Práctica 1:
Pre-procesamiento de datos y clasificación binaria

Curso 2020/2021

PEDRO MANUEL FLORES CRESPO

Sistemas Inteligentes para la Gestión en la Empresa

Máster en Ingeniería Informática

Índice

1. Introducción

Como se indica en el guion de la práctica, partimos de los datos del experimento ATLAS del CERN-LHC y que están disponibles en la competición correspondiente en `Kaggle`. El problema consiste en clasificar si un evento corresponde al decaimiento de un bosón de Higgs (s) o si es ruido (b). Para ello, disponemos de aproximadamente 30 variables diferentes siendo la mayoría de ellas valores reales y un total de 250 000 observaciones diferentes.

En cuanto al planteamiento inicial para resolver el problema y partiendo de las herramientas disponibles vistas en clase se va a seguir el siguiente procedimiento:

1. EDA y visualización: haremos un estudio previo de los datos disponibles basándonos en el archivo `higgs-eda.Rmd` proporcionado.
2. En cuanto a la parte de prepropcesamiento llevaremos a cabo las siguientes tareas:
   a) Eliminar columnas no útiles: aquellas que presenten una gran cantidad de valores perdidos o no presenten valores interesantes.
   b) Imputación de valores perdidos: se rellenarán aquellas datos que no disponemos para suplir posibles inconsistencias.
   c) Discretización de los datos: como veremos más adelante, casi todas las variables son continuas y reales por lo que se llevará a cabo este proceso para facilitar el proceso de entrenamiento del modelo.
   d) Selección de instancias: como podremos ver en el EDA, las clases no están balanceadas así que se va a usar la técnica de *downsampling* para equilibrar ambos conjuntos. En alguna técnica de clasificación como las redes neuronales, el *upsampling* es efectivo ya que es capaz de aprender esos ejemplos "repetidos". Sin embargo, en este caso se va a optar por eliminar instancias de la clase mayoritaria.
   e) Selección de variables: finalmente, se llevará a cabo una selección de variables mediante correlación y PCA y veremos cómo se comportan posteriormente los clasificadores en cada una de ellas.
3. Clasificación: las técnicas de clasificación que se usarán son:
   a) Árboles de decisión (`rpart`).
   b) *Boosted Logistic Regression* (`LogitBoost`).

   Se ha determinado utilizar estas técnicas ya que se quería usar una tanto técnica con la que ya se habría trabajado como con una nueva. Entre las técnicas nuevas se probaron *ensembles* como los vistos en clase, *eXtreme Gradient Boosting*, *AdaBoost Classification Trees*, etc. Sin embargo al disponer de un ordenador no demasiado potente, la técnica seleccionada es la que ha podido completar el entrenamiento satisfactoriamente. En la sección 4.2 se explicará mejor esta técnica.
4. Por último, llevaremos a cabo un análisis de los resultados y obtención de conclusiones.

2. EDA y visualización

Para la parte de análisis exploratorio de los datos y visualización de los mismos vamos a usar los resultados obtenido en el archivo `higgs-eda.Rmd` proporcionado en el repositorio de la asignatura. En primer lugar, eliminamos la columna `Weight` y el identificador ya que no nos serán de ayuda para la clasificación de los eventos. Además, los valores perdidos aparecen como -999.0 por lo que también llevaremos a cabo dicha recodificación. Ahora podemos ver cuántos valores tenemos de cada clase (figura 1). Vemos que tenemos una mayor cantidad de elementos de ruido, lo que tenemos que tener en

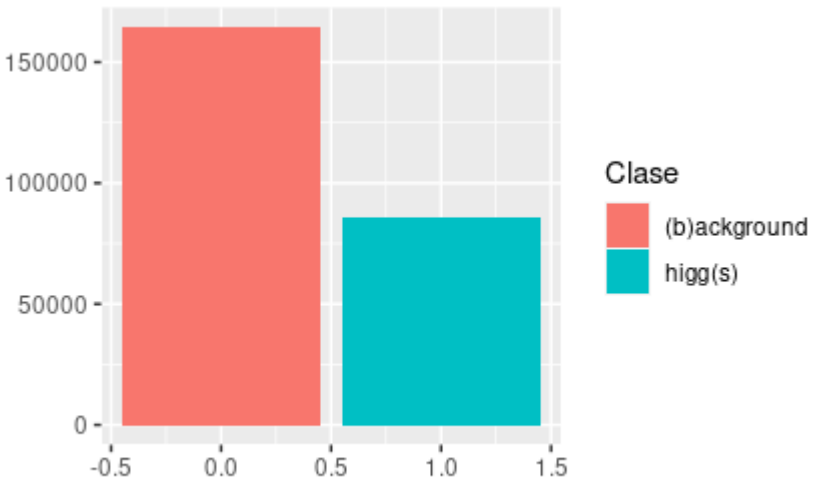


Figura 1: Distribución de los valores a clasificar.

cuenta posteriormente durante el entrenamiento de los modelos. Más concretamente, tenemos un total de 164 333 instancias clasificadas como *background* y 85 667 como higgs.

Podemos tener una idea inicial del estado de los datos mediante la función `df_status` tal y como se muestra en el cuaderno `.Rmd` adjunto de la práctica. Para hacer un estudio más completo sobre el conjunto de los datos con los que estamos trabajando se va a hacer uso de la herramienta DataExplorer que nos genera un informe muy completo. El mismo se adjunto a la entrega de la práctica con el nombre `report.html`. Algunos de los datos relevantes que podemos obtener del mismo son los siguientes:

1. Vemos que existen columnas con una gran cantidad de valores perdidos (algunas de ellas superan más del 70 % de sus valores y algunas el 40 %) tal y como se observa en la figura 2. Debemos considerar posteriormente si eliminamos dichas columnas o podemos rellenar dichos valores con algún valor significativo.
2. Vemos también un análisis de correlaciones entre las distintas variables (lo veremos con más detenimiento usando otras herramientas vistas en la asignatura).
3. Aparece una análisis inicial de componente principales (PCA). Vemos que la componente principal PCA1 explica el 20 % de la varianza del conjunto. Si cogemos cinco tenemos el 50 % mientras que el 80 % se alcanza con 12 (figura 3).

También podemos tener una idea intuitiva sobre cómo de buena puede llegar a ser cada una de las variables a la hora de la predicción dibujando la distribución para cada una de las etiquetas en función de cada variable (sin valores perdidos). Algunos de los gráficos los vemos en la figura 4. Vemos cómo en ciertos casos hay ciertas diferencias entre un tipo de evento u otro. En la sección 3.5, haremos un estudio más detallado sobre selección de variables útiles para llevar a cabo el proceso de clasificación.

3. Preprocesamiento

A continuación, mostramos los pasos del preprocesamiento llevado a cabo.

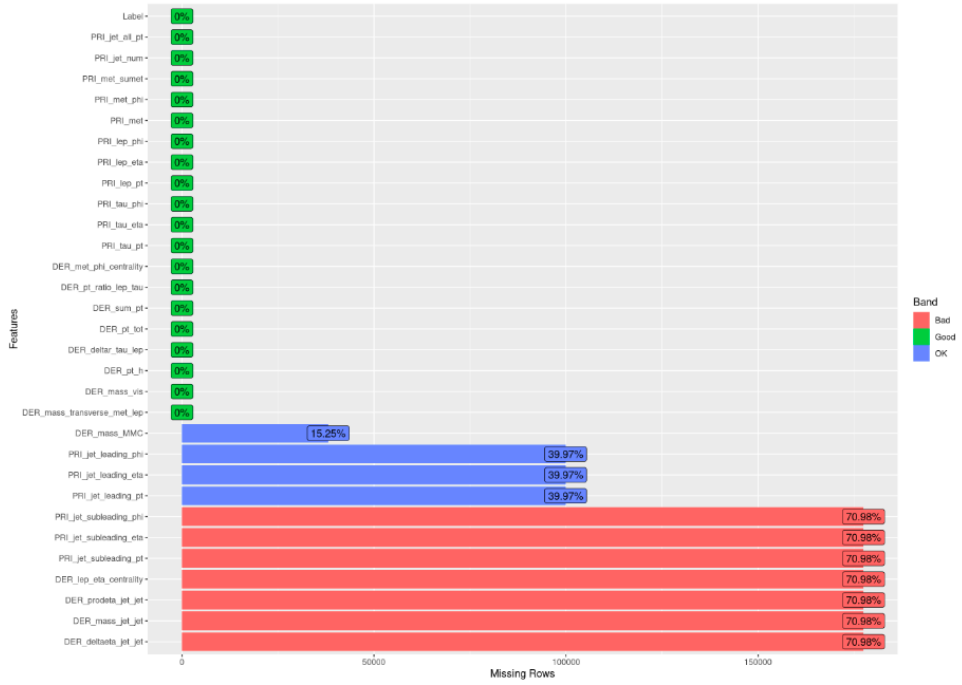


Figura 2: Valores perdidos por característica.

3.1. Eliminar columnas no útiles

El primer paso va a ser eliminar eliminar que no resulten útiles para el proceso. Recordamos, que anteriormente eliminamos directamente del análisis exploratorio las variables *Weight* e identificador ya que sabíamos de antemano que no nos iban a aportar información relevante para el problema. En esta ocasión, nos podemos fijar en algunos aspectos que pueden ser relevantes a través de la función `df_status`:

- Variables con un alto porcentaje de ceros: en nuestro problema, no hemos detectado ninguna que pueda llegar a cumplir esta condición ya que las únicas variables que tienen ese porcentaje elevado son `PRI_jet_num` y `PRI_jet_all_pt` con un 40 %. No consideramos que sea un porcentaje lo suficientemente alto como para eliminarlas.
- Variables con un alto número de valores perdidos: como vimos en la sección anterior (figura 2), hay variables que tienen un porcentaje de valores perdidos del 70 % mientras que otras son del 40 %. Consideramos que el 40 % es un valor aceptable para llevar a cabo la imputación de los datos. Sin embargo, con un 70 % la información imputada no sería demasiado útil. Por ello, se han decidido eliminar dichas columnas. Más concretamente son: `DER_deltaeta_jet_jet`, `DER_mass_jet_jet`, `DER_prodeta_jet_jet`, `DER_lep_eta_centrality`, `PRI_jet_subleading_pt`, `PRI_jet_subleading_eta` y `PRI_jet_subleading_phi`.
- Variables con pocos valores: casi todas las variables son continuas salvo `PRI_jet_num` que toma un valor entero entre 0 y 3 (ambos inclusive). A pesar de tener pocos valores, vemos cómo en la figura 4 puede llegar a tener cierta relación a la hora de predecir la variable objetivo por lo que también se va a mantener.

Así, tras eliminar las siete variables indicadas tenemos actualmente un total de veintitrés. El proceso se muestra en el código 1

```
> status <- status %>%
  filter(variable != 'Label')
> na_cols <- status %>%
```

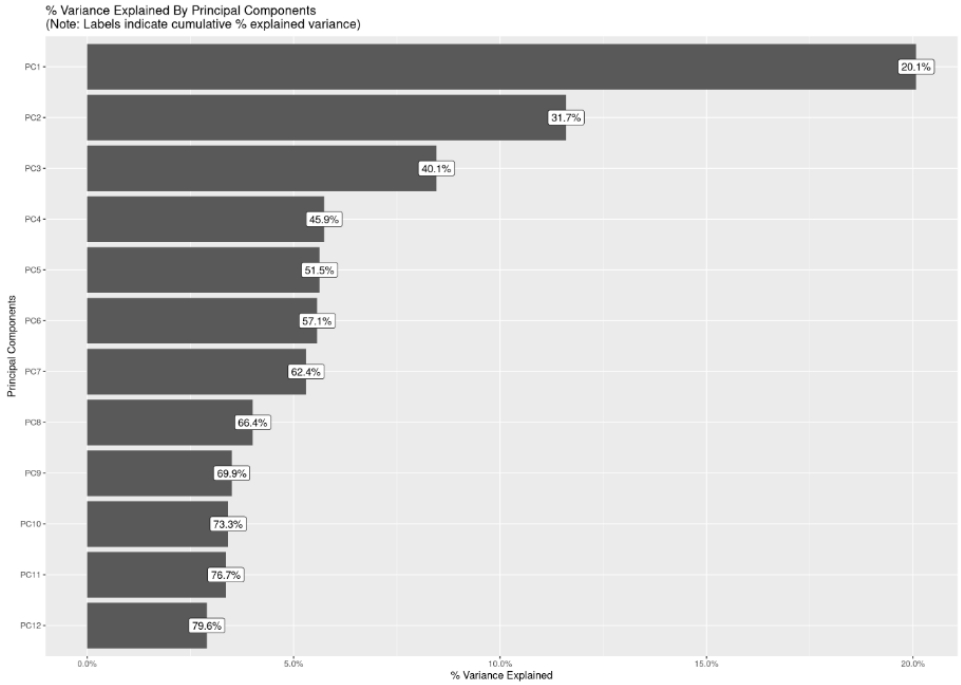


Figura 3: Análisis de componentes principales.

```
  filter(p_na > 70) %>%
  select(variable)
> remove_cols <- bind_rows(
  list(
    na_cols
  )
)

> data <- data %>%
  select(-one_of(remove_cols$variable))
> status <- df_status(data)
```

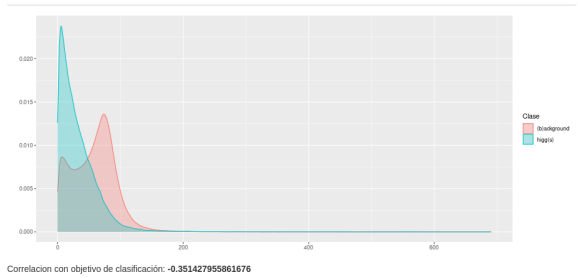
Código 1: Código para eliminar columnas no útiles.

3.2. Imputar valores perdidos

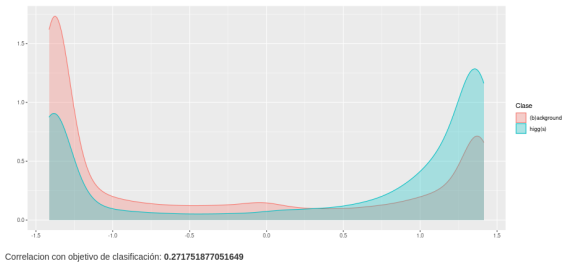
El siguiente paso es imputar los valores perdidos en aquellas columnas que los presente. Dichas variables son (nuevamente observadas mediante `df_status`): `DER_Mmass_MMC`, `PRI_jet_leading_phi`, `PRI_jet_leading_eta` y `PRI_jet_leading_pt`. Para llevar a cabo este proceso, hacemos uso de la herramienta MICE [2]. Al ser todas las columnas valores reales y tomar valores continuos, se ha decidido que el método de imputación sea utilizar la media (*mean*) para todos los casos. Por ello, en vez de una lista le indicamos solamente dicho valor al parámetro `method` de la función (código 2).

```
> imputacion <- mice(data, method = "mean")
> data_imp <- complete(imputacion)
> head(data_imp)
```

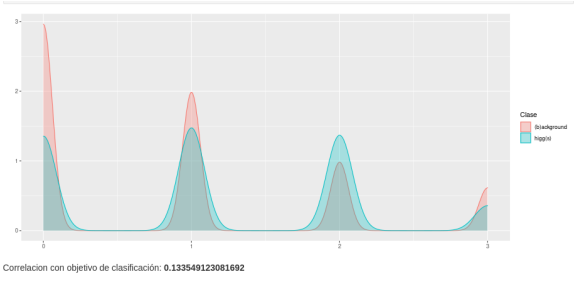
Código 2: Imputación de valores perdidos con MICE.



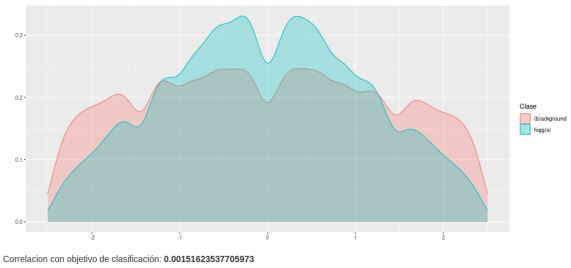
(a) der_mass_transverse



(b) der_met_phi



(c) pri_jet_num



(d) pri_lep_eta

Figura 4: Distribución en función de algunas variables.

3.3. Discretización de los datos

En este paso, los datos presentan una gran cantidad de valores diferentes. Por ello, se ha considerado oportuno llevar a cabo este proceso de discretización de las variables para reducir el esfuerzo computacional y también para intentar mejorar los resultados obtenidos mediante las técnicas de clasificación. Para este proceso, se ha llevado una investigación sobre cómo llevarlo a cabo en R. Las principales fuentes consultadas han sido [3] y [4] siendo la segunda ellas la que mejores resultados nos ha dado. Hacemos uso de la biblioteca `funModeling`, más concretamente de las funciones `discretize_get_bins` y `discretize_df`. El proceso llevado a cabo en el cuaderno es bastante sencillo:

1. Obtención de las variables: primero obtenemos los nombres de las variables que queremos discretizar, que en este caso son todas (tenemos que excluir la variable objetivo).
2. Ahora, mediante `discretize_get_bins` obtenemos los intervalos para cada columna. Le debemos indicar el número de *bins* que queremos. Se ha considerado que un valor adecuado es tener 12 intervalos diferentes. Durante el proceso excluye automáticamente la variable `PRI_jet_num` ya que detecta que presenta un número de valores menor al de los intervalos requeridos. También vemos cómo para algunos valores (`PRI_jet_leading_pt` por ejemplo) reduce también de manera automática el número de intervalos ya que tendrá incluido algún mecanismo de optimización. En la figura 5 podemos ver más claramente los intervalos creados.
3. Finalmente, con `discretize_df` y pasando con como argumento los intervalos creados anteriormente, llevamos a cabo el proceso de discretización de los datos.

De este modo, el estado acutal de los datos es como se muestra en la figura 6.

Vemos que en ahora cada valor se corresponde con un intervalo, por lo que son de tipo *factor*. Para facilitar el trabajo vamos a convertir dichos valores a numérico (orden `mutate_if(is.factor, as.numeric)`) para que tome valores 1,2,3, etc. Los pasos aparecen reflejados en el código 3.

```
> names <- data_imp %>%
          select(starts_with(c("DER", "PRI"))) %>%
          names()
```

	variable	cuts
1	DER_mass_MMC	75.571|87.154|95.666|103.672|111.666|119.959|121.859|130.607|142.694|169.758|Inf
2	DER_mass_transverse_met_lep	5.699|11.955|19.243|27.552|36.806|46.525|56.272|65.32|73.599|82.26|94.903|Inf
3	DER_mass_vis	44.551|53.613|59.389|64.301|68.99|73.753|78.894|84.89|92.26|102.315|124.442|Inf
4	DER_pt_h	1.853|4.297|14.069|23.967|30.249|38.468|48.705|61.666|79.17|104.888|149.867|Inf
5	DER_deltar_tau_lep	1.163|1.523|1.811|2.068|2.294|2.492|2.667|2.822|2.962|3.09|3.262|Inf
6	DER_pt_tot	1.064|1.873|2.842|4.136|6.356|12.316|20.49|24.178|27.592|33.367|46.917|Inf
7	DER_sum_pt	59.618|68.058|77.551|89.95|103.941|120.665|141.547|167.634|200.479|245.84|324.475|Inf
8	DER_pt_ratio_lep_tau	0.573|0.74|0.884|1.02|1.152|1.281|1.417|1.578|1.778|2.054|2.518|Inf
9	DER_met_phi_centrality	-1.41|-1.398|-1.37|-1.312|-1.067|-0.355|0.421|0.984|1.226|1.355|1.406|Inf
10	PRI_tau_pt	21.334|22.872|24.592|26.613|29.013|31.805|35.253|39.517|45.018|52.7|66.798|Inf
11	PRI_tau_eta	-1.789|-1.308|-0.924|-0.604|-0.307|-0.022|0.276|0.568|0.899|1.307|1.79|Inf
12	PRI_tau_phi	-2.622|-2.093|-1.574|-1.056|-0.546|-0.032|0.506|1.039|1.566|2.105|2.626|Inf
13	PRI_lep_pt	28.037|30.139|32.376|34.82|37.494|40.517|44.018|48.159|53.391|60.812|74.939|Inf
14	PRI_lep_eta	-1.864|-1.386|-1.013|-0.665|-0.356|-0.044|0.311|0.614|0.96|1.352|1.851|Inf
15	PRI_lep_phi	-2.619|-2.103|-1.521|-0.947|-0.434|0.087|0.598|1.095|1.619|2.123|2.628|Inf
16	PRI_met	11.165|16.652|21.399|25.88|30.239|34.803|39.643|45.193|51.896|61.764|81.956|Inf
17	PRI_met_phi	-2.623|-2.093|-1.574|-1.05|-0.541|-0.023|0.506|1.027|1.562|2.089|2.613|Inf
18	PRI_met_sumet	80.706|103.938|123.018|141.349|160.014|179.741|202.74|229.693|263.38|310.492|390.739|Inf
19	PRI_jet_leading_pt	37.265|46.422|57.44|71.773|84.823|97.432|137.782|Inf
20	PRI_jet_leading_eta	-2.081|-1.181|-0.432|-0.003|0.434|1.176|2.071|Inf
21	PRI_jet_leading_phi	-2.266|-1.414|-0.555|-0.012|0.504|1.386|2.269|Inf
22	PRI_jet_all_pt	30|31.377|40.513|56.555|81.05|109.936|149.033|214.828|Inf

Figura 5: Intervalos de la discretización.

	DER_mass_MMC	DER_mass_transverse_met_lep	DER_mass_vis	DER_pt_h	DER_deltar_tau_lep
1	138.4700	51.655	97.827	27.980	3.064
2	160.9370	68.768	103.235	48.146	3.473
3	121.8585	162.172	125.953	35.635	3.148
4	143.9050	81.417	80.943	0.414	3.310
5	175.8640	16.915	134.805	16.405	3.891
6	89.7440	13.550	59.149	116.344	1.362

(a) Datos imputados

	DER_mass_MMC	DER_mass_transverse_met_lep	DER_mass_vis	DER_pt_h	DER_deltar_tau_lep
1	[130.61,142.69)	[46.525,56.272)	[92.26,102.31)	[23.967, 30.249)	[2.962,3.090)
2	[142.69,169.76)	[65.320,73.599)	[102.31,124.44)	[38.468, 48.705)	[3.262, Inf]
3	[119.96,121.86)	[94.903, Inf]	[124.44, Inf]	[30.249, 38.468)	[3.090,3.262)
4	[142.69,169.76)	[73.599,82.260)	[78.89, 84.89)	[-Inf, 1.853)	[3.262, Inf]
5	[169.76, Inf]	[11.955,19.243)	[124.44, Inf]	[14.069, 23.967)	[3.262, Inf]
6	[87.15, 95.67)	[11.955,19.243)	[53.61, 59.39)	[104.888,149.867)	[1.163,1.523)

(b) Datos discretizados

Figura 6: Estados de algunas variables antes y después de la discretización.

```
> d_bins=discretize_get_bins(data=data_imp, input=names, n_bins=12)
> data_dis =discretize_df(data=data_imp,
    data_bins=d_bins,
    stringsAsFactors=T)
> data_dis <- data_dis %>%
  mutate_if(is.factor, as.numeric) %>%
  mutate_if(is.character, as.factor) #Convertir variable objetivo a
   factor
```

Código 3: Discretización de las variables.

3.4. Selección de instancias

Debido a que las clases están desbalanceadas, vamos a llevar a cabo un proceso de *downsampling* en el que igualaremos el número de instancias (eliminando de la clase mayoritaria) con el fin de que haya el mismo número de ambas clases de la variable objetivo. Para ello, primero transformamos la variable `Label` a factor (requerido por el proceso) y nos quedamos finalmente con 85 667 instancias de cada una de las clases, considerado un número adecuado para llevar a cabo el proceso. Por último, también se han creado dentro de este apartado los conjuntos de validación (21 416 instancias de cada clase) y entrenamiento (64 251 instancias de cada clase) de los modelos que usaremos posteriormente. El conjunto de entrenamiento supone el 75 % de los datos disponibles y, por tanto, el de validación 25 %.

```
> predictors <- select(data_dis, -Label)
> data_down <- downSample(x = predictors, y = data_dis$Label, yname = '
   Label')
> table(data_down$Label)
> trainIndex <- createDataPartition(data_down$Label, p = .75, list =
   FALSE)
```

```
> train <- data_down[ trainIndex, ]
> val   <- data_down[-trainIndex, ]
> table(train$Label)
> table(val$Label)
```

Código 4: Selección de instancias mediante *downsampling*.

3.5. Selección de variables

Para llevar a cabo este proceso tomaremos dos alternativas diferentes y posteriormente compararemos los resultados de las clasificaciones. Tenemos en cuenta que esta selección se hace sobre el conjunto de entrenamiento que hemos creado anteriormente y por lo tanto partimos de las mismas instancias en ambos casos.

3.5.1. Correlación

El primer método de selección de variables es mediante la detección de variables que presentan una alta correlación entre sí. Usaremos la función `rcorr`. Destacar que no vamos a tener en cuenta la variable objetivo en este proceso ya que queremos eliminar columnas que presenten una alta correlación entre sí y por ello, si la mantenemos podemos dejar por el camino variables importantes para predecir dicho valor (o incluso que no se quede con la variable objetivo y tengamos que introducirla posteriormente). La información obtenida de la matriz y de los clústeres la recogemos en las imágenes 7 y 8

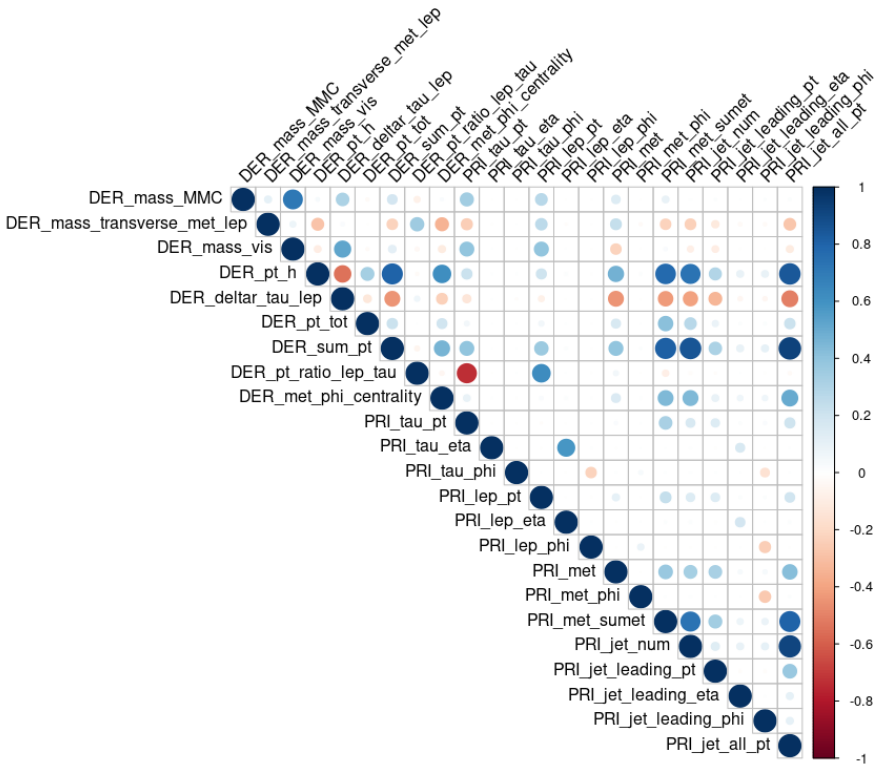


Figura 7: Matriz de correlaciones entre las variables.

Vemos claramente que hay variables que efectivamente presentan una alta correlación entre sí. Para quedarnos con las deseadas hacemos uso de `cutree`. Se ha decidido quedarse con 8 variables para llevar a cabo el proceso de clasificación que han sido:

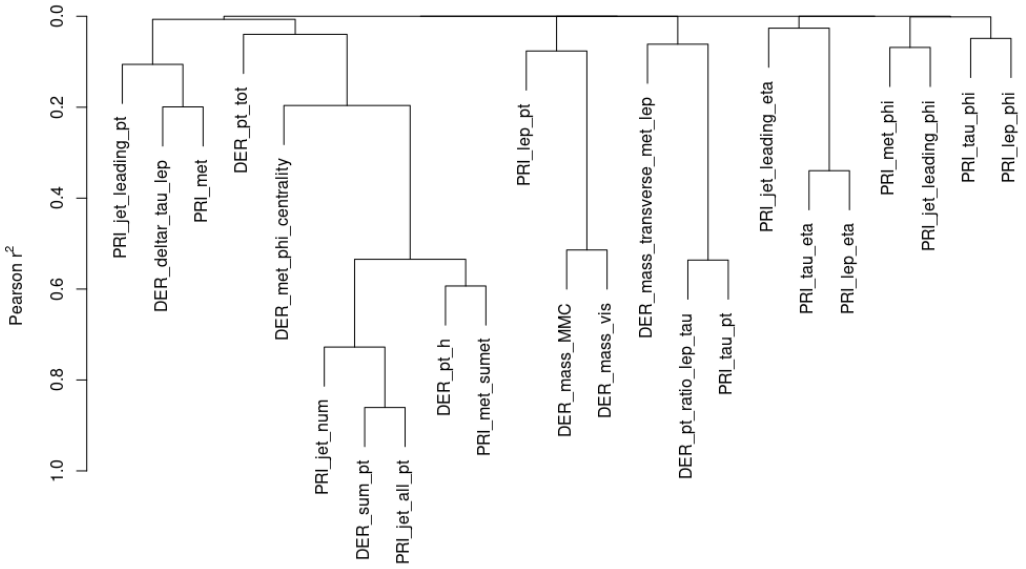


Figura 8: Clústeres obtenidos.

- DER_mass_MMC
- DER_mass_transverse_met_lep.
- PRI_tau_pt.
- PRI_met_sumet.
- DER_deltar_tau_lep.
- PRI_lep_eta.
- PRI_lep_phi.
- PRI_met_phi.
- PRI_jet_leading_eta.

En el código 5 mostramos el proceso utilizado en este caso.

```
> data_selec <- train %>%
  select(starts_with(c("DER", "PRI")))
> rcorr_result <- rcorr(as.matrix(data_selec))
> cor_matrix <- as.tibble(rcorr_result$r, rownames = "variable")
> corrplot(rcorr_result$r, type = "upper", order = "original", tl.col =
    "black", tl.srt = 45)
> v <- varclus(as.matrix(data_selec), similarity="pearson")
> plot(v)
> groups <- cutree(v$hclust, 8)
> not_correlated_vars <- enframe(groups) %>%
  group_by(value) %>%
  sample_n(1)

> train_sel <- train %>%
  select(one_of(not_correlated_vars$name))
```

```
> train_sel$Label <- as.factor(train$Label) # La metemos de nuevo
> head(train_sel)
> val_sel <- val %>%
  select(one_of(not_correlated_vars$name))
Z val_sel$Label <- as.factor(val$Label) # La metemos de nuevo
> head(val_sel)
```

Código 5: Selección de variables mediante correlación.

3.5.2. Análisis de componentes principales (PCA)

Como segundo método de selección de variables, lo que vamos a hacer es usar un análisis de componentes principales o PCA y cuyo proceso en `R` se recoge en el código 6. Usaremos para tal fin la función `procomp` que ya viene incorporada en el lenguaje. El objetivo de llevar a cabo este análisis además del anterior es el poder comparar si hay diferencias significativas mediante un método de reducción de los datos o no. El resultado del mismo lo podemos ver en la figura 9.

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11    PC12
Standard deviation     2.3774 1.5475 1.48428 1.29336 1.18708 1.1470 1.09448 0.99827 0.96283 0.91731 0.90265 0.87198
Proportion of Variance 0.2457 0.1041 0.09579 0.07273 0.06127 0.0572 0.05208 0.04333 0.04031 0.03658 0.03543 0.03306
Cumulative Proportion  0.2457 0.3499 0.44565 0.51838 0.57965 0.6369 0.68893 0.73226 0.77257 0.80915 0.84458 0.87764
                          PC13    PC14    PC15    PC16    PC17    PC18    PC19    PC20    PC21    PC22    PC23
Standard deviation     0.77339 0.74259 0.64559 0.61646 0.49994 0.46115 0.42217 0.32483 0.24011 0.20681 0.14564
Proportion of Variance 0.02601 0.02398 0.01812 0.01652 0.01087 0.00925 0.00775 0.00459 0.00251 0.00186 0.00092
Cumulative Proportion  0.90364 0.92762 0.94574 0.96226 0.97313 0.98238 0.99012 0.99471 0.99722 0.99908 1.00000
```

Figura 9: Análisis de componentes principales.

No olvidamos que en la sección de análisis exploratorio de datos mediante `DataExplorer` también obtuvimos un PCA. Sin embargo, ahora hemos llevado a cabo previamente un preprocesamiento de los datos por lo que los resultados son diferentes a los obtenidos en aquel momento. Centrándonos en el los porcentajes actuales, vemos que PC1 explica el 24.57 % de la varianza del conjunto de datos y aproximadamente el 95 % lo obtenemos con las 15 primeras componentes principales. Observamos que son mejores que los iniciales lo que se puede deber al preprocesamiento que hemos realizado hasta el momento. Podemos visualizar gráficamente en la figura 10 (teniendo en cuenta el alto número de instancias) la proyección usando la variable `PRI_jet_num` que es al que tiene un menos número de valores (figura 10).

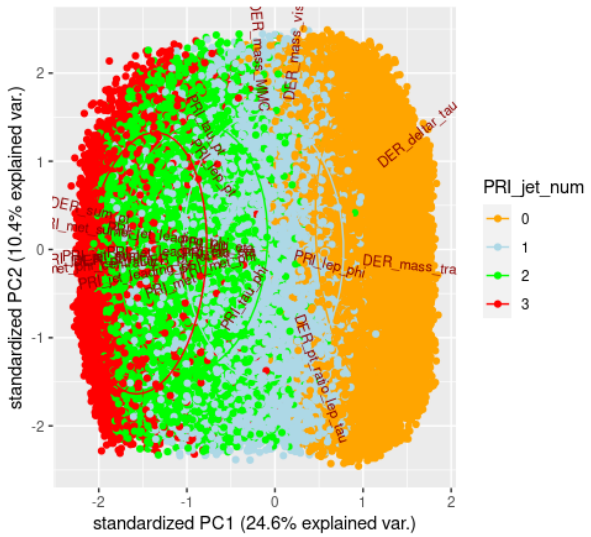


Figura 10: Proyección PCA.

A raíz de los resultados, vamos a quedarnos con las primeras 10 componentes principales que como hemos indicado explican alrededor del 80 % de la varianza.

```
> pca <- prcomp(train[,1:ncol(train)-1], scale=TRUE) # Quitamos la objetivo
> summary(pca)
> library(ggbiplot)
> ggbiplot(pca, groups = as.factor(train$PRI_jet_num), ellipse = TRUE) +
scale_colour_manual(name="PRI_jet_num", labels=c(0,1,2,3),  values= c("orange", "lightblue", "green", "red"))
> train_samples_proj <- predict(pca, train)
> train_pca <- as_tibble(train_samples_proj[,1:10]) %>%
  mutate(Label = train$Label)

> val_samples_proj <- predict(pca, val)
> val_pca <- as_tibble(val_samples_proj[,1:10]) %>%
  mutate(Label = val$Label)
```

Código 6: Selección de variables mediante PCA.

4. Clasificación

En este momento, ya tenemos nuestros conjuntos de entrenamiento y de validación. Por un lado uno creado mediante selección de variables mediante correlación y otro mediante PCA. Veamos cómo se comportan al aplicar técnicas de clasificación.

4.1. Árboles de decisión

La primera técnica que vamos a utilizar es árboles de decisión, usada con `caret` con el método `rpart`. Para obtener mejores resultados también se va a llevar a cabo una validación cruzada con 10 *folds*. Para el entrenamiento también se le ha pasado una rejilla de parámetros tal y como mostramos en el código 7.

4.1.1. Correlación

```
> rpartCtrl <- trainControl(verboseIter = F,
                        classProbs = TRUE,
                        method = "repeatedcv",
                        number = 10,
                        repeats = 1,
                        summaryFunction = twoClassSummary)
> rpartParametersGrid <- expand.grid(.cp = c(0.001, 0.01, 0.1, 0.5))
> rpartModel_sel <- train(Label ~ ., data = train_sel, method = "rpart", metric = "ROC", trControl = rpartCtrl, tuneGrid = rpartParametersGrid)
> rpart.plot(rpartModel_sel$finalModel)
> rpartModel_sel_roc <- my_roc(val_sel, predict(rpartModel_sel, val_sel, type = "prob"), "Label", "s")
> confusionMatrix(predict(rpartModel_sel, val_sel, type = "raw"), val_sel[["Label"]], positive = "s")
```

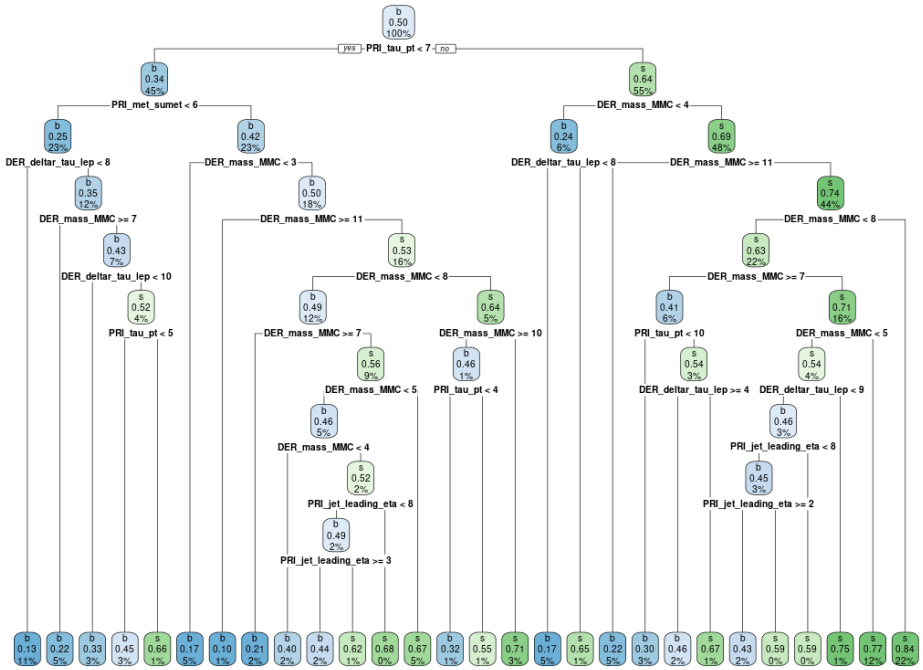


Figura 11: Árbol de decisión obtenido para el conjunto de datos con variables seleccionadas mediante correlación.

Código 7: Código para entrenar un árbol de decisión a partir de las variables seleccionadas por correlación.

Destacamos que en el código, la función `my_roc` es la vista en clase durante la sesión 7 del archivo sobre "Modelos avanzados de clasificación y ensembles con German Credit". Los resultados han sido de un área bajo la curva de 0.8201. Gráficamente se mostrarán dichas curvas en la sección 5 donde compararemos los resultados obtenidos en cada uno de los casos.

Mediante la función `rpart.plot` podemos ver gráficamente el árbol obtenido durante el proceso (figura 11). En la tabla 1 mostramos la matriz de confusión que obtiene un *accuracy* de 0.7661.

clase/pred	b	s
b	16 833	5 437
s	4 583	15 979

Tabla 1: Matriz de confusión obtenida para el conjunto de datos con variables seleccionadas mediante correlación (`rpart`).

4.1.2. PCA

Del mismo modo, llevamos a cabo el proceso mediante los conjuntos creados por selección de características mediante PCA (código 8 y árbol obtenido figura 12). En este caso el área bajo la curva es de 0.7903 y la exactitud de 0.7467 (tabla 2).

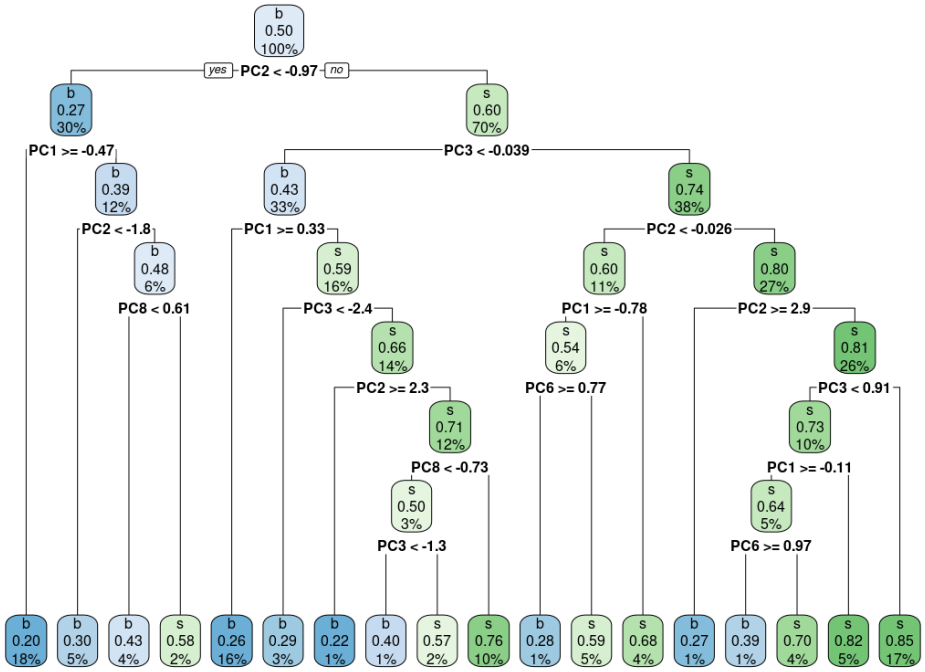


Figura 12: Árbol de decisión obtenido para el conjunto de datos con variables seleccionadas mediante PCA.

clase/pred	b	s
b	16 333	5 766
s	5 083	15 650

Tabla 2: Matriz de confusión obtenida para el conjunto de datos con variables seleccionadas mediante PCA (`rpart`).

```
> rpartModel_pca <- train(Label ~ ., data = train_pca, method = "rpart"
    , metric = "ROC", trControl = rpartCtrl, tuneGrid =
    rpartParametersGrid)
> rpart.plot(rpartModel_sel$finalModel)
> rpartModel_pca_roc <- my_roc(val_pca, predict(rpartModel_pca, val_pca
    , type = "prob"), "Label", "s")
> confusionMatrix(predict(rpartModel_sel, val_sel, type = "raw"), val_
    sel[["Label"]], positive = "s")
```

Código 8: Código para entrenar un árbol de decisión a partir de las variables seleccionadas por PCA.

4.2. *Boosted Logistic Regression*

Como se ha indicado en la introducción se pretendía utilizar una técnica nueva para la práctica que no se hubiera utilizado antes. Entre las pruebas realizadas se probaron algunas mencionadas en clase como es el caso de *eXtreme Gradient Boosting* o *AdaBoost Classification Trees*. También se buscó más información sobre el conjunto de técnicas disponibles en `caret` [7]. Debido a los recursos técnicos disponibles, el método que finalmente mejores resultados nos ha proporcionado es el finalmente usado,

Boosted Logistic Regression[9]. Este método se basa en `AdaBoost` y utiliza mecanismos lineales en vez de exponenciales reduciendo de este modo la vulnerabilidad frente a datos ruidosos [8]. Por ello, al utilizar *boosting* vemos que estamos utilizando técnicas de *ensembles* que hemos visto en clase. Mostramos ahora los resultados obtenidos en cada uno de nuestros conjuntos.

4.2.1. Correlación

```
> regCtrl <- trainControl(verboseIter = T,
                          classProbs = TRUE,
                          summaryFunction = twoClassSummary)
> regModel_sel <- train(Label ~ ., data = train_sel, method = "
    LogitBoost", metric = "ROC", trControl = regCtrl)
> rregModel_sel_roc <- my_roc(val_sel, predict(regModel_sel, val_sel,
    type = "prob"), "Label", "s")
> confusionMatrix(predict(regModel_sel, val_sel, type = "raw"), val_sel
    [["Label"]], positive = "s")
```

Código 9: Código para entrenar una regresión logística a partir de las variables seleccionadas por correlación.

Los resultados han sido de un área bajo la curva de 0.7485. Por su parte, en la tabla 3 recogemos la matriz de confusión y vemos que el *accuracy* es de 0.6992.

clase/pred	b	s
b	14 345	5 811
s	7 071	15 605

Tabla 3: Matriz de confusión obtenida para el conjunto de datos con variables seleccionadas mediante correlación (`logitBoost`).

4.2.2. PCA

Finalmente, usamos los conjuntos creados por selección de características mediante PCA (código 10) En este caso el área bajo la curva es de 0.7303 mientras que tenemos un *accuracy* de 0.6929 (tabla 4).

```
> regModel_pca <- train(Label ~ ., data = train_pca, method = "
    LogitBoost", metric = "ROC", trControl = regCtrl)
> rregModel_pca_roc <- my_roc(val_pca, predict(regModel_pca, val_pca,
    type = "prob"), "Label", "s")
> confusionMatrix(predict(regModel_pca, val_pca, type = "raw"), val_pca
    [["Label"]], positive = "s")
```

Código 10: Código para entrenar una regresión logística a partir de las variables seleccionadas por PCA.

clase/pred	b	s
b	15 372	7 109
s	6 044	14 307

Tabla 4: Matriz de confusión obtenida para el conjunto de datos con variables seleccionadas mediante PCA (`LogitBoost`).

5. Discusión de resultados

En esta sección vamos a discutir los resultados obtenidos en la sección 4. En primer lugar vamos a analizar los resultados en función de curvas ROC. En la figura 13 vemos gráficamente dichas curvas.

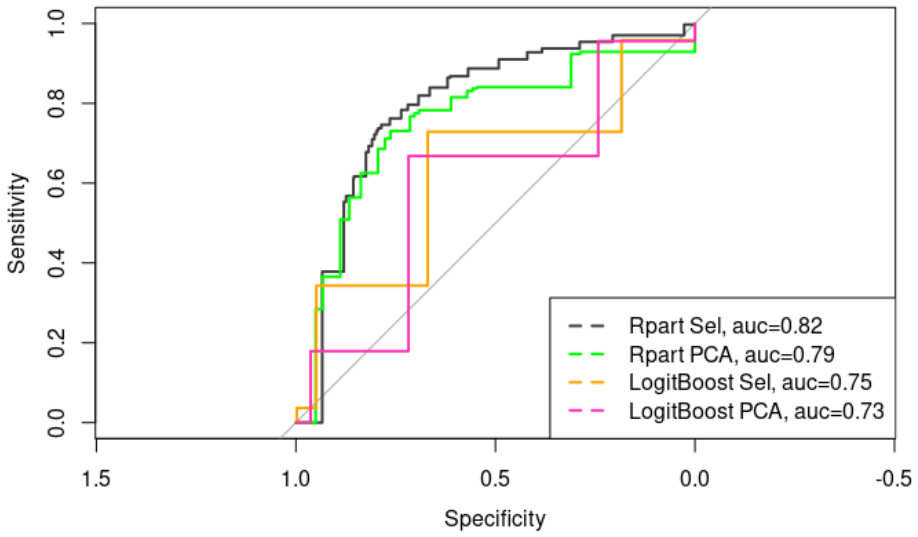


Figura 13: Comparación de las curvas ROC obtenidas.

Se observa cómo los mejores resultados se han obtenido mediante el árbol de decisión que utiliza selección de variables mediante correlación, donde se ha obtenido un área bajo la curva de 0.82. También destacamos que los resultados obtenidos para la regresión logística son peores que los obtenidos mediante árboles de decisión. Además, en ambos casos siempre se ha obtenido una clasificación peor usando PCA a pesar de tener un mayor número de variables (10 frente a 8). Si nos centramos ahora en el *accuracy*, obtenemos la misma clasificación que el caso anterior: `rpart` con correlación (0.7661), `rpart` con PCA (0.7467), `LogitBost` con correlación (0.6992) y `LogitBoost` con PCA (0.6929). Si nos fijamos en las matrices de confusión vemos que en la mayoría de los casos, los modelos predicen un mayor número de valores como ruido a pesar de estar el conjunto de entrenamiento balanceado ya que se rebajaron las instancias de la clase mayoritaria mediante *downsampling*.

6. Conclusiones

A modo de conclusiones, podemos poner de manifiesto la importancia que tiene el preprocesamiento de datos en tareas en las que tenemos que tratar con una gran cantidad de datos. Por ejemplo, hemos visto que hemos conseguido explicar un mayor porcentaje de la varianza mediante PCA tras algunas técnicas de preprocesamiento que con los datos iniciales. A pesar de ello, en esta ocasión los resultados finales de los procesos de clasificación no han sido demasiados buenos. Esto pone de manifiesto que puede que haya que aplicar otras técnicas (complementando o sustituyendo a las actuales) para mejorar los resultados. Entre ellas encontramos la normalización de los datos, la reducción de ruido o la detección de *outliers*. Otra posibilidad sería utilizar otros mecanismos de clasificación que nos pudieran aportar mejores resultados finales.

Bibliografía

1. Apuntes de la asignatura y cuadernos en R proporcionados, Juan Gómez Romero.
2. Biblioteca MICE, https://www.rdocumentation.org/packages/mice/versions/3.13.0/topics/mice. Última visita: 13/04/2021.
3. Discretización de datos, https://rpubs.com/IranNash/discretizacion. Última visita: 13/04/2021.
4. Data discretization made easy with funModeling, https://blog.datascienceheroes.com/data-discretization-made-easy-with-funmodeling/. Última visita: 13/04/2021.
5. rcorr: Matrix of Correlations and P-values, https://www.rdocumentation.org/packages/Hmisc/versions/4.5-0/topics/rcorr. Última visita: 13/04/2021.
6. varclus: Variable Clustering, https://www.rdocumentation.org/packages/Hmisc/versions/4.5-0/topics/varclus. Última visita: 13/04/2021.
7. caret: A List of Available Models in train, https://www.datatechnotes.com/2018/07/logisboost-classification-sample-in-r.html. Última visita: 15/04/2021.
8. Classification Example with LogitBoost Method in R, https://www.datatechnotes.com/2018/07/logisboost-classification-sample-in-r.html. Última visita: 15/04/2021.
9. Friedman, J., Hastie, T., & Tibshirani, R. (2000). *Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors).* Annals of statistics, 28(2), 337-407.