Sistemas Inteligentes para la Gestión de la Empresa

UNIVERSIDAD DE GRANADA

Práctica 1:
Pre-procesamiento de datos y clasificación binaria

Autor
Juan Manuel Castillo Nievas

Máster Profesional en Ingeniería Informática 2020-2021

—

Granada, 11 de abril de 2021

Índice

1. Introducción

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

Se ha trabajado con el conjunto de datos ofrecido en la competiciónde **Kaggle Higgs Boson Machine Learning Challenge** [1]. Este dataset está formado por **33 variables**, siendo una de ellas una variable binaria clasificadora que indica si la instancia es un **bosón** ('s') o es **ruido de fondo** ('b'). Todas las variables son **numéricas** exceptuando la variable binaria clasificadora que es de tipo **carácter**. Todos los **valores perdidos** se codifican con el número **-999.0**.

2. Lectura y visualización de los datos

Se ha procedido a leer el archivo `training.csv` para posteriormente hacer una visualización general de los datos.

```
training_data_raw <- read_csv('training.csv')
training_data_raw
```

En la Figura 1 se puede ver la salida de la lectura de datos. Aunque la visualización no se pueda ver entera por razones obvias, se pueden observar un total de **250.000 instancias** y **33 variables**.

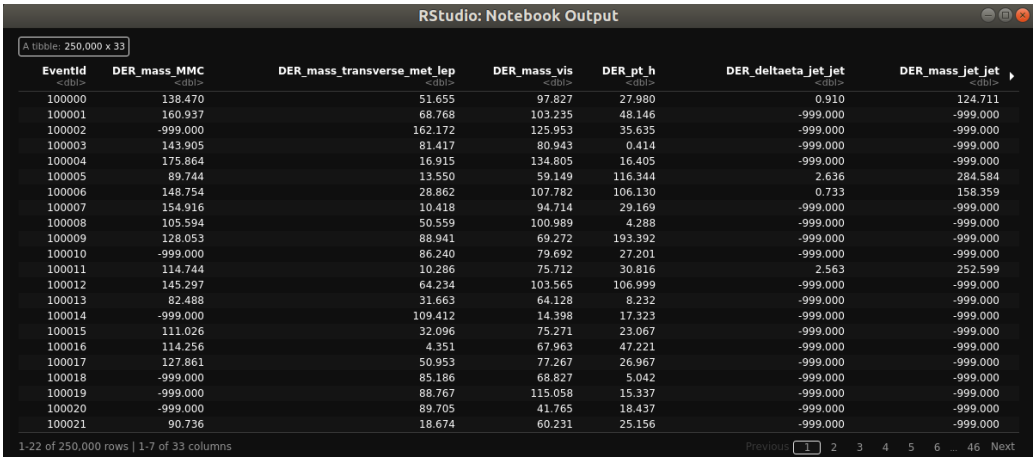

Figura 1: Visualización de datos

En la Figura 2 se puede ver un gráfico en el que se muestran los porcentajes de instancias que pertenecen a la clase **bosón** y **ruido de fondo**. Se puede observar que hay más ruido de fondo que bosones. Realmente podría considerarse un **equilibro de clases no balanceado** ya que las instancias clasificadas como ruido son el doble de aquellas clasificadas como bosón.

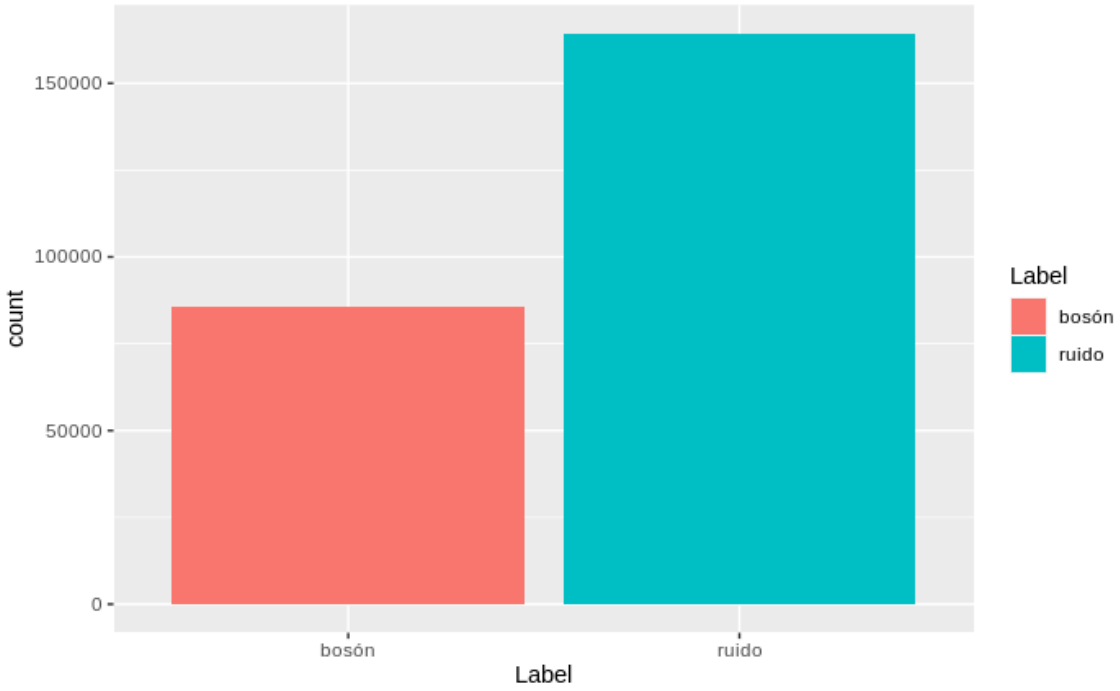

Figura 2: Visualización de la variable clasificadora

3. Preprocesamiento

3.1. Transformación y limpieza de los datos

Lo primero que se ha hecho ha sido convertir la variable clasificadora (columna *Label*) en **factor**, asignando un **1** a las etiquetas cuyo valor es **'s'** y **0** a aquellas cuyo valor es **'b'**. También se han convertido en **valores perdidos** aquellos campos cuyo valor es **-999.0**.

```
training_data <- training_data_raw %>%
    mutate(Label = as.factor(ifelse(Label == 's', 1, 0))) %>%
    na_if(-999.0)

status <- df_status(training_data)
```

A través de la función `df_status` se pueden identificar los valores perdidos y alguna información más. Su salida se muestra en la Figura 3. Hay algunas observaciones interesantes:

- Hay **11 variables con valores perdidos**.
- Hay un **65.73 %** de filas cuyo valor en *Label* es **0**, que indica que **no es un bosón de Higgs** y por lo tanto es ruido.
- **Los valores de la variable *EventID* son únicos**, y otras variables presentan un porcentaje de valores únicos bastante alto.

RStudio: Notebook Output

Description: df[,9] [33 × 9]

variable <chr>	q_zeros <int>	p_zeros <dbl>	q_na <int>	p_na <dbl>	q_inf <int>	p_inf <dbl>	type <chr>	unique <int>
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	164333	65.73	0	0.00	0	0	factor	2

33 rows

Figura 3: Salida de **df_status(training_data)**

3.2. Imputación de valores perdidos

Antes de iniciar cualquier preprocesamiento, primero se van a imputar valores perdidos para trabajar sobre un dataset completo. Se ha utilizado la librería **MICE** tal y como se ha visto en clase. Como método de imputación, se ha usado la **media**, es decir, que se imputan los valores perdidos con la media del resto de valores.

Como dato, en un primer intento se ha imputado con el método *cart*, pero además de tardar demasiado tiempo, mi ordenador se ha quedado sin suficiente memoria RAM y no termina.

```
# Imputar valores perdidos
imputation <- mice(training_data,
                   method = c("","mean","","","","mean","mean","mean","","","",
                              "","","mean","","","","","","","","","","","mean",
                             "mean","mean","mean","mean","mean","","","",""))

training_data <- complete(imputation)
```

3.3. Selección de variables

Partiendo de la información anterior, primero se ha procedido a eliminar aquellas columnas que tienen **más de un 70 % de valores únicos**.

```
# Identificar columnas que tienen más de un 70% de valores únicos
dif_cols <- status %>%
  filter(unique > 0.7 * nrow(training_data)) %>%
  select(variable)

# Eliminar columnas
remove_cols <- bind_rows(
  list(dif_cols)
)
training_data_clean <- training_data %>%
  select(-one_of(remove_cols$variable))
```

En esta Sección se va a hacer una selección de variables utilizando dos métodos. En el primer método se van a seleccionar variables teniendo en cuenta la **correlación de ellas con la variable objetivo** y la **correlación entre ellas mismas**. En el segundo método se va a valorar la **importancia de las variables con un modelo de predicción**.

3.3.1. Correlaciones

Al final de esta Sección, la selección de variables obtenida se guarda en la variable llamada **training_data1** y se queda con un total de **10 variables** (además de la variable clasificadora).

Alta correlación con la variable objetivo

La idea básica es quedarse con aquellas variables que son buenos predictores de la variable objetivo. Para ello, primero se eliminan las filas del conjunto de datos que tienen valores perdidos y se transforman en numérico todas las variables. En este caso todos las variables son numéricas a excepción de la variable clasificadora que está en factor.

En la Figura 4 se puede ver la salida de la tabla de correlación. Se obtiene una tabla con un número que indica la correlación entre la variable y la variable clasificadora, cuyo nombre es **Label**. Al ver los números obtenidos, se ha decidido quedarse con **aquellas variables que presenten una correlación mayor o igual a 0.05**. En total se quedan **15 variables**.

```r
# Eliminar filas con valores perdidos y convertir en numérico los factores
# (la variable clasificadora)
training_data_num <- training_data_clean %>%
  na.exclude() %>%
  mutate_if(is.factor, as.numeric)

# Crear tabla de correlación
cor_target <- correlation_table(training_data_num, target='Label')

cor_target
```

```r
# Quedarse con aquellas variables que su correlación es >= 0.05
important_vars <- cor_target %>%
  filter(abs(Label) >= 0.05)
training_data1 <- training_data_clean %>%
  select(one_of(important_vars$Variable))
```

RStudio: Notebook Output

Description: df[,2] [31 × 2]

Variable <chr>	Label <dbl>
Label	1.00
DER_met_phi_centrality	0.27
PRI_tau_pt	0.24
DER_pt_h	0.19
DER_deltaeta_jet_jet	0.19
DER_mass_jet_jet	0.18
DER_lep_eta_centrality	0.17
DER_sum_pt	0.15
PRI_jet_num	0.13
PRI_jet_all_pt	0.13
PRI_jet_leading_pt	0.09
PRI_met	0.02
DER_mass_MMC	0.01
DER_deltar_tau_lep	0.01
PRI_met_phi	0.01
PRI_tau_eta	0.00
PRI_tau_phi	0.00
PRI_lep_eta	0.00
PRI_lep_phi	0.00
PRI_jet_leading_eta	0.00
PRI_jet_leading_phi	0.00
PRI_jet_subleading_eta	0.00
PRI_jet_subleading_phi	0.00
DER_mass_vis	-0.01
PRI_jet_subleading_pt	-0.01
DER_pt_tot	-0.02
PRI_lep_pt	-0.03
DER_prodeta_jet_jet	-0.17
DER_pt_ratio_lep_tau	-0.20
DER_mass_transverse_met_lep	-0.35
Weight	-0.63

1-31 of 31 rows

Figura 4: Salida de la función **correlation_table**

Alta correlación entre sí

La idea es no quedarse con variables que ofrecen una información muy similar. Al igual que en la anterior Sección, lo primero es eliminar instancias con valores perdidos y convertir la variable factor en numérica. Esta vez se ha utilizado la función **rcorr** para crear la matriz de correlación y **corrplot** para visualizarla, para la cual hay que hacer una previa transformación a `as_tibble`.

```
# Eliminar filas con valores perdidos y convertir en numérico los factores
# (la variable clasificadora)
training_data_num <- training_data1 %>%
  na.exclude() %>%
  mutate_if(is.factor, as.numeric)

# Crear tabla de correlación
rcorr_result <- rcorr(as.matrix(training_data_num))

# Visualizar tabla de correlación
cor_matrix <- as_tibble(rcorr_result$r, rownames = "variable")
corrplot(rcorr_result$r, order = "original", tl.col = "black", tl.srt = 45)
```

En la Figura 5 se muestra la tabla de correlación obtenida.

A continuación se van a agrupar las variables según el valor de correlación entre ellas usando el **coeficiente de Pearson**. Se usa la función **varclus** que crea un **cluster jerárquico** a partir de la similitud teniendo en cuenta la correlación entre ellas.

En la Figura 6 se muestra el **cluster jerárquico** obtenido. Viendo la jerarquía, se ha decidido cortar el árbol indicando que se quieren obtener **10 grupos**, dejando que la función interprete cómo debe realizar el corte.

```
# Cluster jerárquico a partir de la similitud de las variables
v <- varclus(as.matrix(training_data_num), similarity="pearson")
plot(v)

# Cortar el cluster jerárquico obteniendo 10 grupos
groups <- cutree(v$hclust, k = 10)

# Quedarse con una variable representativa de cada grupo
not_correlated_vars <- enframe(groups) %>%
  group_by(value) %>%
  sample_n(1)
training_data1 <- training_data1 %>%
  select(one_of(not_correlated_vars$name))
```

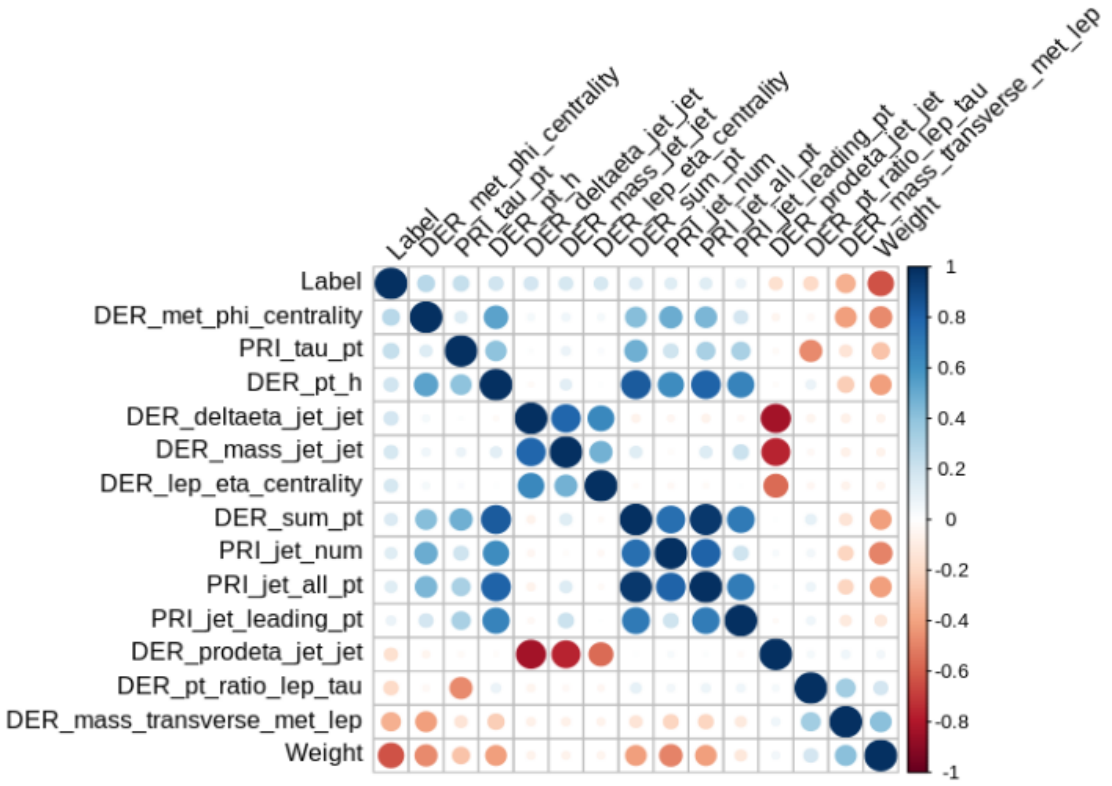

Figura 5: Salida de la función **rcorr**

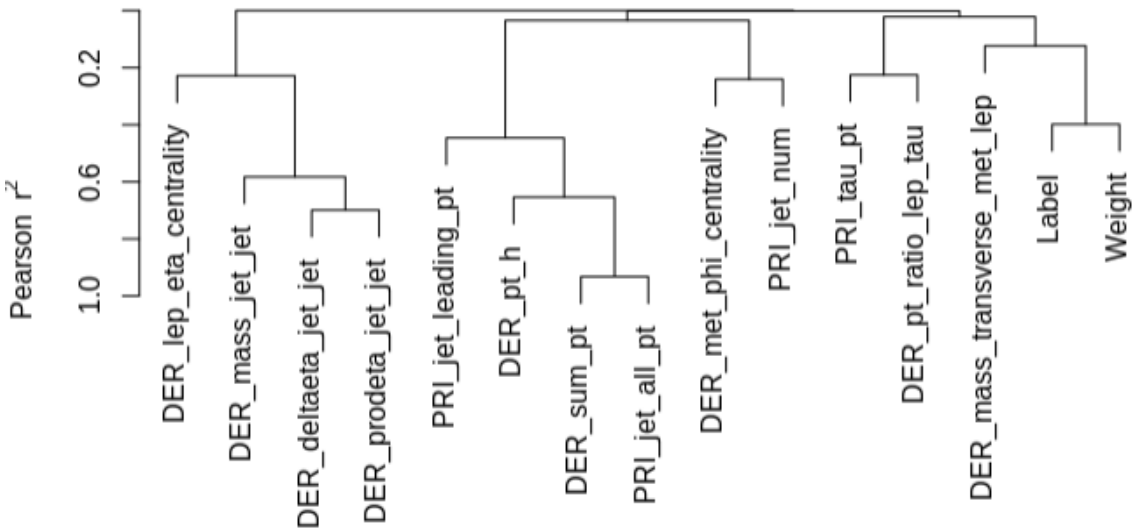

Figura 6: Cluster jerárquico

3.3.2. Importancia de las variables con modelo de predicción

La idea de este proceso es **entrenar un modelo de predicción** que evalúa la importancia de las variables. Al final del proceso, la variable llamada **training_data2** contendrá el resultado de esta selección de variables y contará con sólamente **1 variable** (además de la variable clasificadora).

Lo primero es hacer un pequeño preprocesamiento para eliminar aquellas filas que presenten valores perdidos y transformar la variable objetivo poniendo **'No'** cuando se trata de ruido de fondo, y **'Yes'** cuando se trata de un bosón.

```
# Preprocesamiento
training_data2 <-
  training_data_clean %>%
  mutate(Label = as.factor(ifelse(Label == 0, 'No', 'Yes'))) %>%
  na.exclude()
```

A continuación se crea el modelo predictivo. Como parámetro **cp** se ha usado un valor de **0.01**, ya que este indica la complejidad del árbol. Un valor tan pequeño indica que se quiere un **árbol muy complejo**. También se ha dividido el **conjunto de entrenamiento y validación** en un **80 %-20 %**, respectivamente.

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

# Conjuntos de entrenamiento y validación
trainIndex <- createDataPartition(training_data2$Label, p = .8, list = FALSE,
                                  times = 1)
train <- training_data2[trainIndex, ]

# Entrenamiento del modelo
rpartModel <- train(Label ~ .,
                    data = train,
                    method = "rpart",
                    metric = "ROC",
                    trControl = rpartCtrl,
                    tuneGrid = rpartParametersGrid)
# Visualización del modelo
rpart.plot(rpartModel$finalModel)
```

En la Figura 7 se puede ver la **visualización del modelo creado**. A pesar de haber indicado una complejidad del árbol grande, el resultado es un **árbol muy simple** con simplemente **un nodo**

de decisión.

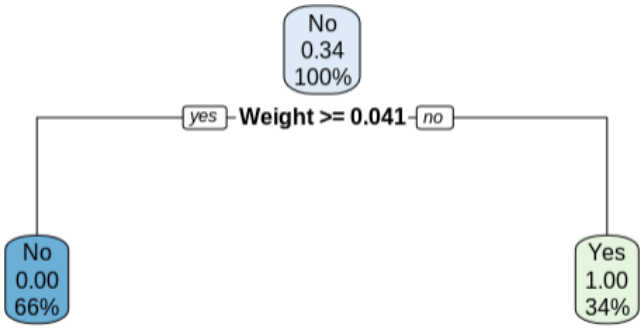

Figura 7: Visualización del modelo creado

A continuación, lo que se debe hacer es **obtener resultados** con el **conjunto de validación** y observar las **métricas** obtenidas. Se han tenido en cuenta dos métricas:

- **Accuracy**: usando la **matriz de confusión** se obtiene el porcentaje de errores y aciertos.
- **Curva ROC:** usando la métrica **AUC**, es decir, el **área bajo la curva**.

```r
# Predicciones con clases
val <- training_data2[-trainIndex, ]
prediction <- predict(rpartModel, val, type = "raw")

# Predicciones con probabilidades
predictionValidationProb <- predict(rpartModel, val, type = "prob")

# Matriz de confusión
cm_train <- confusionMatrix(prediction, val[["Label"]])
cm_train

# Curva ROC
auc <- roc(val$Label, predictionValidationProb[["Yes"]], levels = unique(val[["Label"]]))
roc_validation <- plot.roc(auc,
                           ylim=c(0,1),
                           type = "S" ,
                           print.thres = TRUE,
                           main=paste('Validation AUC:', round(auc$auc[[1]], 2)))
```

La **matriz de confusión** se muestra en la Figura 8. Se puede observar que la precisión del modelo es del **100 %**, es decir, **acierta siempre**.

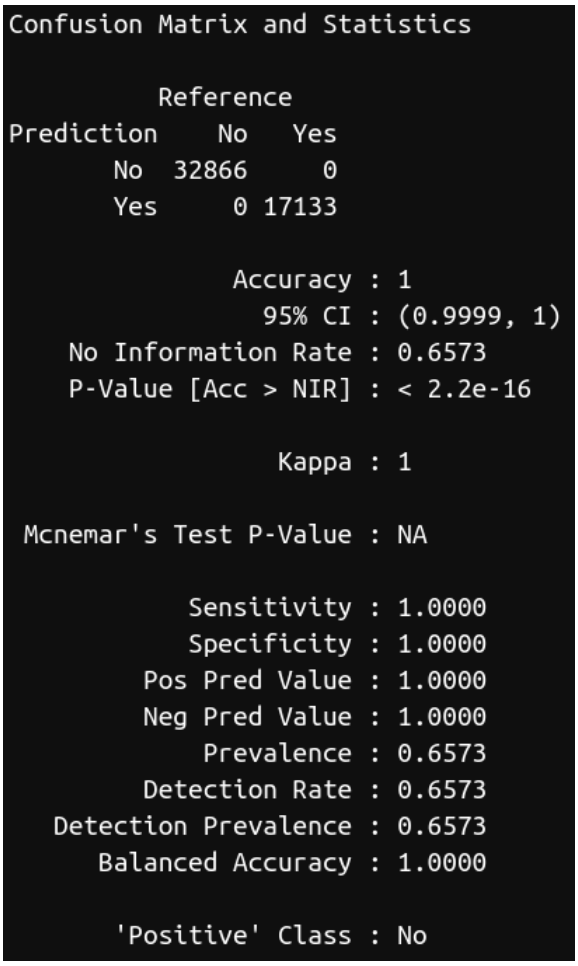

Figura 8: Matriz de confusión

La **curva ROC** se muestra en la Figura 9. El **área bajo la curva** es de **1**, siendo el mejor valor que se puede obtener.

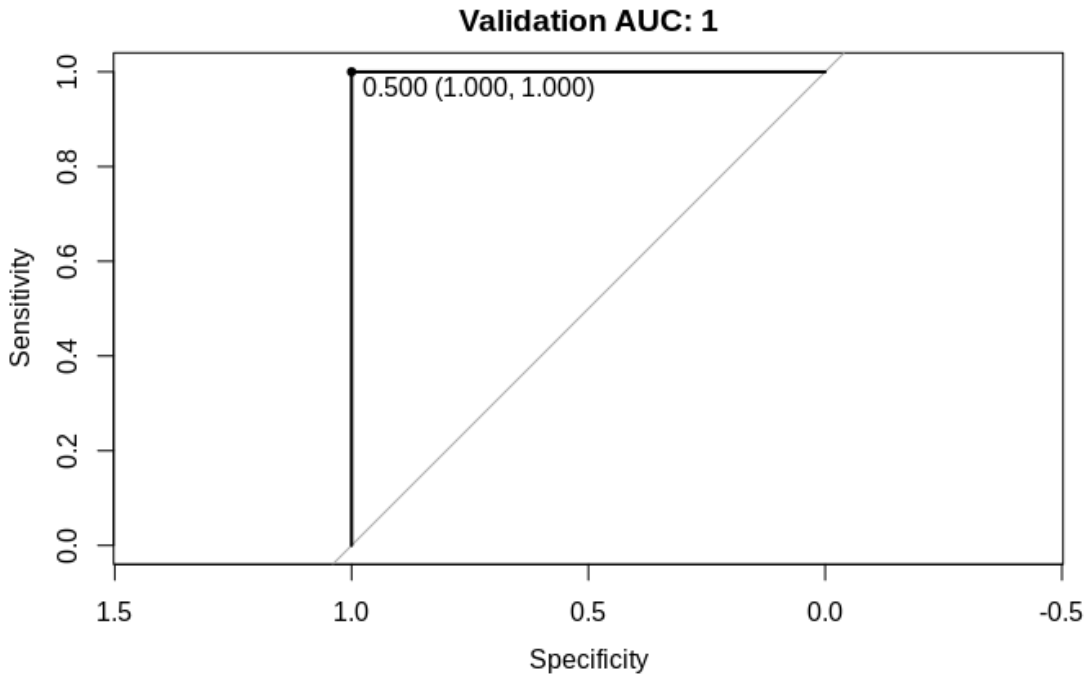

Figura 9: Curva ROC

El paso final sería ver la importancia que se le otorga a cada variable, aunque podemos esperar que la variable que más se utilice sea **Weight**. En la Figura 10 se muestran las variables con su importancia.

Tal y como se podía preveer, la variable **Weight** tiene una importancia del 100 %, mientras que la mayoría tiene una importancia de 0. Se ha decidido quedarse solamente con **aquellas variables tengan una importancia mayor al 50 %**, es decir, sólamente con **Weight**.

```
# Importancia de las variables
varImp(rpartModel)

# Quedarse con las variables cuyo Overall > 50
important_vars <- varImp(rpartModel)$importance %>%
  rownames_to_column() %>%
  filter(Overall > 50) %>%
  select(rowname)
training_data2 <- training_data_clean %>%
  select(one_of(important_vars$rowname), Label)
```

	Overall <dbl>
Weight	100.000000
DER_mass_transverse_met_lep	13.779862
PRI_tau_pt	8.153197
DER_met_phi_centrality	8.123489
DER_pt_ratio_lep_tau	6.860989
PRI_jet_num	0.000000
DER_mass_jet_jet	0.000000
PRI_lep_pt	0.000000
PRI_tau_eta	0.000000
PRI_jet_all_pt	0.000000
PRI_lep_eta	0.000000
DER_pt_h	0.000000
PRI_lep_phi	0.000000
DER_pt_tot	0.000000
PRI_tau_phi	0.000000
DER_prodeta_jet_jet	0.000000
DER_deltaeta_jet_jet	0.000000
DER_mass_MMC	0.000000
PRI_met	0.000000
DER_mass_vis	0.000000

20 rows

Figura 10: Importancia de las variables

3.4. Detección de conflictos e inconsistencias en los datos

Para la detección de outliers (valores atípicos), se ha usado un **diagrama de cajas y bigotes**. Este proceso se ha hecho sobre la variable **training_data2** que es el conjunto de datos obtenido después de haber hecho la selección de variables con el método de importancia de variables con modelo de predicción, en el que sólo quedaba la variable **Weight** debido su importancia del 100 %.

Para visualizar el diagrama de cajas y bigotes, se utiliza la función **boxplot** [2]. En la Figura 11 se puede visualizar dicho diagrama.

```
# Diagrama de cajas y bigotes
g_caja<-boxplot(training_data2$Weight, col="skyblue", frame.plot=F)
```

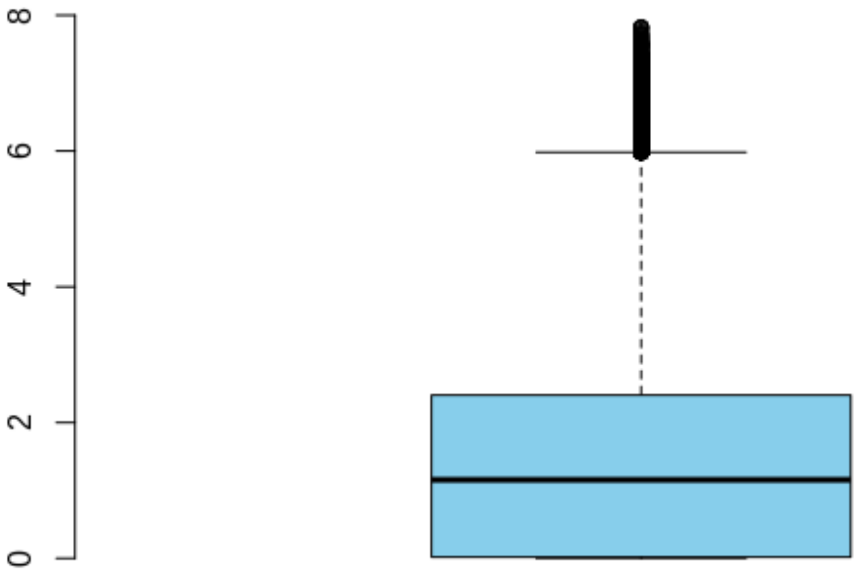

Figura 11: Diagrama de cajas y bigotes

El diagrama de cajas y bigotes agrupa los datos de acuerdo a los **cuartiles**. Los bigotes se usan para indicar cómo varían los datos fuera de los cuartiles superior e inferior. Los **outliers** son los puntos que aparecen fuera de los bigotes.

En este caso, el número de valores atípicos en la variable **Weight** es bastante grande. De hecho, hay **5.452 outliers** en un conjunto de datos de 250.000 instancias, con lo cual es muy alto este índice. Se ha procedido a eliminar dichos valores.

```r
# Eliminar outliers
training_data2<-training_data2[!(training_data2$Weight %in% g_caja$out),]
```

3.5. Selección de ejemplos: downsampling

Ya se vio en la Figura 2 que la variable clasificadora estaba desbalanceada porque hay el **doble de instancias que son ruido** que de instancias que son bosones.

La técnica **downsampling** se usa para el tratamiento de clases no balanceadas. En esta técnica, se reduce el número de muestras de la clase mayoritaria, en este caso de la clase **ruido**. Para ello, se ha usado **caret** y lo que se hace es una selección de ejemplos de manera **aleatoria**.

Esto es algo muy sencillo, pero si luego vamos a obtener un conjunto de entrenamiento y validación a partir de lo obtenido en el **downsampling**, primero se debería particionar el conjunto de validación con el conjunto de datos **antes de aplicar esta técnica**. Si se hace justo después, esto implicaría cierto sesgo ya que se está sacando un conjunto de validación después de haber seleccionando un porcentaje de ejemplos. Entonces el **downsampling** se aplica sobre el conjunto de entrenamiento, que supone el **80 %**.

```r
# Conjunto de entrenamiento y validación de training_data1
train1 <- training_data1 %>%
  head(round(0.8 * nrow(training_data1)))
val1 <- training_data1 %>%
  tail(round(0.2 * nrow(training_data1))) %>%
  mutate(Label = as.factor(ifelse(Label == 1, 'Yes', 'No')))

# Conjunto de entrenamiento y validación de training_data2
train2 <- training_data2 %>%
  head(round(0.8 * nrow(training_data2)))
val2 <- training_data2 %>%
  tail(round(0.2 * nrow(training_data2))) %>%
  mutate(Label = as.factor(ifelse(Label == 1, 'Yes', 'No')))
```

Esta técnica se ha aplicado tanto a **training_data1** como a **training_data2**, puesto que la idea es comparar ambos conjuntos de datos en la clasificación.

Aplicando esta técnica, se obtiene un balanceo de datos en el que el **número de ruido es igual al número de bosones**, que es de **68504** (ver Figura 12).

```r
# Downsampling training_data1
predictors <- select(train1, -Label)
training_data1_down <- downSample(x = predictors, y = train1$Label, yname = 'Label')

table(training_data1_down$Label)

# Downsampling training_data2
predictors <- select(train2, -Label)
training_data2_down <- downSample(x = predictors, y = train2$Label, yname = 'Label')

table(training_data2_down$Label)
```

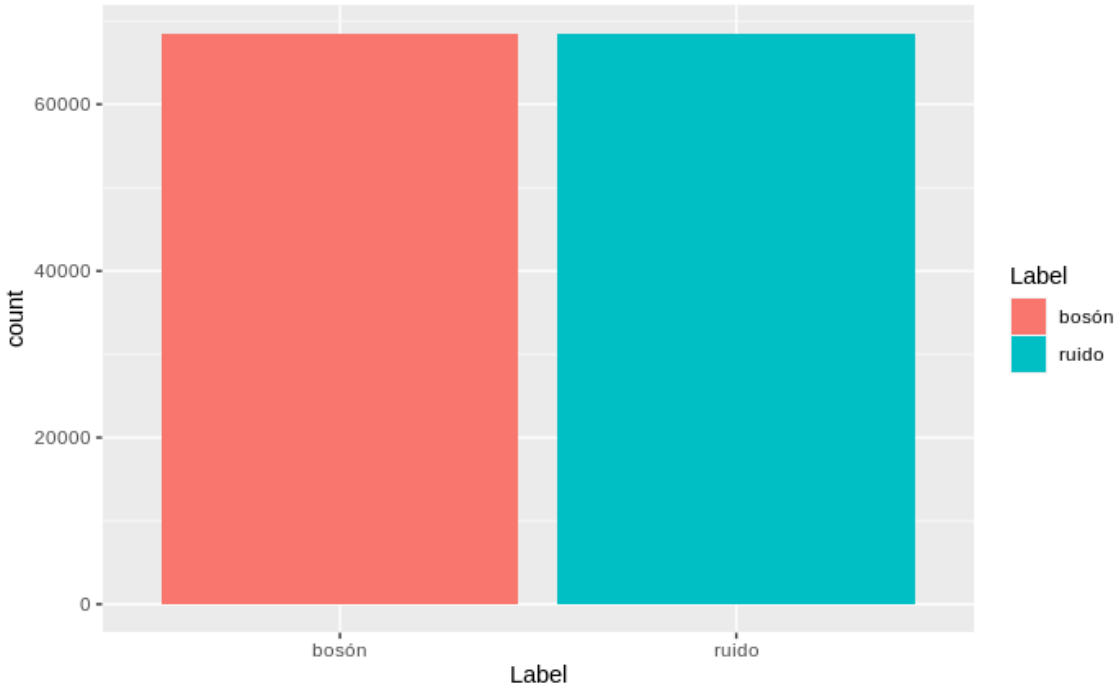

Figura 12: Balanceo de datos con **downsampling**

4. Clasificación

A lo largo de este documento se han ido aplicando distintas técnicas de preprocesamiento. La idea ahora es proponer distintas técnicas de clasificación para comparar los resultados entre ellas y también ver el impacto que supone el haber aplicado un preprocesamiento de los datos.

Se cuenta con tres conjuntos de datos para la clasificación:

- **train**: este es el conjunto de datos inicial, sólamente se han **imputado los valores perdidos**.
- **train1_down**: este conjunto ha sido preprocesado aplicando **selección de variables con correlaciones**, **imputación de valores perdidos** y **downsampling**.
- **train2_down**: este conjunto ha sido preprocesado aplicando **selección de variables con importancia de variables con predicción de modelo**, **eliminación de outliers**, **imputación de valores perdidos** y **downsampling**.

Para realizar la clasificación, es necesario dividir los conjuntos de datos en un **conjunto de entrenamiento** y un **conjunto de validación**. El conjunto de validación en los conjuntos de datos que tienen **downsampling** no es necesario, ya que se tiene porque se ha dividido en la Sección anterior antes de aplicar la técnica. El conjunto de entrenamiento sí es necesario y supone un **80 %** de los datos.

```
set.seed(1234)
trainIndex <- createDataPartition(training_data$Label, p = .7, list = FALSE, times = 1)
train <- training_data[ trainIndex, ]
train <- na.exclude(train) %>%
        mutate(Label = as.factor(ifelse(Label == 1, 'Yes', 'No')))

val <- training_data[-trainIndex, ]
val <- na.exclude(val) %>%
        mutate(Label = as.factor(ifelse(Label == 1, 'Yes', 'No')))

trainIndex <- createDataPartition(training_data1_down$Label, p = .8, list = FALSE, times = 1)
train1_down <- training_data1_down[ trainIndex, ] %>%
        mutate(Label = as.factor(ifelse(Label == 1, 'Yes', 'No')))

trainIndex <- createDataPartition(training_data2_down$Label, p = .8, list = FALSE, times = 1)
train2_down <- training_data2_down[ trainIndex, ] %>%
        mutate(Label = as.factor(ifelse(Label == 1, 'Yes', 'No')))
```

Se van a aplicar dos técnicas de clasificación: **árboles de regresión** (CART) y **el vecino más cercano** (kNN). En un principio también se iba a aplicar la técnica vista en clase **random forest**, pero al ejecutarla en R y tras dejarla una hora funcionando, no ha finalizado, con lo cual se ha optado por descartar esta técnica.

Los árboles de regresión se han visto en clase y el método del vecino más cercano se ha visto en la asignatura del primer cuatrimestre **Tratamiento Inteligente de Datos**. Puede ser curioso comparar estas dos técnicas.

4.1. Árboles de regresión

En esta primera técnica de clasificación se usa **rpart** [3] para implementar árboles de decisión.

Primero se definen unos **parámetros del algoritmo del aprendizaje** con la función **trainControl**. Esto ya se vio un poco en la Sección 3.3.2 cuando se utilizó este modelo de predicción para obtener las variables más importantes. Se debe configurar el parámetro **cp** que indica la **complejidad del árbol**, y su rango de valores suele estar entre 0 y 0.1, siendo 0 la complejidad más alta.

Como el código en R es igual para los tres conjuntos de datos, se procede a poner un ejemplo de este:

```
# Entrenar modelo
rpartModel <- train(Label ~ .,
                    data = train,
                    method = "rpart",
                    metric = "Accuracy",
                    trControl = rpartCtrl,
                    tuneGrid = rpartParametersGrid)

# Predicciones con clases
prediction <- predict(rpartModel, val, type = "raw")

# Predicciones con probabilidades
predictionValidationProb <- predict(rpartModel, val, type = "prob")

# Matriz de confusión
cm_train <- confusionMatrix(prediction, val[["Label"]])
cm_train

# Curva ROC
auc <- roc(val$Label, predictionValidationProb[["Yes"]], levels = unique(val[["Label"]]))
roc_validation <- plot.roc(auc,
                           ylim=c(0,1),
                           type = "S" ,
                           print.thres = TRUE,
                           main=paste('Validation AUC:', round(auc$auc[[1]], 2)))
```

4.1.1. Conjunto de datos train

En este primer conjunto de datos, el porcentaje de aciertos es del **100 %**. Su matriz de confusión y demás datos se muestran en la Figura 13.

En la Figura 14 se muestra la **curva ROC** obtenida, con un área bajo la curva de **1**.

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  7225    0
       Yes    0 6429

               Accuracy : 1
                 95% CI : (0.9997, 1)
    No Information Rate : 0.5291
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5291
         Detection Rate : 0.5291
   Detection Prevalence : 0.5291
      Balanced Accuracy : 1.0000

       'Positive' Class : No
```

Figura 13: Matriz de confusión y precisión

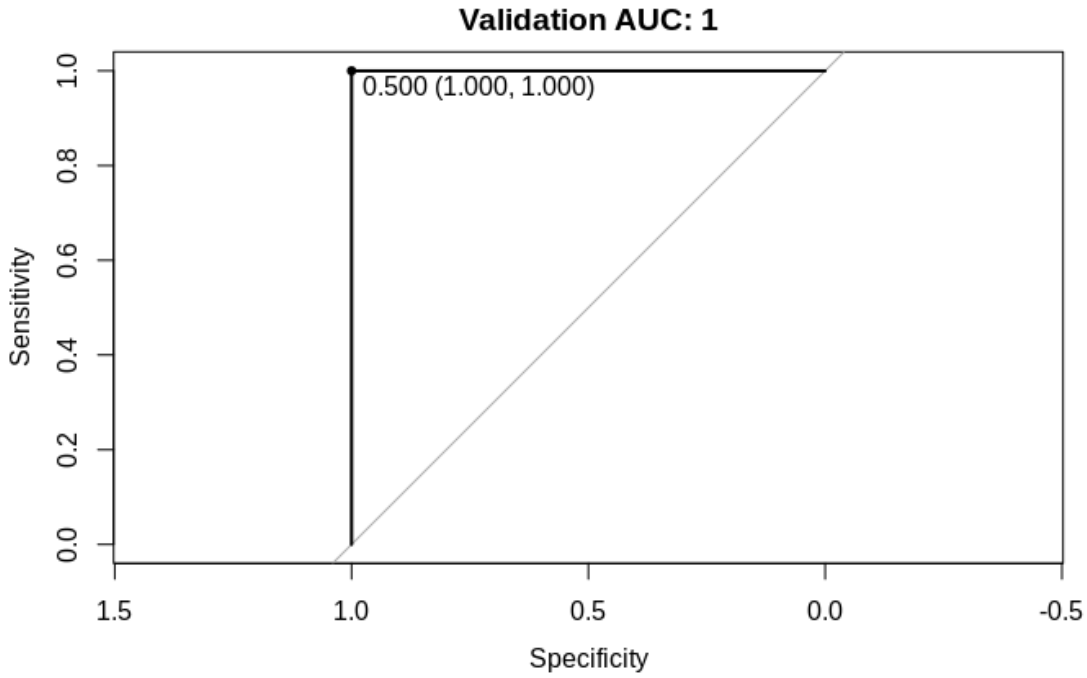

Figura 14: Curva ROC

4.1.2. Conjunto de datos train1_down

En este segundo conjunto de datos, el porcentaje de aciertos es del **100 %**. Acierta absolutamente todos los datos del conjunto de validación. Su matriz de confusión y demás datos se muestran en la Figura 15.

En la Figura 16 se muestra la **curva ROC** obtenida, con un área bajo la curva de **1**. Es una curva perfecta.

```
Confusion Matrix and Statistics

          Reference
Prediction    No   Yes
       No  32837     0
       Yes     0 17163

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.6567
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6567
         Detection Rate : 0.6567
   Detection Prevalence : 0.6567
      Balanced Accuracy : 1.0000

       'Positive' Class : No
```

Figura 15: Matriz de confusión y precisión

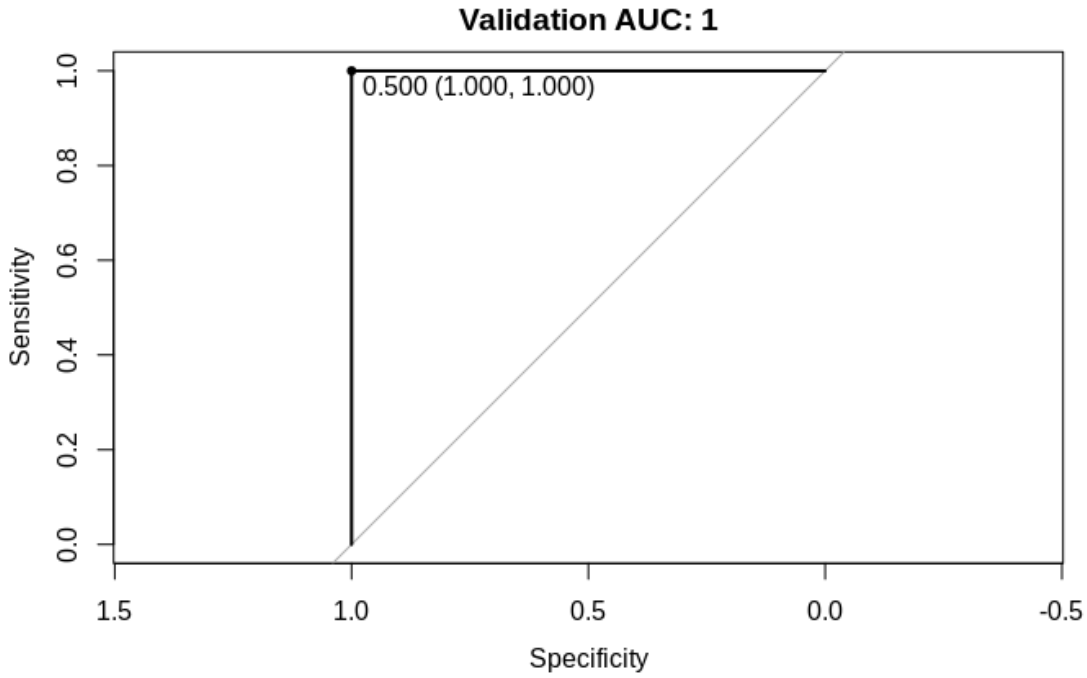

Figura 16: Curva ROC

4.1.3. Conjunto de datos train2_down

En este tercer conjunto de datos, al igual que en el anterior conjunto, el porcentaje de aciertos es del **100 %**. Acierta absolutamente todos los datos del conjunto de validación. Su matriz de confusión y demás datos se muestran en la Figura 17.

En la Figura 18 se muestra la **curva ROC** obtenida, con un área bajo la curva de **1**. Es una curva perfecta.

```
Confusion Matrix and Statistics

          Reference
Prediction    No   Yes
       No  32837     0
       Yes     0 17163

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.6567
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.6567
         Detection Rate : 0.6567
   Detection Prevalence : 0.6567
      Balanced Accuracy : 1.0000

       'Positive' Class : No
```

Figura 17: Matriz de confusión y precisión

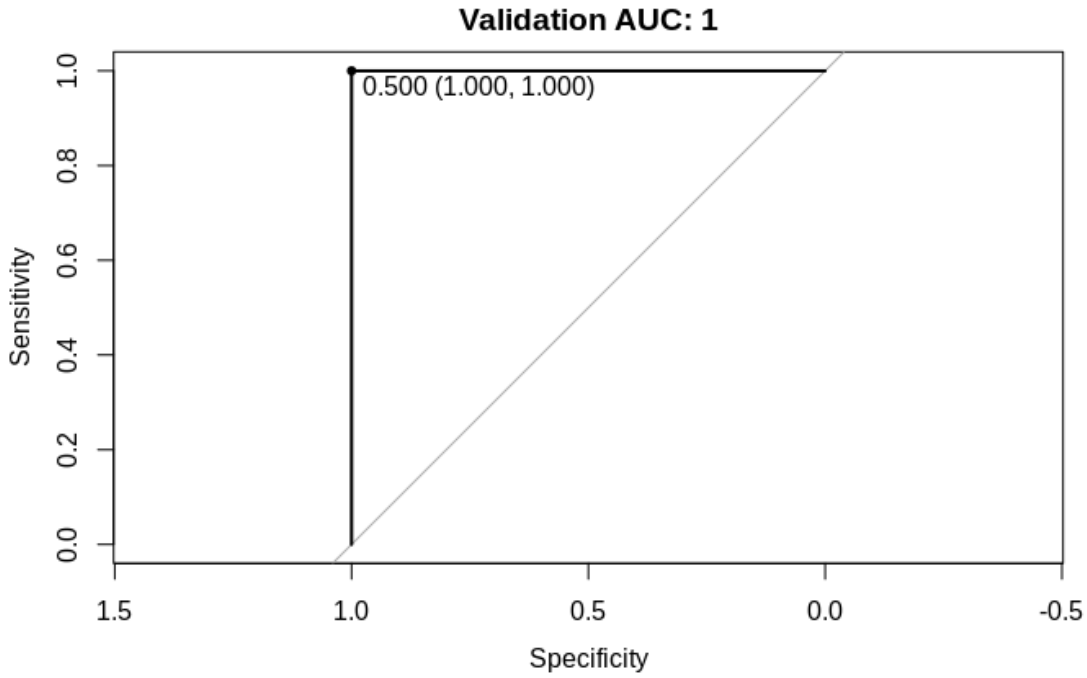

Figura 18: Curva ROC

4.2. Vecino más cercano

Como ya se ha comentado, la técnica **kNN** se vio en el anterior cuatrimestre y se ha decidido usar en contraste con los árboles de regresión. En esta técnica, se seleccionan los **k vecinos más cercanos** y se clasifica con la clase más común entre esos vecinos. El valor de vecinos por el que se ha optado es de **k=5**. Se recuerda que el valor debe ser impar para que así no haya empates de vecinos.

```
# Entrenar modelo
knn=kknn(formula=Label ~ .,train,val,k=5)
table=table(knn$fit,val$Label)

# Calculo porcentaje de aciertos
diag=diag(table)
bien_clasificados=(sum(diag)/nrow(val))*100

# Predicción y curva ROC
library("ROCR")
m1=knn$prob[,2]
pred=prediction(m1,val$Label)
perf=performance(pred,"tpr","fpr")
auc=as.numeric(performance(pred, "auc")@y.values)
plot(perf, main=paste('Validation AUC:', round(auc, 2)))
```

4.2.1. Conjunto de datos train

En este primer conjunto de datos, el porcentaje de aciertos es del **86.04 %**. Su matriz de confusión se muestra en la Figura 19. El índice de aciertos es mucho menor que con el árbol de decisión, casi un 15 % de diferencia.

En la Figura 20 se muestra la **curva ROC** obtenida, con un área bajo la curva del **0.94**.

```
        No   Yes
No   29002  3112
Yes   3864 14021
```

Figura 19: Matriz de confusión

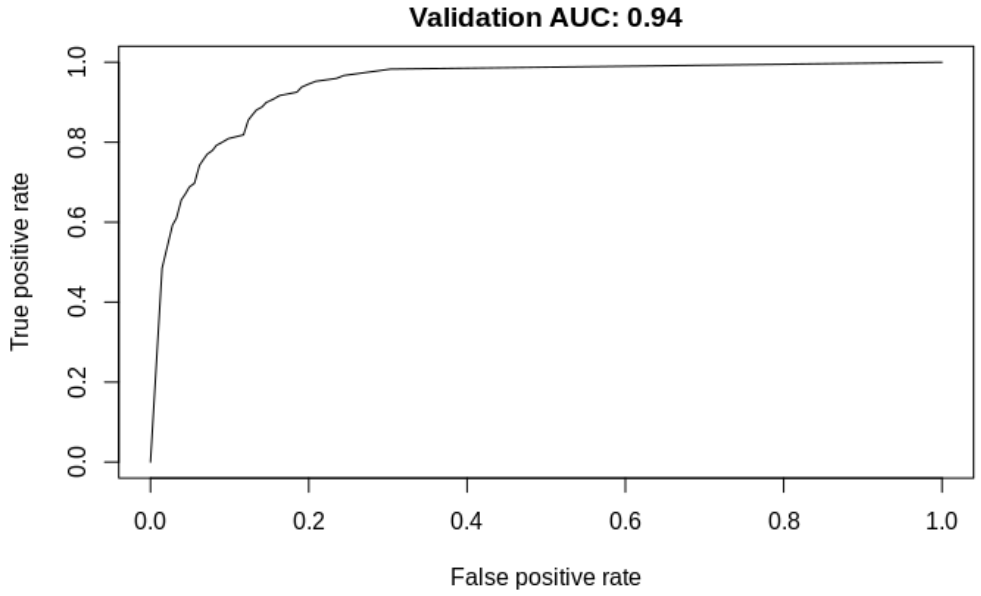

Figura 20: Curva ROC

4.2.2. Conjunto de datos train1_down

En este segundo conjunto de datos, el porcentaje de aciertos es del **89.19 %**. Su matriz de confusión se muestra en la Figura 21. En el árbol de decisión, este conjunto de datos ya obtenía el 100 % de aciertos.

En la Figura 22 se muestra la **curva ROC** obtenida, con un área bajo la curva del **0.95**. A pesar de que el índice de aciertos ha disminuido, el área bajo la curva es bastante bueno.

```
        No   Yes
No   28521  1088
Yes   4316 16075
```

Figura 21: Matriz de confusión

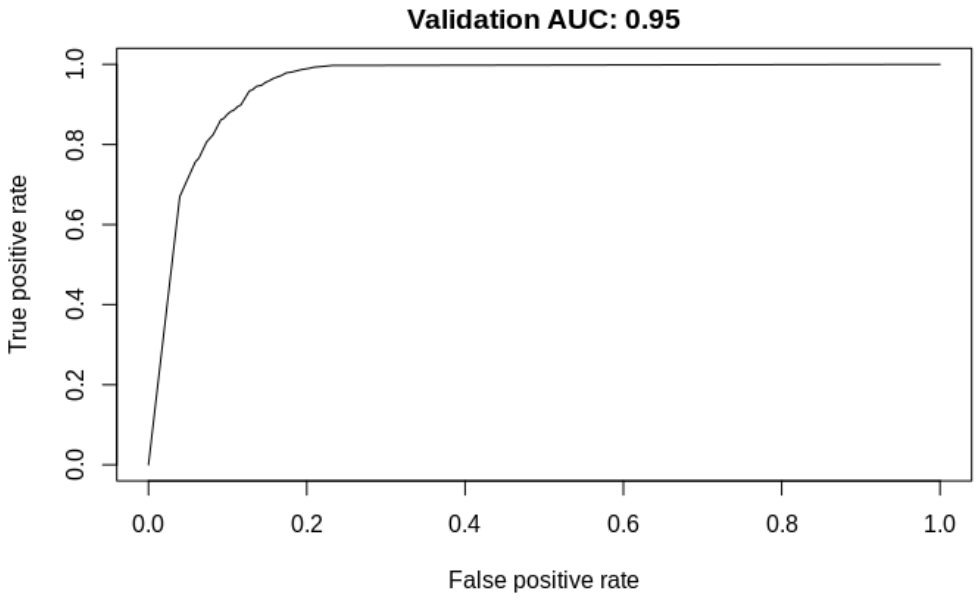

Figura 22: Curva ROC

4.2.3. Conjunto de datos train2_down

En este tercer conjunto de datos, el porcentaje de aciertos es del **100 %**, al igual que en el árbol de regresión. Su matriz de confusión se muestra en la Figura 23.

En la Figura 24 se muestra la **curva ROC** obtenida, con un área bajo la curva del **1**, también al igual que en el árbol de regresión.

```
       No   Yes
No  32837     0
Yes     0 17163
```

Figura 23: Matriz de confusión

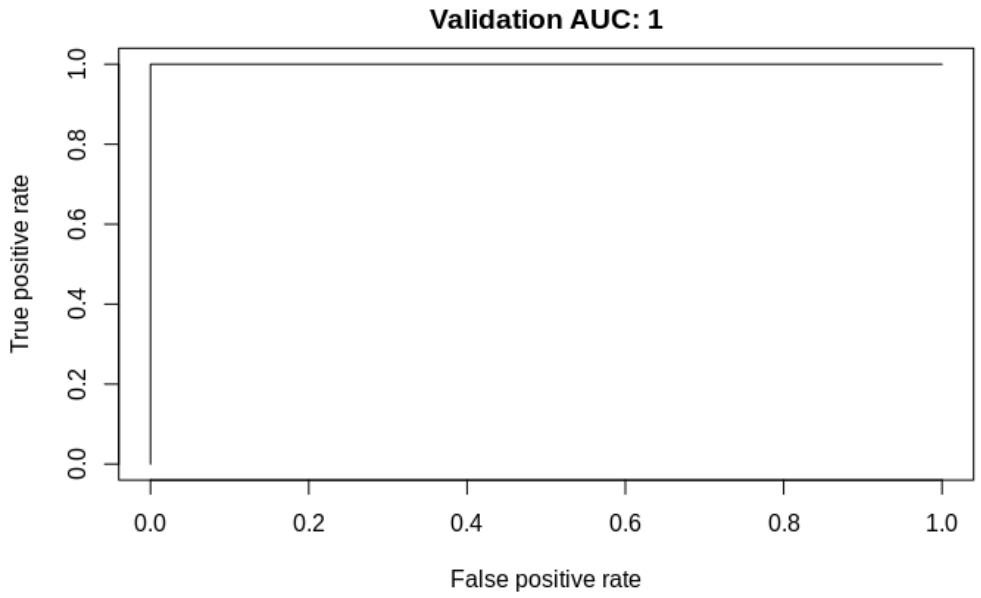

Figura 24: Curva ROC

5. Discusión de resultados

A lo largo de este documento se han visto diferente preprocesamientos de datos aplicadas a dos técnicas de clasificación. Además, al tener un conjunto de datos desbalanceado, se ha sumado la técnica de *downsampling*.

En la Tabla 1 se muestra la comparación de los tres conjuntos de datos (explicados en la Sección 4) de acuerdo a la **precisión** obtenida, es decir, el número de aciertos:

- **CART**: tanto los conjuntos preprocesados como el no preprocesado han obtenido una **clasificación perfecta**. Esto se debe a que los árboles de decisión obtenidos sólamente tienen en cuenta la variable **Weight**, que está presente en todos los conjuntos de datos y que otorga una clasificación correcta siempre.
- **kNN**: el conjunto de datos **no preprocesado** ha obtenido el porcentaje más malo, de **86.04 %**. El **primer conjunto preprocesado** no ha conseguido la perfección tampoco, si no que se ha quedado en un **89.19 %**. El **segundo conjunto preprocesado** sí ha conseguido de nuevo el **100 %**.

En la Tabla 2 se muestra la comparación de los tres conjuntos de datos de acuerdo al área bajo la curva obtenida:

- **CART**: todos los conjuntos de datos han obtenido un área de **1**.
- **kNN**: el conjunto de datos **no preprocesado** ha obtenido un área del **0.94**. El **primer conjunto de datos preprocesado** ha obtenido un área del **0.95**, a diferencia del **1** que se obtuvo en la otra técnica. Por último, el **segundo conjunto de datos preprocesado** sigue alcanzando la perfección, con un área bajo la curva de **1**. En este clasificador es donde mejor se aprecia el **impacto del preprocesamiento**.

	train	**train1_down**	**train2_down**
CART	100	100	100
kNN	86.04	89.19	100

Tabla 1: Comparación de porcentajes de aciertos

	train	**train1_down**	**train2_down**
CART	1	1	1
kNN	0.94	0.95	1

Tabla 2: Comparación de áreas bajo la curva

6. Conclusiones

Partiendo del conjunto de datos inicial en el que había **250.000 instancias** y **33 columnas**, se han hecho dos preprocesamientos distintos para posteriormente aplicar dos técnicas de clasificación y ver cómo influye cada preprocesaminto en los datos.

Para recordar un poco estos conjuntos de datos:

- **train**: este es el conjunto inicial sin ningún preprocesamiento. Sólamente se **imputan los valores perdidos**.
- **train1_down**: aparte de la **imputación de valores perdidos**, en este conjunto se ha aplicado una **selección de variables viendo sus correlaciones**, quedando sólo 10 variables. También se ha aplicado la técnica de ***downsampling***.
- **train2_down**: aparte de la **imputación de valores perdidos**, en este conjunto se ha aplicado una **selección de variables de acuerdo a su importancia con un modelo de predicción**, quedando sólo una variable (aparte de la clasificadora). También se han **detectado y eliminado *outliers***, y por último se ha aplicado ***downsampling***.

En la Sección 5 se compararon los resultados obtenidos. Se han sacado las siguientes conclusiones:

- El conjunto de datos **train2_down** ofrece una **clasificación perfecta** en ambas técnicas de clasificación, y es el conjunto de datos más simple que hay porque sólamente se tiene en cuenta la variable **Weight**.
- El conjunto de datos **train1_down** ofrece una **clasificación perfecta** en **CART**, pero se queda cerca del **90 %** de aciertos en **kNN**.
- El conjunto de datos **no preprocesado** es el que peor clasificación ofrece, demostrando que el **preprocesamiento de datos es algo fundamental**, no solo para disminuir los datos y así mejorar el tiempo de ejecución, si no para conseguir **mejores resultados**. A pesar de su **100 %** de aciertos en **CART**, en **kNN** no ha tenido tanto éxito, obteniendo el peor resultado de todos.
- El preprocesamiento hecho también influye en los datos, pues ya se ha visto que el preprocesamiento aplicado a **train2_down** hace que se obtenga un **100 %** en **ambos clasificadores**.
- Se deben probar **diferentes técnicas de preprocesamiento** de acuerdo a los resultados que se vayan obteniendo. Hay muchos parámetros posibles en ellas y todo va a depender del conjunto de datos que se tenga, **no es una ciencia exacta**.

7. Bibliografía

[1] Kaggle. Higgs boson machine learning challenge. https://www.kaggle.com/c/higgs-boson/, 2012.

[2] César Anderson Huamaní Ninahuanca. Detección y eliminación de valores outliers. https://rpubs.com/Cesar_AHN/deteccion-eliminacion-valores-outliers-en-r-boxplot, 2020.

[3] Juan Bosco Mendoza Vega. Árboles de decisión con r - clasificación. https://rpubs.com/jboscomendoza/arboles_decision_clasificacion, 2018.