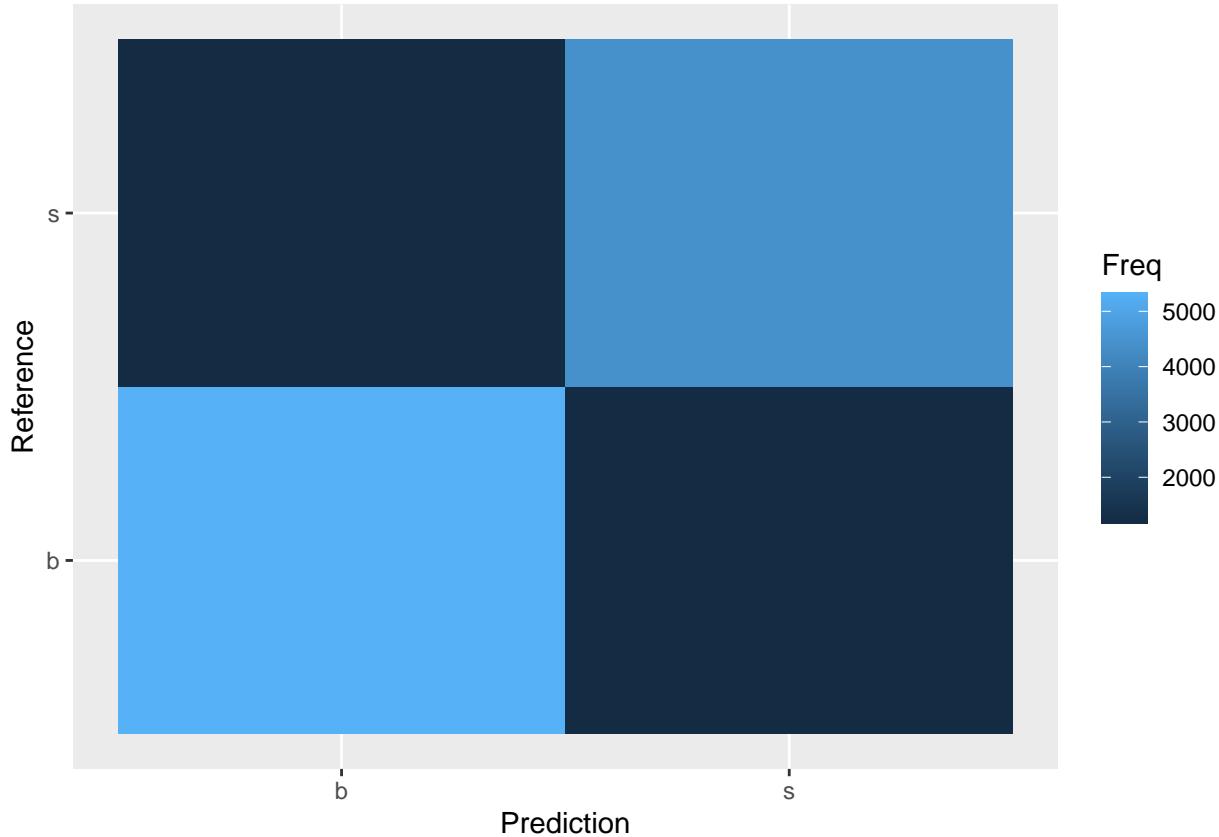
# Get AUC by ROC curve
auc_nb <- roc(validation_set$Label, predictionValidationProb[["s"]],
                levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_nb, ylim=c(0,1), type = "S" , print.thres = T,
                           main=paste('Validation AUC:',
                                      round(auc_nb$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_nb <- confusionMatrix(predictionValidation, validation_set$Label)
acc_nb$overall["Accuracy"]

## Accuracy
## 0.8047572

# Plot confusion matrix
matrix_table <- data.frame(acc_nb$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



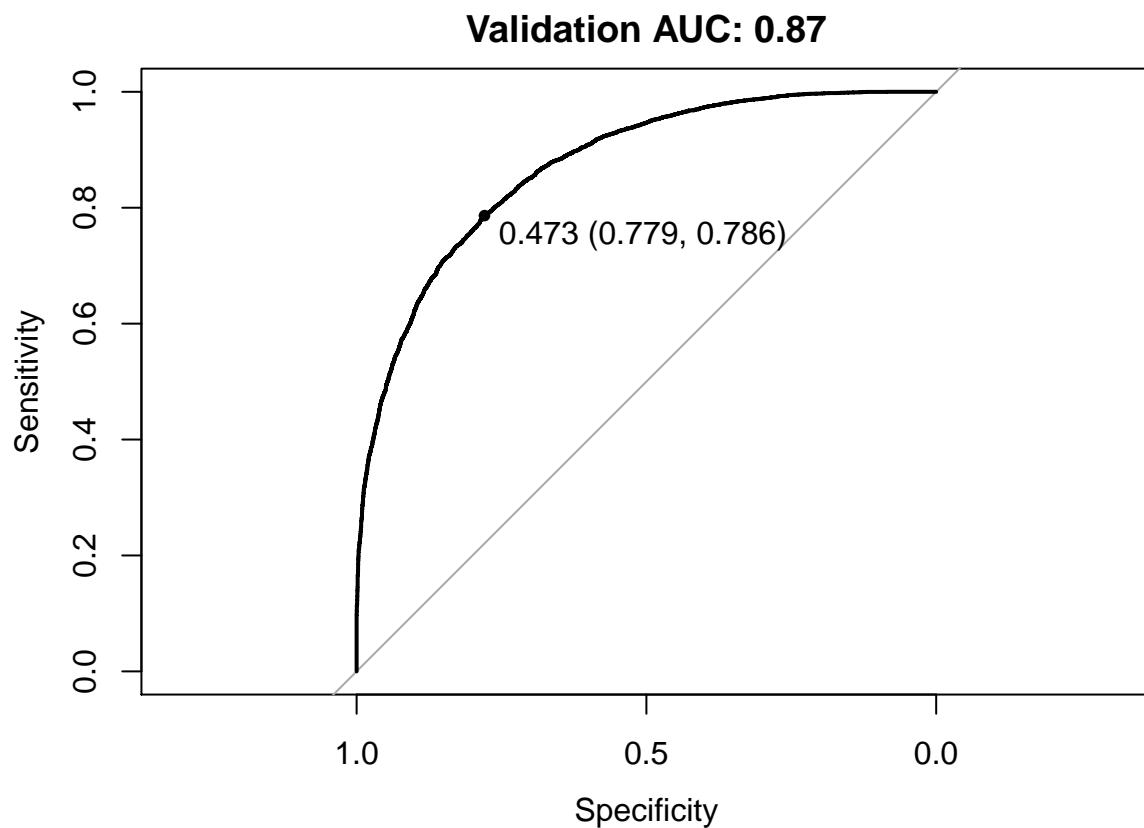
1.4.6 GLM

A continuación el siguiente modelo es Generalized Linear Model el cual es una generalización de la regresión lineal, que permite responder mediante de modelos de distribución permitiendo que la variable de respuesta se relacion con un modelo lineal y que la magnitud de la varianza sea una función predictora.

Para ello se ha utilizado la versión estándar del modelo *glm* utilizando los valores por defecto:

```
# Train model
modelo_glm <- train(Label ~ ., data = training_set, method = "glm",
                      metric = "ROC", trControl = trClassCtrl)
# Get prediction validation probabilities
predictionValidationProb <- predict(modelo_glm, validation_set, type = "prob")
# Get prediction validation
predictionValidation <- predict(modelo_glm, validation_set)

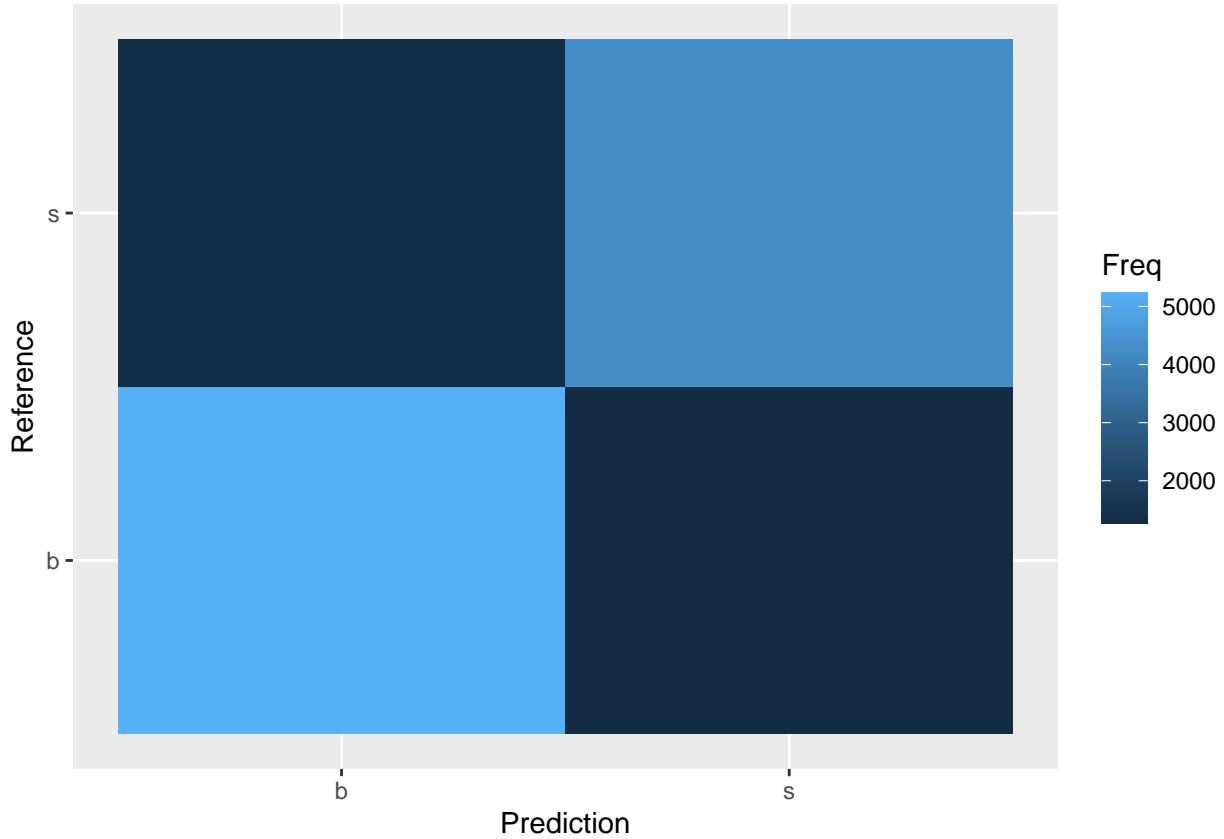
# Get AUC by ROC curve
auc_glm <- roc(validation_set$Label, predictionValidationProb[["s"]],
                 levels = unique(validation_set[["Label"]]))
roc_validation <- plot.roc(auc_glm, ylim=c(0,1), type = "S" , print.thres = T,
                           main=paste('Validation AUC: ',
                                      round(auc_glm$auc[[1]], 2)))
```



```
# Get accuracy by confusion matrix
acc_glm <- confusionMatrix(predictionValidation, validation_set$Label)
acc_glm$overall["Accuracy"]

## Accuracy
## 0.7822927

# Plot confusion matrix
matrix_table <- data.frame(acc_glm$table)
ggplot(matrix_table, aes(x=Prediction, y=Reference, fill=Freq)) + geom_tile()
```



Tras realizar los diferentes modelos, se procede a visualizar una tabla comparativa del *accuracy* obtenido por los diferentes modelos:

```

models_name <- c("Árbol de decisión", "Árbol de decisión mejorado",
                 "Random Forest", "Red Neuronal Artificial",
                 "Red Neuronal Artificial mejorado", "KNN",
                 "Naive Bayes", "GLM")
models_acc <- c(acc_rpart$overall[["Accuracy"]],
                  acc_rpart_mejorado$overall[["Accuracy"]],
                  acc_rf$overall[["Accuracy"]],
                  acc_rna$overall[["Accuracy"]],
                  acc_rna_mejorado$overall[["Accuracy"]],
                  acc_knn$overall[["Accuracy"]],
                  acc_nb$overall[["Accuracy"]],
                  acc_glm$overall[["Accuracy"]])
models_auc <- c(auc_rpart$auc[1],
                  auc_rpart_mejorado$auc[1],
                  auc_rf$auc[1],
                  auc_rna$auc[1],
                  auc_rna_mejorado$auc[1],
                  auc_knn$auc[1],
                  auc_nb$auc[1],
                  auc_glm$auc[1])
acc_table <- data.frame(models_name, models_acc, models_auc)
names(acc_table) <- c("Modelos", "Accuracy", "AUC")
kable(acc_table)

```

Modelos	Accuracy	AUC
Árbol de decisión	0.7533862	0.7875582
Árbol de decisión mejorado	0.8614139	0.9339792
Random Forest	0.8987446	0.9642460
Red Neuronal Artificial	0.8899075	0.9605879
Red Neuronal Artificial mejorado	0.8905682	0.9608424
KNN	0.7737859	0.8543283
Naive Bayes	0.8047572	0.8931037
GLM	0.7822927	0.8721486

2 Alternativas

Nota: En las diferentes alternativas se van a repetir la ejecución de los mismos modelos con los mismos parámetros, por lo que se encontrará el código prácticamente idéntico al visto previamente. Finalmente se compararán los resultados de las distintas alternativas.

2.1 Utilización del conocimiento primitivo

Este enfoque consiste en descartar todo el conocimiento derivado por los investigadores del conocimiento, suponiendo que este conocimiento pueda ser incorrecto o llevar a modelos erróneos debido a cualquier problema en el planteamiento del problema. Por lo tanto se trabajará únicamente con las variables primitivas, es decir, con los datos obtenidos tal cual del experimento.

Previamente se obtienen los datos que ya han recibido una normalización, detección de outliers y eliminación de ruido.

```
training_data <- training_data_raw
# Delete DERivated columns
training_data <- training_data[, -grep("DER", colnames(training_data))]
```

```
set.seed(0)
trainIndex <- createDataPartition(training_data$Label, p = .8, list = FALSE)
training_set <- training_data[trainIndex, ]
validation_set <- training_data[-trainIndex, ]

trClassCtrl <- trainControl(classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             method = "cv", number = 10,
                             verboseIter = FALSE)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.6538652	0.6504694
Árbol de decisión mejorado	0.7002808	0.7732742
Random Forest	0.7726297	0.8538873
Red Neuronal Artificial	0.7518996	0.8261803
Red Neuronal Artificial mejorado	0.7352990	0.8176307
KNN	0.7329039	0.8044313
Naive Bayes	0.7186158	0.7955711
GLM	0.6693095	0.7285056

2.2 Utilización del conocimiento derivado

Este enfoque consiste en descartar todo el conocimiento primitivo del problema, suponiendo que el conocimiento derivado por los investigadores aporta una información más precisa acerca del problema, y que el uso de datos primitivos puede llevar a conclusiones erróneas. Por lo tanto se trabajará únicamente con las variables derivadas, es decir, con los datos obtenidos tras el procesamiento de los investigadores.

Previamente se obtienen los datos que ya han recibido una normalización, detección de outliers y eliminación de ruido.

```
training_data <- training_data_raw
# Delete PRImitive columns
training_data <- training_data[, -grep("PRI", colnames(training_data))]

set.seed(0)
trainIndex <- createDataPartition(training_data$Label, p = .8, list = FALSE)
training_set <- training_data[trainIndex, ]
validation_set <- training_data[-trainIndex, ]

trClassCtrl <- trainControl(classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             method = "cv", number = 10,
                             verboseIter=FALSE)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.8173109	0.8257992
Árbol de decisión mejorado	0.8890816	0.9537824
Random Forest	0.9159234	0.9740465
Red Neuronal Artificial	0.9036174	0.9673459
Red Neuronal Artificial mejorado	0.9005616	0.9648644
KNN	0.8487777	0.9240923
Naive Bayes	0.8554675	0.9374972
GLM	0.7562768	0.8466590

2.3 Selección aleatoria de instancias

Esta alternativa consiste en obtener un subconjunto de instancias aleatorias para realizar los diferentes modelos, para ello se ha de obtener un conjunto de datos lo suficientemente grande para evitar un sobreajuste sobre un modelo pequeño de los datos. Por lo tanto tras realizar puebas se ha optado pasar del conjunto de 66.246 datos que se tenían en un principio en el modelo inicial a un total de 10.000 datos escogidos aleatoriamente.

Para realizar la selección de subconjuntos de datos se va a emplear la función sample_n del paquete tidyverse.

```
training_data <- training_data_raw
# Random subset
training_data <- sample_n(training_data, size=1000)

set.seed(0)
trainIndex <- createDataPartition(training_data$Label, p = .8, list = FALSE)
training_set <- training_data[trainIndex, ]
```

```

validation_set <- training_data[-trainIndex, ]

trClassCtrl <- trainControl(classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             method = "cv", number = 10,
                             verboseIter=FALSE)

```

Modelos	Accuracy	AUC
Árbol de decisión	0.8592965	0.8706422
Árbol de decisión mejorado	0.8341709	0.8914883
Random Forest	0.9095477	0.9598369
Red Neuronal Artificial	0.8492462	0.9436290
Red Neuronal Artificial mejorado	0.7989950	0.9084608
KNN	0.7587940	0.8164118
Naive Bayes	0.8592965	0.9222222
GLM	0.7788945	0.8612640

2.3.1 Realizar un balanceo de datos

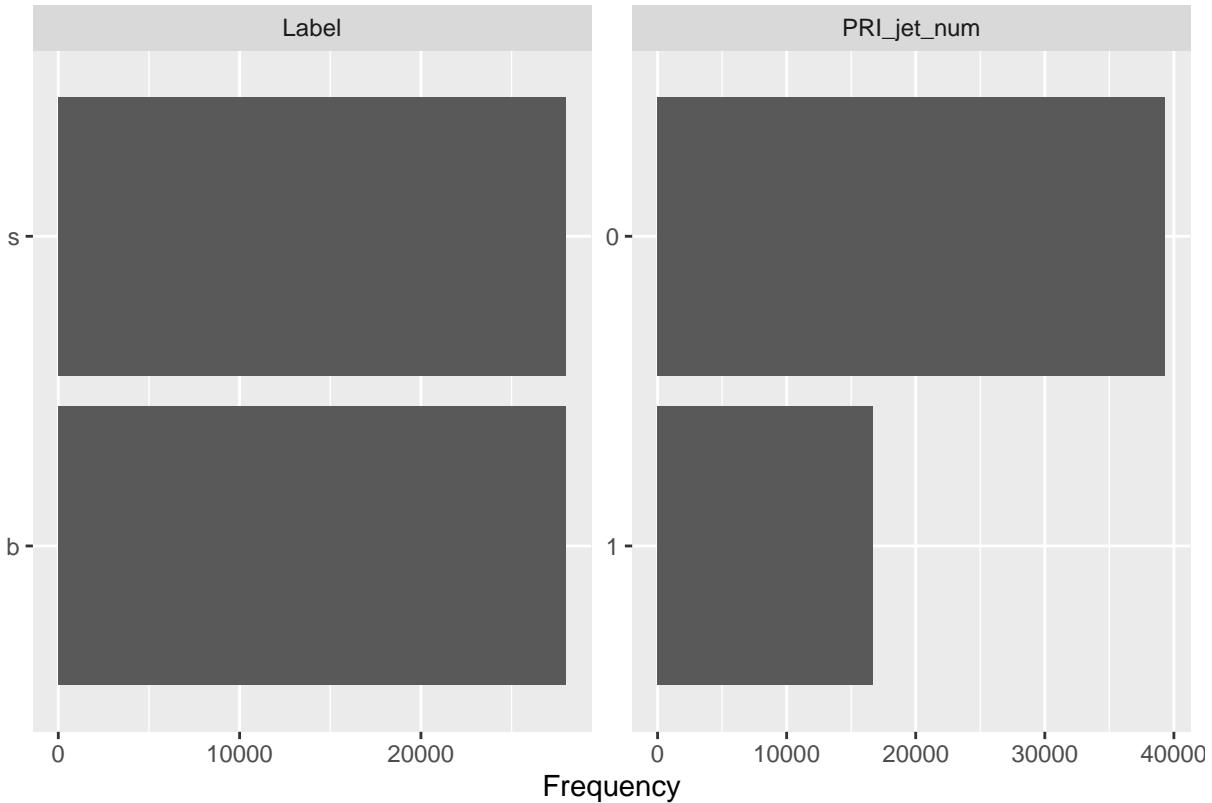
Esta alternativa consiste en realizar un submuestreo aleatorio de la clase mayoritaria (ruido) hasta obtener un conjunto de datos balanceado con el mismo número de eventos para cada clase. Para ello se utilizará el conjunto de datos del procedimiento inicial (con el conjunto de características reducidas).

Para realizar este *down-sampling* se utilizará la función *downSample* de *caret*.

```

training_data <- training_data_saved
training_data <- downSample(training_data, training_data$Label)
# Delete class column generated
training_data[,"Class"] <- NULL
plot_bar(training_data)

```



```

set.seed(0)
trainIndex <- createDataPartition(training_data$Label, p = .8, list = FALSE)
training_set <- training_data[trainIndex, ]
validation_set <- training_data[-trainIndex, ]

trClassCtrl <- trainControl(classProbs = TRUE,
                             summaryFunction = twoClassSummary,
                             method = "cv", number = 10,
                             verboseIter=FALSE)

```

Modelos	Accuracy	AUC
Árbol de decisión	0.7542433	0.7822060
Árbol de decisión mejorado	0.8604610	0.9320436
Random Forest	0.8983384	0.9633150
Red Neuronal Artificial	0.8902090	0.9604001
Red Neuronal Artificial mejorado	0.8880650	0.9592596
KNN	0.7672860	0.8503103
Naive Bayes	0.7943541	0.8856899
GLM	0.7788101	0.8682564

3 Comparativas de diferentes planteamientos

3.1 Planteamiento inicial:

```
kable(acc_table)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.7533862	0.7875582
Árbol de decisión mejorado	0.8614139	0.9339792
Random Forest	0.8987446	0.9642460
Red Neuronal Artificial	0.8899075	0.9605879
Red Neuronal Artificial mejorado	0.8905682	0.9608424
KNN	0.7737859	0.8543283
Naive Bayes	0.8047572	0.8931037
GLM	0.7822927	0.8721486

3.2 Utilización de conocimiento primitivo

```
kable(acc_table_alt_2)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.6538652	0.6504694
Árbol de decisión mejorado	0.7002808	0.7732742
Random Forest	0.7726297	0.8538873
Red Neuronal Artificial	0.7518996	0.8261803
Red Neuronal Artificial mejorado	0.7352990	0.8176307
KNN	0.7329039	0.8044313
Naive Bayes	0.7186158	0.7955711
GLM	0.6693095	0.7285056

3.3 Utilización de conocimiento derivado

```
kable(acc_table_alt_3)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.8173109	0.8257992
Árbol de decisión mejorado	0.8890816	0.9537824
Random Forest	0.9159234	0.9740465
Red Neuronal Artificial	0.9036174	0.9673459
Red Neuronal Artificial mejorado	0.9005616	0.9648644
KNN	0.8487777	0.9240923
Naive Bayes	0.8554675	0.9374972
GLM	0.7562768	0.8466590

3.4 Selección aleatoria de instancias

```
kable(acc_table_alt_4)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.8592965	0.8706422
Árbol de decisión mejorado	0.8341709	0.8914883
Random Forest	0.9095477	0.9598369
Red Neuronal Artificial	0.8492462	0.9436290
Red Neuronal Artificial mejorado	0.7989950	0.9084608
KNN	0.7587940	0.8164118
Naive Bayes	0.8592965	0.9222222
GLM	0.7788945	0.8612640

3.5 Realizar un balanceo de datos

```
kable(acc_table_alt_5)
```

Modelos	Accuracy	AUC
Árbol de decisión	0.7542433	0.7822060
Árbol de decisión mejorado	0.8604610	0.9320436
Random Forest	0.8983384	0.9633150
Red Neuronal Artificial	0.8902090	0.9604001
Red Neuronal Artificial mejorado	0.8880650	0.9592596
KNN	0.7672860	0.8503103
Naive Bayes	0.7943541	0.8856899
GLM	0.7788101	0.8682564

4 Conclusiones

Tras analizar los datos, realizar un análisis exploratorio, realizar un preprocesamiento de los datos y finalmente realizar varios modelos, se han extraído una serie de conclusiones sobre el problema planteado. Además, realizar diferentes alternativas que se han considerado oportunos nos permiten también extraer una nueva fuente de información de cara a posibles conclusiones adicionales que también se han valorado.

Tras analizar los diferentes modelos, se han extraído las siguientes conclusiones:

- Un modelo de *árboles de decisión* simple no llega a obtener unos grandes resultados pese a tener un rendimiento aceptable. Sin embargo, emplear diferentes parámetros o llegar a explorar todo el árbol en su profundidad (con el parámetro $cp=-1$) nos permite observar la potencia de esta herramienta y alcanzar unos buenos resultados que saben distinguir los bosones del ruido.
- El modelo con mejores resultados es *Random Forest*, ya que con su promedio de árboles de decisión, y sus diferentes decisiones, es capaz de crear un modelo complejo que representa correctamente el conjunto de datos, llegando a obtener en torno a un 90% de acierto en los datos de entrenamiento que se han empleado en este trabajo.

- El modelo de *Red Neuronal* obtiene también grandes resultados, aunque en diferentes alternativas la mejora de parámetros obtiene mejores o peores resultados, por lo que en este problema la parametrización o bien no ha sido la óptima, o bien no llega a producir una diferencia notable en la clasificación. Sin embargo en base al coste computacional del algoritmo y los resultados obtenidos, sería una gran segunda opción por detrás del previamente mencionado *Random Forest*.
- Los modelos de *KNN*, *Naive Bayes* y *GLM* si bien obtienen unos resultados aceptables, al tratarse de modelos muy simples no llegan a comprender la complejidad del conjunto de datos y obtienen unos resultados considerablemente inferiores al resto de modelos más complejos, por lo que no serían una opción aceptable.

Por otro lado, tras analizar las diferentes alternativas, se puede extraer que:

- El mejor resultado se produce en la alternativa que solo contempla las variables derivadas, es decir, la información aportada por los expertos, por lo que quizás un mayor estudio y la utilización de datos expertos podría conllevar una mejora en la predicción de bosones. Con el conjunto de entrenamiento se ha llegado a alcanzar una precisión del 91,59%.
- El peor resultado de promedio se ha obtenido en la alternativa del conocimiento primitivo exclusivamente, por lo que se puede concluir con que es necesario un análisis de los datos por los expertos de cara a poder obtener información más fiable y que permita deducir la predicción de un bosón con mayor exactitud.
- Se produce una mejora en la selección aleatoria de instancias, de la cual se puede deducir que ciertos subconjuntos de datos pueden llegar a ser más prometedores que otros de cara a una predicción y quizás se debería estudiar posibles relaciones entre los datos.
- Si bien con el balanceo de datos no se llega a producir una mejora significativa de los resultados, nos da una mayor fiabilidad de cara a evitar posibles etiquetados erróneos, lo cual es positivo frente al desconocimiento de este aspecto sobre el planteamiento inicial.

Finalmente, para concluir cabe indicar la importancia de la reducción de datos en las fases de supresión de valores perdidos y detección de outliers. Quizás empleando técnicas de discretización se podrían alcanzar mejores resultados pero requiere de un conocimiento previo de la materia que no se posee como se explica en dicho apartado.

Cabe destacar finalmente que se trata de un problema con un alto coste computacional y que ha requerido cierto tiempo al haberse entrenado tantos modelos y en tantas alternativas, por lo que quizás este tipo de problemas sería mejor afrontarlos desde una máquina preparada para ello, ya que cada ejecución en un ordenador personal con unos buenos componentes de promedio podría alcanzar unos tres cuartos de hora, siendo unos resultados inviables en un ámbito más profesional.

5 Bibliografía

- [1] Repositorio oficial de Caret en GitHub.
- [2] Outlier Treatment - Selva Prabhakaran.
- [3] NoiseFilterR Package.
- [4] How, When, and Why Should You Normalize / Standardize / Rescale Your Data?
- [5] ols_step_both_p: Stepwise regression.
- [6] rpart: Recursive Partitioning and Regression Trees.
- [7] ranger: Ranger.
- [8] nnet: Fit Neural Networks.

- [9] kNN: k-Nearest Neighbour Classification.
- [10] ejemplo naive Bayes - Mauro Valencia.
- [11] Modelos lineales generalizados - Jessica Nathaly Pulzara Mora.
- [12] Sample n rows from a table.
- [13] Subsampling For Class Imbalances.