PEC4: Predicción de dolencias cardiacas a partir de un electrocardiograma

Demetrio Muñoz Alvarez

2024-01-28

Contents

Introducción

```r
dataset <- read.csv("ECGCvdata.csv") # Cargamos nuestro conjunto de datos.
```

En esta última actividad evaluable, se realizará un análisis predictivo del tipo de dolencia cardíaca utilizando los datos de los pacientes recopilados en el conjunto de datos ECGCvdata.csv. Para analizar estas dolencias, se implementarán distintos algoritmos: **k-Nearest Neighbour, Naive Bayes, Artificial Neural Network, Support Vector Machine, Árbol de Decisión y Random Forest**.

Este informe se dividirá en tres partes. La primera parte mostrará un análisis descriptivo del conjunto de datos. Además, se tratarán los datos en busca de valores faltantes y se procederá a su eliminación. La segunda parte consistirá en dividir el conjunto de datos en datos de entrenamiento y prueba para luego entrenar los distintos algoritmos. Por último, se compararán los modelos y se concluirá cuál o cuáles son los mejores para predecir las dolencias cardíacas.

Exploración y tratamiento de los datos

El conjunto de datos ECGCvdata.csv esta formado por 1200 observaciones y 56 variables.

En un primer contacto, mostramos las primeras y las últimas seis entradas. En este caso, se elimina la columna **"RECORD"** del conjunto de datos, ya que no aporta información útil para nuestros posteriores análisis. Por último, mostramos el resumen de las diez primeras entradas.

```r
head(dataset) # Mostramos las primeras entradas del conjunto de datos.
```

```
  RECORD hbpermin       Pseg      PQseg     QRSseg      QRseg     QTseg      RSseg
1      1 74.92567 0.07650794 0.10888889 0.08825397 0.04357143 0.1930159 0.04468254
2      2 68.50347 0.07248264 0.09618056 0.09392361 0.04626736 0.1934896 0.04765625
3      3 83.48860 0.07115385 0.08660969 0.03952991 0.01858974 0.1324786 0.02094017
4      4 68.50347 0.08281250 0.10815972 0.09036458 0.04522569 0.1888021 0.04513889
5      5 82.08000 0.07076023 0.10263158 0.10102339 0.04941520 0.1937135 0.05160819
6      6 66.96000 0.07025090 0.06191756 0.09094982 0.04596774 0.1912186 0.04498208
       STseg       Tseg     PTseg    ECGseg  QRtoQSdur  RStoQSdur   RRmean   PPmean
1 0.10476190 0.13047619 0.3019048 0.4261111 0.001371360 0.001406417 291.7941 291.7353
2 0.09956597 0.08914931 0.2896701 0.3666667 0.001368380 0.001409398 318.3871 318.3226
3 0.09294872 0.09444444 0.2190883 0.2932336         NaN         NaN 259.7632 259.5263
4 0.09843750 0.08828125 0.2969618 0.3731771 0.001390204 0.001387574 312.8387 312.8387
5 0.09269006 0.08596491 0.2963450 0.3750731 0.001358812 0.001418966 260.7838 260.8649
6 0.10026882 0.09713262 0.2531362 0.3302867 0.001403823 0.001373955 313.9333 313.9333
     PQdis  PonQdis    PRdis  PonRdis    PSdis  PonSdis     PTdis   PonTdis  PToffdis
1 39.15115 55.14889 54.82707 70.82739 70.91373 86.91274 108.64707 124.64708 137.91182
2 34.58844 46.48701 51.26239 63.16713 68.42260 80.32397 104.22584 116.12908 119.93566
3 31.26623 42.53782 38.13127 49.40603 45.86732 57.14121  79.23823  90.52672  94.76524
4 38.94167 54.77718 55.23159 71.07051 71.48675 87.32412 106.90324 122.74197 118.45169
5 37.00513 49.97480 54.78642 67.76063 73.38103 86.35245 106.75679 119.72978 122.18936
6 22.27948 37.63916 38.84048 54.20710 55.03741 70.40214  91.13335 106.50005 103.53346
      QRdis    QSdis    QTdis QToffdis     RSdis    RTdis RToffdis    STdis SToffdis
1 15.721401 31.76473 69.50192 98.76570 16.129107 53.82771 83.09160 37.73834 67.00123
2 16.736067 33.83879 69.64818 85.35647 17.212920 52.97306 68.68296 35.81116 51.51818
3  6.896067 14.60644 48.01302 63.53184  7.751319 41.14278 56.66424 33.40805 48.92607
4 16.358555 32.54843 67.97091 79.51807 16.321751 51.68419 63.23273 35.42451 46.97045
5 17.821022 36.37839 69.75882 85.19019 18.630945 51.97661 67.40945 33.38282 48.81263
6 16.634063 32.76678 68.87019 81.26881 16.261555 52.30608 64.70632 36.10526 48.50267
  PonToffdis PonPQang    PQRang    QRSang    RSTang STToffang RRTot NNTot     SDRR
1   153.9118 1.245455 -5.127081  8.356354 -4.774444 0.8704898    35    34 17.13232
```

```
2   131.8387 2.151593 -6.003337  9.217430 -5.355692 1.3703198    32    31 10.61246
3   106.0529      NaN       NaN       NaN       NaN 2.9995653    39    38 84.51298
4   134.2903      NaN -6.253180 10.293861 -6.035961 1.4626173    32    31 11.41617
5   135.1622 1.791385 -4.646931  7.254628 -4.489193 1.3823574    38    37 35.77115
6   118.9000      NaN -7.104194 10.136861 -5.957746 1.4704725    31    30 39.66857
      IBIM    IBISD       SDSD    RMSSD  QRSarea  QRSperi     PQslope    QRslope
1 291.7941 17.38996  27.158481 292.2966 18.45762 63.61524 -0.01436363 0.07526997
2 318.3871 10.78789   9.665517 318.5639 23.04323 67.78777 -0.02120743 0.08377260
3 259.7632 85.64743 111.816694 273.1654 10.75635 29.25383 -0.04254194        NaN
4 312.8387 11.60487  11.193252 313.0469 23.84509 65.22874 -0.01780602 0.09159094
5 260.7838 36.26456  60.245998 263.2257 20.94279 72.83035 -0.01665019 0.06454671
6 313.9333 40.34672  13.811712 316.4297 23.78713 65.66240 -0.03402993 0.09022200
      RSslope    STslope NN50     pNN50 ECG_signal
1 -0.07084590 0.01260642    2  5.882353        ARR
2 -0.07745817 0.01617517    1  3.225806        ARR
3         NaN 0.02713144   16 42.105263        ARR
4 -0.08855894 0.01702174    2  6.451613        ARR
5 -0.06224594 0.01619113    2  5.405405        ARR
6 -0.08716756 0.01703774    3 10.000000        ARR
```

```
tail(dataset) # Mostramos las últimas entradas del conjunto de datos.
```

```
     RECORD hbpermin       Pseg      PQseg     QRSseg       QRseg      QTseg        RSseg
1195   1195 72.19200 0.05125762 0.07212271 0.03229802 0.01648247 0.1164253 0.015815549
1196   1196 91.39200 0.05135290 0.06250000 0.01915015 0.01005145 0.1180450 0.009098704
1197   1197 61.44000 0.05735518 0.06154726 0.06126143 0.03134527 0.1340987 0.029916159
1198   1198 93.32271 0.05892721 0.10913681 0.08093559 0.04030107 0.1681117 0.040634527
1199   1199 63.74400 0.06149962 0.10127668 0.08179306 0.04115854 0.1459127 0.040634527
1200   1200 70.65600 0.06307165 0.10542111 0.08303163 0.04153963 0.1714939 0.041491997
          STseg       Tseg     PTseg    ECGseg  QRtoQSdur  RStoQSdur    RRmean
1195 0.08412729 0.07993521 0.1885480 0.2367092         NaN         NaN 105.46237
1196 0.09889482 0.08655678 0.1805450 0.2213700         NaN         NaN  84.11017
1197 0.07283727 0.09098704 0.1956460 0.2788681 0.003986151 0.003826349 124.55696
1198 0.08717607 0.08808117 0.2772485 0.3437024 0.003871321 0.003941179  81.98347
1199 0.06411966 0.08117378 0.2471894 0.3308403 0.003940478 0.003872022 120.07317
1200 0.08846227 0.08165015 0.2769150 0.3415587 0.003894670 0.003917830 107.96703
        PPmean     PQdis   PonQdis     PRdis   PonRdis     PSdis   PonSdis    PTdis
1195 105.46237  7.068159  8.723866  7.141192  8.798433  7.141192  8.798433 19.00870
1196  84.11017  7.020798  9.316893  7.128290  9.426490  7.128290  9.426490 20.58185
1197 124.55696  6.686759 11.824368 12.362308 17.528025 17.815321 22.970875 24.58235
1198  81.98347 15.098145 18.816598 20.547847 24.279059 26.110730 29.832392 37.52916
1199 120.07317 12.920299 17.082490 18.615402 22.787275 24.206564 28.372832 30.59757
1200 107.95604 14.062848 18.460068 19.817331 24.225889 25.591232 29.993793 38.09897
      PonTdis PToffdis      QRdis       QSdis    QTdis QToffdis    RSdis    RTdis
1195 20.74258 22.64073 0.07528038  0.07528038 12.10469 15.73798 0.000000 12.02950
1196 22.95773 22.86151 0.11017363  0.11017363 13.71429 15.98885 0.000000 13.60428
1197 29.74702 35.16506 5.87890540 11.15207871 17.94971 28.52430 5.661904 12.26472
1198 41.25626 42.89318 5.61812344 11.01699865 22.45347 27.81370 5.745388 17.01043
1199 34.76842 39.87831 5.88881576 11.29288552 17.70731 26.98090 5.811831 12.02916
1200 42.50555 42.27506 5.95207235 11.53861109 24.06394 28.23636 5.959027 18.31619
     RToffdis     STdis STtoffdis PonToffdis  PonPQang    PQRang   QRSang    RSTang
1195 15.66274 12.029502 15.66274   24.37685 25.243733       NaN      NaN       NaN
1196 15.87880 13.604285 15.87880   25.23743 21.328827       NaN      NaN       NaN
1197 22.84051  6.826239 17.37778   40.32913  8.188221 -20.72186 29.90325 -21.94370
1198 22.37344 11.451169 16.80291   46.61986  5.403345 -16.46365 28.39688 -18.20181
```

```
1199 21.30210  6.443333 15.69440   44.04879  4.611690 -17.78978 30.43289 -22.12574
1200 22.49220 12.530919 16.69995   46.68133  4.751589 -17.65742 28.78335 -16.84394
     STToffang RRTot NNTot      SDRR     IBIM     IBISD     SDSD     RMSSD  QRSarea
1195       NaN    94   163 10.117120 89.27607 14.462542 3.183632 105.94653 0.000000
1196       NaN   119   163  3.458676 79.36810  5.009289 2.188401  84.18125 0.000000
1197  7.121384    80   163  4.788559 98.46012 18.528265 3.254416 124.64898 8.243245
1198  5.037948   122   163  8.546406 78.07975  8.120195 3.141910  82.42773 7.623618
1199  7.221305    83   163  6.140192 99.20859 15.473730 5.647497 120.23006 8.614011
1200  4.070685    92   163  7.910836 94.27607 11.850967 5.129207 108.25646 8.494619
        QRSperi     PQslope   QRslope    RSslope    STslope NN50     pNN50 ECG_signal
1195  0.1505608 -0.20409542       NaN        NaN 0.06997283    2 1.2269939        NSR
1196  0.2203473 -0.19547732       NaN        NaN 0.06419579    0 0.0000000        NSR
1197 22.6928876 -0.11169809 0.2559636 -0.2785314 0.11201242    0 0.0000000        NSR
1198 22.3805101 -0.04527077 0.2471451 -0.2593030 0.06431761    1 0.6134969        NSR
1199 22.9935326 -0.05262603 0.2639214 -0.2804169 0.11342241    9 5.5214724        NSR
1200 23.4497109 -0.05185913 0.2622807 -0.2511948 0.04802289    7 4.2944785        NSR
```

```r
dataset <- dataset[,-1] # Eliminamos la columna "RECORD" ya que no aporta informacion útil a nuestros d
summary(dataset[,1:10]) # Mostramos el un resumen de las 10 primeras entradas.
```

```
    hbpermin           Pseg              PQseg              QRSseg
 Min.   : 12.86   Min.   :0.02156   Min.   :0.04453   Min.   :0.00000
 1st Qu.: 67.56   1st Qu.:0.05394   1st Qu.:0.06034   1st Qu.:0.01510
 Median : 79.87   Median :0.06064   Median :0.07518   Median :0.04469
 Mean   : 81.89   Mean   :0.06090   Mean   :0.07810   Mean   :0.04824
 3rd Qu.: 96.00   3rd Qu.:0.06685   3rd Qu.:0.09519   3rd Qu.:0.08303
 Max.   :160.50   Max.   :0.09532   Max.   :0.14558   Max.   :0.12016
     QRseg              QTseg              RSseg              STseg
 Min.   :0.000000   Min.   :0.09876   Min.   :0.000000   Min.   :0.05234
 1st Qu.:0.007804   1st Qu.:0.11602   1st Qu.:0.007143   1st Qu.:0.08942
 Median :0.022676   Median :0.13702   Median :0.021965   Median :0.09782
 Mean   :0.024437   Mean   :0.14154   Mean   :0.023800   Mean   :0.09330
 3rd Qu.:0.041915   3rd Qu.:0.16631   3rd Qu.:0.041097   3rd Qu.:0.10120
 Max.   :0.065278   Max.   :0.21111   Max.   :0.058333   Max.   :0.13844
      Tseg              PTseg
 Min.   :0.03494   Min.   :0.1500
 1st Qu.:0.09005   1st Qu.:0.1773
 Median :0.09921   Median :0.2152
 Mean   :0.10271   Mean   :0.2196
 3rd Qu.:0.11241   3rd Qu.:0.2603
 Max.   :0.19841   Max.   :0.3473
```

La siguiente tabla muestra el número de casos para cada tipo de dolencia:

ECG_signal	Frecuencia
AFF	300
ARR	300
CHF	300
NSR	300

Si analizamos la estructura de los datos, podemos observar que casi todas las variables son de tipo numérico, excepto la variable que muestra las clases, que es de tipo categórica. En algunas de estas variables, observamos valores faltantes que trataremos en el siguiente paso.

```r
str(dataset) # Estructura de las variables.
```

```
'data.frame':	1200 obs. of  55 variables:
 $ hbpermin  : num  74.9 68.5 83.5 68.5 82.1 ...
 $ Pseg      : num  0.0765 0.0725 0.0712 0.0828 0.0708 ...
 $ PQseg     : num  0.1089 0.0962 0.0866 0.1082 0.1026 ...
 $ QRSseg    : num  0.0883 0.0939 0.0395 0.0904 0.101 ...
 $ QRseg     : num  0.0436 0.0463 0.0186 0.0452 0.0494 ...
 $ QTseg     : num  0.193 0.193 0.132 0.189 0.194 ...
 $ RSseg     : num  0.0447 0.0477 0.0209 0.0451 0.0516 ...
 $ STseg     : num  0.1048 0.0996 0.0929 0.0984 0.0927 ...
 $ Tseg      : num  0.1305 0.0891 0.0944 0.0883 0.086 ...
 $ PTseg     : num  0.302 0.29 0.219 0.297 0.296 ...
 $ ECGseg    : num  0.426 0.367 0.293 0.373 0.375 ...
 $ QRtoQSdur : num  0.00137 0.00137 NaN 0.00139 0.00136 ...
 $ RStoQSdur : num  0.00141 0.00141 NaN 0.00139 0.00142 ...
 $ RRmean    : num  292 318 260 313 261 ...
 $ PPmean    : num  292 318 260 313 261 ...
 $ PQdis     : num  39.2 34.6 31.3 38.9 37 ...
 $ PonQdis   : num  55.1 46.5 42.5 54.8 50 ...
 $ PRdis     : num  54.8 51.3 38.1 55.2 54.8 ...
 $ PonRdis   : num  70.8 63.2 49.4 71.1 67.8 ...
 $ PSdis     : num  70.9 68.4 45.9 71.5 73.4 ...
 $ PonSdis   : num  86.9 80.3 57.1 87.3 86.4 ...
 $ PTdis     : num  108.6 104.2 79.2 106.9 106.8 ...
 $ PonTdis   : num  124.6 116.1 90.5 122.7 119.7 ...
 $ PToffdis  : num  137.9 119.9 94.8 118.5 122.2 ...
 $ QRdis     : num  15.7 16.7 6.9 16.4 17.8 ...
 $ QSdis     : num  31.8 33.8 14.6 32.5 36.4 ...
 $ QTdis     : num  69.5 69.6 48 68 69.8 ...
 $ QToffdis  : num  98.8 85.4 63.5 79.5 85.2 ...
 $ RSdis     : num  16.13 17.21 7.75 16.32 18.63 ...
 $ RTdis     : num  53.8 53 41.1 51.7 52 ...
 $ RToffdis  : num  83.1 68.7 56.7 63.2 67.4 ...
 $ STdis     : num  37.7 35.8 33.4 35.4 33.4 ...
 $ SToffdis  : num  67 51.5 48.9 47 48.8 ...
 $ PonToffdis: num  154 132 106 134 135 ...
 $ PonPQang  : num  1.25 2.15 NaN NaN 1.79 ...
 $ PQRang    : num  -5.13 -6 NaN -6.25 -4.65 ...
 $ QRSang    : num  8.36 9.22 NaN 10.29 7.25 ...
 $ RSTang    : num  -4.77 -5.36 NaN -6.04 -4.49 ...
 $ STToffang : num  0.87 1.37 3 1.46 1.38 ...
 $ RRTot     : int  35 32 39 32 38 31 32 28 44 32 ...
 $ NNTot     : int  34 31 38 31 37 30 31 27 43 31 ...
 $ SDRR      : num  17.1 10.6 84.5 11.4 35.8 ...
 $ IBIM      : num  292 318 260 313 261 ...
 $ IBISD     : num  17.4 10.8 85.6 11.6 36.3 ...
 $ SDSD      : num  27.16 9.67 111.82 11.19 60.25 ...
 $ RMSSD     : num  292 319 273 313 263 ...
 $ QRSarea   : num  18.5 23 10.8 23.8 20.9 ...
 $ QRSperi   : num  63.6 67.8 29.3 65.2 72.8 ...
 $ PQslope   : num  -0.0144 -0.0212 -0.0425 -0.0178 -0.0167 ...
 $ QRslope   : num  0.0753 0.0838 NaN 0.0916 0.0645 ...
 $ RSslope   : num  -0.0708 -0.0775 NaN -0.0886 -0.0622 ...
```

```
 $ STslope   : num  0.0126 0.0162 0.0271 0.017 0.0162 ...
 $ NN50      : int  2 1 16 2 2 3 2 5 0 4 ...
 $ pNN50     : num  5.88 3.23 42.11 6.45 5.41 ...
 $ ECG_signal: chr  "ARR" "ARR" "ARR" "ARR" ...
```

Los valores faltantes se muestran en la siguiente tabla:

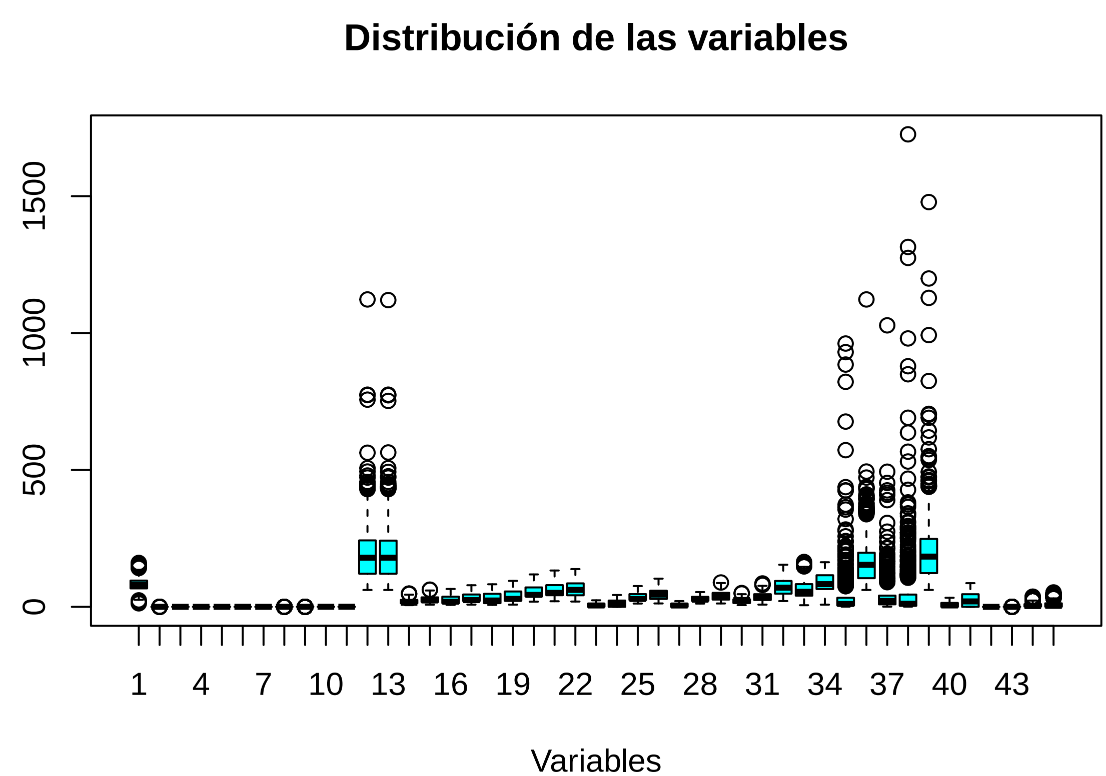
```r
valores_nan <- table(is.na(dataset)) # Contamos y mostramos los valores "NaN"
valores_nan
```

```
FALSE  TRUE
59636  6364
```

```r
dataset_nan <- dataset[, colSums(is.na(dataset)) == 0]
```

Como indica el enunciado de la actividad, vamos a tratar los valores faltantes eliminando aquellas variables que presenten algún valor 'NaN'. Una vez eliminados los datos faltantes nuestro conjunto de datos muestra 1200 observaciones y 46 variables.

En la siguiente gráfica, observamos la distribución de las variables. Las variables no se distribuyen de forma uniforme y se observan valores atípicos.



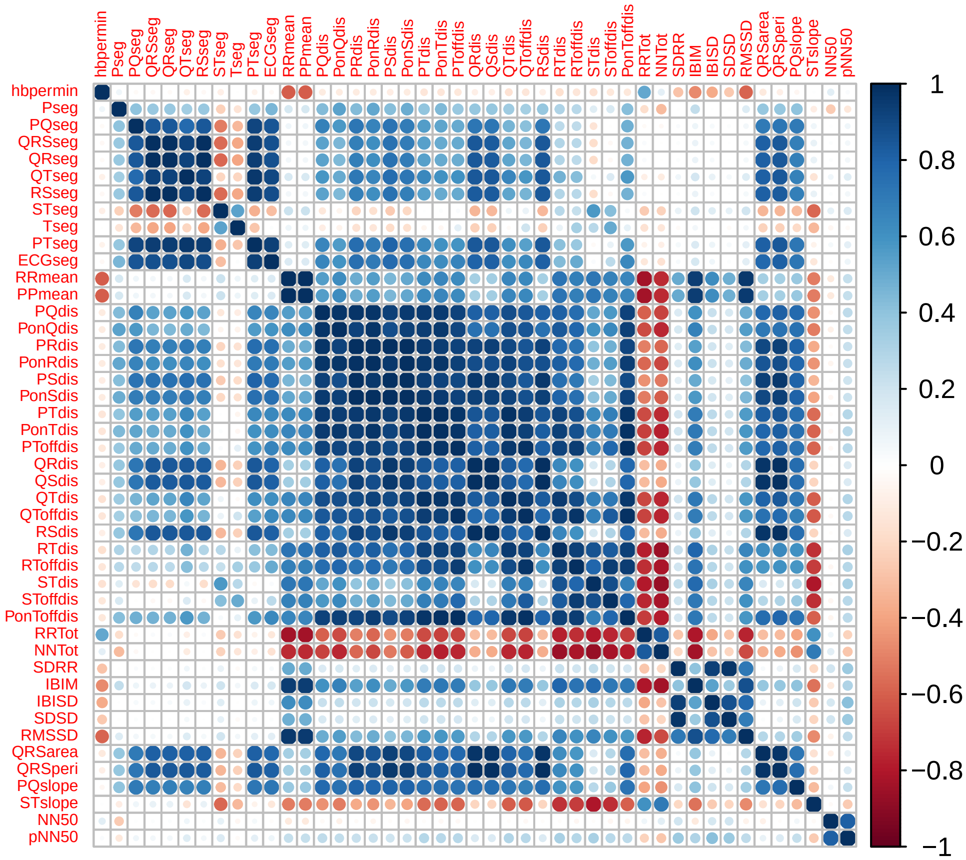
Normalización

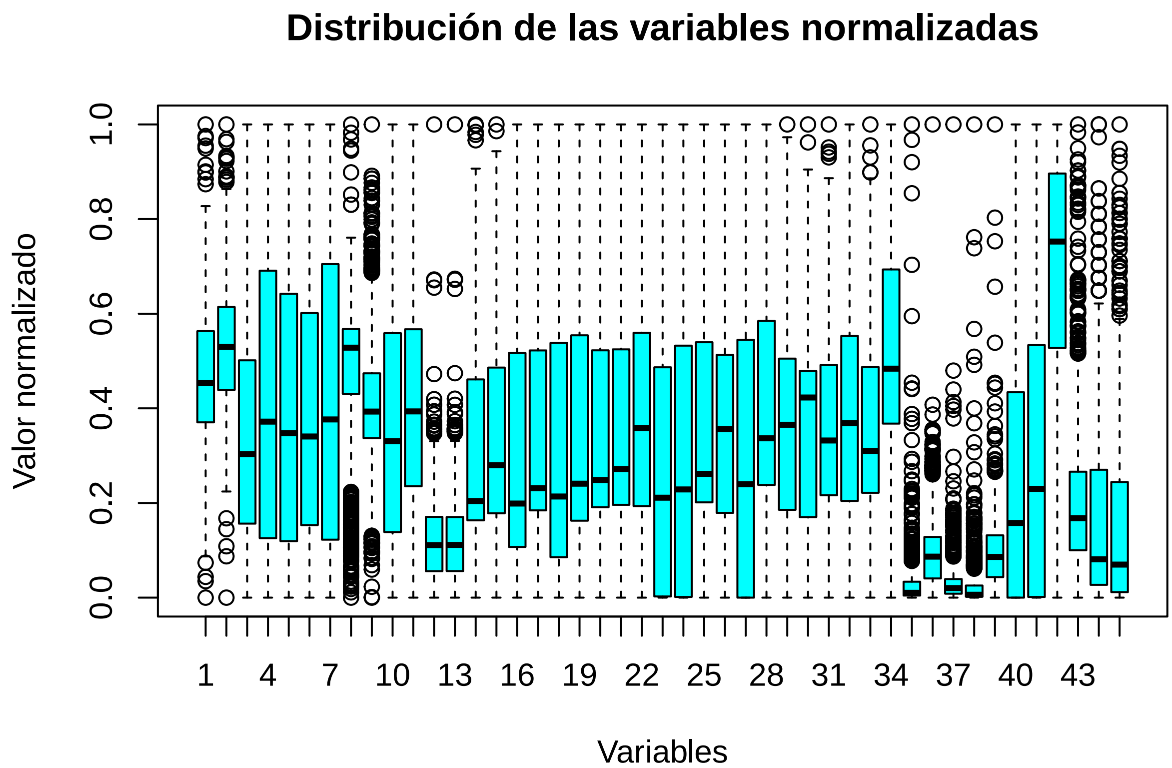
Con este conjunto de datos, vamos a realizar los algoritmos, pero antes, como hemos observado anteriormente, algunas variables muestran números enteros y otras números decimales. Además, la escala de las variables es diferente. Por ello, se va a proceder a hacer una normalización de los datos. Para realizar esta normalización, se va a crear una función que se aplicará al conjunto de datos siguiendo la formula:

$$Normalización = (x - min(x))/(max(x) - min(x))$$

```
ECG_signal <- dataset_nan[,46] # Extraemos la columna de clases antes de normalizar los datos.
dataset_norm <- as.data.frame(lapply(dataset_nan[1:45], normalizar)) # Aplicamos la funcion 'lapply()' f
data_norm <- data.frame(ECG_signal, dataset_norm) # Creamos el conjunto de datos normalizado.
```

Con los datos normalizados, realizamos una exploración visual de nuestros datos. Examinamos la correlación entre las variables y presentamos otro gráfico de cajas con los datos normalizados.





Partición de los datos en entrenamiento/prueba

El conjunto de datos resultante de la exploración y manejo de los datos se dividirá en dos partes: una para el conjunto de entrenamiento y otra para el conjunto de prueba. Para el conjunto de entrenamiento, dividiremos los datos en un 67% de las observaciones totales, mientras que el conjunto de prueba contendrá el 33% de las observaciones restantes.

```
set.seed(seed)

# Extraemos los indices de una muestra aleatoria de la filas de los datos
indices <- sample(1:nrow(data_norm), size = nrow(data_norm), replace = FALSE)
# Tamaño del conjunto de entrenamiento
train_size <- round(train_part * nrow(data_norm))

# Dividimos los datos en entrenamiento/prueba.
train_data <- data_norm[indices[1:train_size], ]
test_data <- data_norm[indices[(train_size + 1):nrow(data_norm)], ] # Conjunto de prueba (33%).

# Extraemos las labels/etiquetas.
train_label <- train_data$ECG_signal
test_label <- test_data$ECG_signal
```

Aplicación de Algortimos

En esta sección, exploraremos distintos algoritmos para la clasificación de nuestros datos. Utilizaremos la misma división de los datos para cada algoritmo y seguiremos las directrices del libro de referencia (Lantz 2019) para implementar los modelos. Los algoritmos a explorar son:

1. k-Nearest Neighbour
2. Naive Bayes
3. Artificial Neural Network
4. Support Vector Machine
5. Árbol de Clasificación
6. Random Forest

La implementación de estos algoritmos la realizaremos íntegramente con R y RStudio.

k-Nearest Neighbour (kNN)

Para implementar el algoritmo **k-Nearest Neighbors (kNN)**, utilizamos el paquete ***class*** (Venables and Ripley 2002) con la función `knn()` y exploraremos los valores de k = 1, 3, 5, 7, 11. Posteriormente, se presentarán las matrices de confusión para cada modelo utilizando el paquete *'caret'* (Kuhn and Max 2008).

```
set.seed(seed)

# Listas para almacenar los modelos y las matrices resultantes.
models <- list()
confusion_matrices <- list()

for (k in k_values) {
  model_name <- paste("model_knn_k", k, sep = "")
  # Entrenamiento del modelo con los datos establecidos para cada valor de 'k':
  model <- knn(train = train_data[,-1], test = test_data[,-1], cl = train_data$ECG_signal, k = k)
  # Almacenamos cada modelo resultante de cada valor de 'k':
  models[[model_name]] <- model

  # Calculamos la matriz de confusión y almacenamos:
  cm <- confusionMatrix(data = model, reference = as.factor(test_data$ECG_signal))
  confusion_matrices[[model_name]] <- cm
}
# Imprimimos las matrices de confusión:
print(confusion_matrices)
```

```
$model_knn_k1
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF 102   0   2   0
       ARR   0  80   0   0
       CHF   9   0 104   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9722
                 95% CI : (0.9508, 0.9861)
    No Information Rate : 0.2803
```

```
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9628

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.9189      1.000     0.9811       1.00
Specificity              0.9930      1.000     0.9690       1.00
Pos Pred Value           0.9808      1.000     0.9204       1.00
Neg Pred Value           0.9692      1.000     0.9929       1.00
Prevalence               0.2803      0.202     0.2677       0.25
Detection Rate           0.2576      0.202     0.2626       0.25
Detection Prevalence     0.2626      0.202     0.2854       0.25
Balanced Accuracy        0.9560      1.000     0.9750       1.00

$model_knn_k3
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  96   0   3   0
       ARR   0  80   0   0
       CHF  15   0 103   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9545
                 95% CI : (0.9291, 0.9728)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9391

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8649      1.000     0.9717       1.00
Specificity              0.9895      1.000     0.9483       1.00
Pos Pred Value           0.9697      1.000     0.8729       1.00
Neg Pred Value           0.9495      1.000     0.9892       1.00
Prevalence               0.2803      0.202     0.2677       0.25
Detection Rate           0.2424      0.202     0.2601       0.25
Detection Prevalence     0.2500      0.202     0.2980       0.25
Balanced Accuracy        0.9272      1.000     0.9600       1.00

$model_knn_k5
Confusion Matrix and Statistics
```

```
          Reference
Prediction AFF ARR CHF NSR
       AFF  91   0   4   0
       ARR   0  80   0   0
       CHF  20   0 102   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9394
                 95% CI : (0.9112, 0.9608)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9189

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8198      1.000     0.9623       1.00
Specificity              0.9860      1.000     0.9310       1.00
Pos Pred Value           0.9579      1.000     0.8361       1.00
Neg Pred Value           0.9336      1.000     0.9854       1.00
Prevalence               0.2803      0.202     0.2677       0.25
Detection Rate           0.2298      0.202     0.2576       0.25
Detection Prevalence     0.2399      0.202     0.3081       0.25
Balanced Accuracy        0.9029      1.000     0.9466       1.00

$model_knn_k7
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  92   0   4   0
       ARR   0  80   0   0
       CHF  19   0 102   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9419
                 95% CI : (0.9141, 0.9628)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9222

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
```

```
Sensitivity              0.8288      1.000     0.9623       1.00
Specificity              0.9860      1.000     0.9345       1.00
Pos Pred Value           0.9583      1.000     0.8430       1.00
Neg Pred Value           0.9367      1.000     0.9855       1.00
Prevalence               0.2803      0.202     0.2677       0.25
Detection Rate           0.2323      0.202     0.2576       0.25
Detection Prevalence     0.2424      0.202     0.3056       0.25
Balanced Accuracy        0.9074      1.000     0.9484       1.00

$model_knn_k11
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  92   0  10   0
       ARR   0  80   0   0
       CHF  19   0  96   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9268
                 95% CI : (0.8965, 0.9504)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9019

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8288      1.000     0.9057       1.00
Specificity              0.9649      1.000     0.9345       1.00
Pos Pred Value           0.9020      1.000     0.8348       1.00
Neg Pred Value           0.9354      1.000     0.9644       1.00
Prevalence               0.2803      0.202     0.2677       0.25
Detection Rate           0.2323      0.202     0.2424       0.25
Detection Prevalence     0.2576      0.202     0.2904       0.25
Balanced Accuracy        0.8969      1.000     0.9201       1.00
```

Table 2: Comparación de Modelos (‘kNN’)

	Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
1	kNN k = 1	0.972	0.963	0.028	0.919	1	0.981	1
2	kNN k = 3	0.955	0.939	0.045	0.865	1	0.972	1
4	kNN k = 7	0.942	0.922	0.058	0.829	1	0.962	1
3	kNN k = 5	0.939	0.919	0.061	0.820	1	0.962	1

	Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
5	kNN k = 11	0.927	0.902	0.073	0.829	1	0.906	1

De todos los valores de **k** (1, 3, 5, 7, 11), el modelo con el valor de kNN k = 1 tiene la mejor precisión y valor de kappa, con un 97.2% y 96.3%, respectivamente. Además, presenta el menor porcentaje de error con un 2.8%.

También observamos en la tabla los valores de sensibilidad de cada clase. El modelo con las mejores métricas tiene un valor de sensibilidad del 91.9% para la clase AFF, 100% para la clase ARR, 98.1% para la clase CHF y 100% para la última clase NSR.

Naive Bayes

En este apartado, implementamos el algoritmo de **Naive Bayes** con el paquete `e1071` (Meyer et al. 2023) y visualizaremos las matrices de confusión con el paquete *'caret'*. Exploraremos la opción de activar (laplace = 1) o no activar (laplace = 0) **Laplace**.

```r
set.seed(seed)

# Modelo Nain Bayes con el valor laplace = 0.
naive_model_1 <- naiveBayes(train_data, train_data$ECG_signal, laplace=0)
naive_test_pred_1 <- predict(naive_model_1, test_data)
confusion_matrix_naive_1 <- confusionMatrix(naive_test_pred_1, as.factor(test_label))
print(confusion_matrix_naive_1)
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  99   8  15   2
       ARR   1  72   0   0
       CHF  11   0  91   0
       NSR   0   0   0  97

Overall Statistics

               Accuracy : 0.9066
                 95% CI : (0.8735, 0.9334)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8746

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8919     0.9000     0.8585     0.9798
Specificity              0.9123     0.9968     0.9621     1.0000
Pos Pred Value           0.7984     0.9863     0.8922     1.0000
Neg Pred Value           0.9559     0.9752     0.9490     0.9933
Prevalence               0.2803     0.2020     0.2677     0.2500
```

```
Detection Rate            0.2500     0.1818     0.2298     0.2449
Detection Prevalence      0.3131     0.1843     0.2576     0.2449
Balanced Accuracy         0.9021     0.9484     0.9103     0.9899
```

```r
# Modelo Nain Bayes con el valor laplace = 1.
naive_model_2 <- naiveBayes(train_data, train_data$ECG_signa, laplace=1)
naive_test_pred_2 <- predict(naive_model_2, test_data)
confusion_matrix_naive_2 <- confusionMatrix(naive_test_pred_2, as.factor(test_label))
print(confusion_matrix_naive_2)
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  99   8  16   2
       ARR   1  72   0   0
       CHF  11   0  90   0
       NSR   0   0   0  97

Overall Statistics

               Accuracy : 0.904
                 95% CI : (0.8707, 0.9312)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8712

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8919     0.9000     0.8491     0.9798
Specificity              0.9088     0.9968     0.9621     1.0000
Pos Pred Value           0.7920     0.9863     0.8911     1.0000
Neg Pred Value           0.9557     0.9752     0.9458     0.9933
Prevalence               0.2803     0.2020     0.2677     0.2500
Detection Rate           0.2500     0.1818     0.2273     0.2449
Detection Prevalence     0.3157     0.1843     0.2551     0.2449
Balanced Accuracy        0.9003     0.9484     0.9056     0.9899
```

Table 3: Comparación de Modelos ('Nain Bayes')

Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
NB Laplace = 0	0.907	0.875	0.093	0.892	0.9	0.858	0.98
NB Laplace =1	0.904	0.871	0.096	0.892	0.9	0.849	0.98

Al comparar los modelos en la tabla anterior observamos que el modelo obtenido sin activar laplace tiene una mayor precisión , el valor kappa también es superior en el modelo sin laplace y el valor de error es superior en el modelo con laplace .

Los valores de sensibilidad para el modelo con mejor precisión son 89.2% para la clase AFF, 90% para la clase ARR, 85.8% para la clase CHF y 98% para la última clase NSR.

Si ambos modelos son muy similares seleccionaremos el modelo más sencillo.

Artificial Neural Network

Implementamos el algoritmo **Artificial Neural Network** con el paquete *keras* (Allaire and Chollet 2023). Exploraremos dos tipos de modelos: uno con una única capa oculta con 15 nodos, a la que añadiremos un dropout del 20% para evitar el sobreajuste, y una capa de salida con 4 nodos coincidiendo con nuestras clases de estudio. El segundo modelo contendrá dos capas ocultas con 15 y 35 nodos, a las que se añadirá una capa de dropout del 20%, junto con la capa de salida. Ambos modelos se compilarán y entrenarán, mostrando las gráficas de pérdida y precisión. Por último, visualizaremos las matrices de confusión con el paquete *caret*.

Antes de implementar el modelo, realizaremos un ajuste en nuestros datos de entrenamiento y prueba. Convertiremos nuestras etiquetas a valores numéricos para poder ejecutar el modelo. En la siguiente tabla, observamos la equivalencia de la transformación de las clases de estudio:

Clases	Clases_ann
AFF	0
ARR	1
CHF	2
NSR	3

```r
# Ajustamos los datos de entramiento y prueba para adaptarlos al algoritmo ann.
train_data_ann <- as.matrix(train_data[,-1])
test_data_ann <- as.matrix(test_data[,-1])

# Transformamos nuestras clases a valores numericos, del 0 al 3.
train_label_ann <- as.numeric(factor(train_label)) - 1
test_label_ann <- as.numeric(factor(test_label)) - 1
```

```r
set.seed(seed)

# Modelo ann con una capa oculta.
modelo_ann_1 <- keras_model_sequential() %>%
  # Agregamos una capa densa al modelo con 15 nodos. Establecemos la activacion 'relu' para introducir
  layer_dense(units = 15, activation = 'relu', input_shape = ncol(train_data_ann)) %>%
  # Capa de Dropout desactivando aleatoriamente el 20% de las neuronas para prevenir el sobreajuste.
  layer_dropout(rate = 0.2) %>%
  #Capa de salida, usamos la función de activación 'softmax'.
  layer_dense(units = 4, activation = 'softmax')
summary(modelo_ann_1)
```

```
Model: "sequential"
________________________________________________________________________________________
 Layer (type)                          Output Shape                       Param #
========================================================================================
 dense_1 (Dense)                       (None, 15)                         690
 dropout (Dropout)                     (None, 15)                         0
 dense (Dense)                         (None, 4)                          64
========================================================================================
Total params: 754 (2.95 KB)
Trainable params: 754 (2.95 KB)
```

```
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________________________________
```

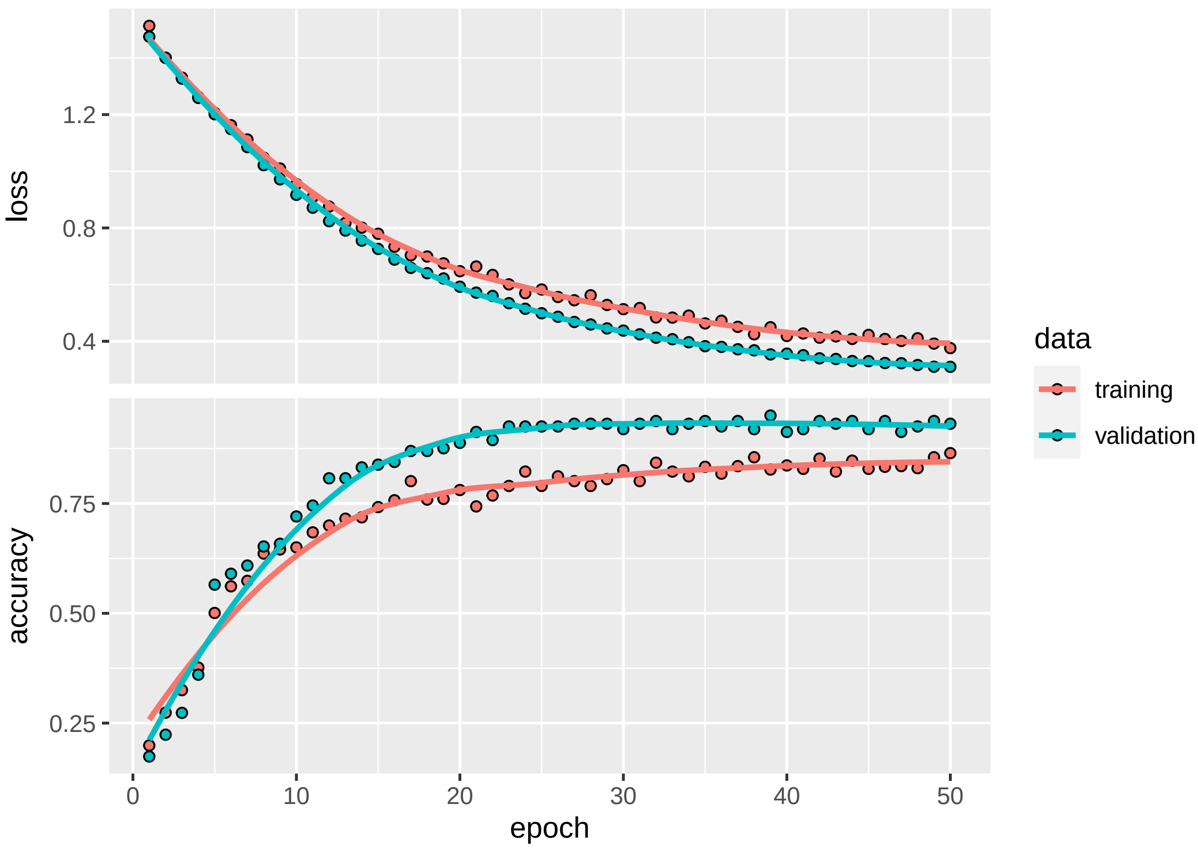
```r
set.seed(seed)

# Compilamos el modelo_ann_1:
modelo_ann_1 %>% compile(
  loss = 'sparse_categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
# Entrenamos el model0_ann_1 y almacenamos la información en la variable "history__ann_1":
history_ann_1 <- modelo_ann_1 %>% fit(
  x = train_data_ann,
  y = train_label_ann,
  epochs = 50, # Establecemos las épocas de interacciones del entrenamiento a 50.
  validation_split = 0.2, # 20% de los datos se usarán como datos de validación.
  verbose = 2 # Nivel de detalle durante el entrenamiento.
)
```

```
Epoch 1/50
21/21 - 1s - loss: 1.5131 - accuracy: 0.1991 - val_loss: 1.4749 - val_accuracy: 0.1739 - 585ms/epoch -
Epoch 2/50
21/21 - 0s - loss: 1.4000 - accuracy: 0.2737 - val_loss: 1.4008 - val_accuracy: 0.2236 - 63ms/epoch - 3
Epoch 3/50
21/21 - 0s - loss: 1.3304 - accuracy: 0.3250 - val_loss: 1.3267 - val_accuracy: 0.2733 - 50ms/epoch - 2
Epoch 4/50
21/21 - 0s - loss: 1.2626 - accuracy: 0.3764 - val_loss: 1.2589 - val_accuracy: 0.3602 - 45ms/epoch - 2
Epoch 5/50
21/21 - 0s - loss: 1.2062 - accuracy: 0.5008 - val_loss: 1.2013 - val_accuracy: 0.5652 - 48ms/epoch - 2
Epoch 6/50
21/21 - 0s - loss: 1.1623 - accuracy: 0.5614 - val_loss: 1.1489 - val_accuracy: 0.5901 - 48ms/epoch - 2
Epoch 7/50
21/21 - 0s - loss: 1.1122 - accuracy: 0.5739 - val_loss: 1.0853 - val_accuracy: 0.6087 - 52ms/epoch - 2
Epoch 8/50
21/21 - 0s - loss: 1.0480 - accuracy: 0.6361 - val_loss: 1.0214 - val_accuracy: 0.6522 - 50ms/epoch - 2
Epoch 9/50
21/21 - 0s - loss: 1.0096 - accuracy: 0.6454 - val_loss: 0.9715 - val_accuracy: 0.6584 - 49ms/epoch - 2
Epoch 10/50
21/21 - 0s - loss: 0.9547 - accuracy: 0.6501 - val_loss: 0.9167 - val_accuracy: 0.7205 - 49ms/epoch - 2
Epoch 11/50
21/21 - 0s - loss: 0.9089 - accuracy: 0.6843 - val_loss: 0.8715 - val_accuracy: 0.7453 - 53ms/epoch - 3
Epoch 12/50
21/21 - 0s - loss: 0.8754 - accuracy: 0.6998 - val_loss: 0.8238 - val_accuracy: 0.8075 - 51ms/epoch - 2
Epoch 13/50
21/21 - 0s - loss: 0.8179 - accuracy: 0.7154 - val_loss: 0.7908 - val_accuracy: 0.8075 - 50ms/epoch - 2
Epoch 14/50
21/21 - 0s - loss: 0.8010 - accuracy: 0.7185 - val_loss: 0.7551 - val_accuracy: 0.8323 - 53ms/epoch - 3
Epoch 15/50
21/21 - 0s - loss: 0.7795 - accuracy: 0.7418 - val_loss: 0.7259 - val_accuracy: 0.8385 - 48ms/epoch - 2
Epoch 16/50
21/21 - 0s - loss: 0.7337 - accuracy: 0.7574 - val_loss: 0.6881 - val_accuracy: 0.8447 - 47ms/epoch - 2
Epoch 17/50
21/21 - 0s - loss: 0.7034 - accuracy: 0.8009 - val_loss: 0.6590 - val_accuracy: 0.8696 - 49ms/epoch - 2
Epoch 18/50
```

```
21/21 - 0s - loss: 0.6991 - accuracy: 0.7589 - val_loss: 0.6402 - val_accuracy: 0.8696 - 49ms/epoch - 2r
Epoch 19/50
21/21 - 0s - loss: 0.6750 - accuracy: 0.7605 - val_loss: 0.6217 - val_accuracy: 0.8758 - 50ms/epoch - 2r
Epoch 20/50
21/21 - 0s - loss: 0.6473 - accuracy: 0.7807 - val_loss: 0.5926 - val_accuracy: 0.8882 - 50ms/epoch - 2r
Epoch 21/50
21/21 - 0s - loss: 0.6639 - accuracy: 0.7434 - val_loss: 0.5717 - val_accuracy: 0.9130 - 52ms/epoch - 2r
Epoch 22/50
21/21 - 0s - loss: 0.6342 - accuracy: 0.7683 - val_loss: 0.5598 - val_accuracy: 0.8944 - 49ms/epoch - 2r
Epoch 23/50
21/21 - 0s - loss: 0.6007 - accuracy: 0.7900 - val_loss: 0.5344 - val_accuracy: 0.9255 - 50ms/epoch - 2r
Epoch 24/50
21/21 - 0s - loss: 0.5698 - accuracy: 0.8227 - val_loss: 0.5147 - val_accuracy: 0.9255 - 51ms/epoch - 2r
Epoch 25/50
21/21 - 0s - loss: 0.5827 - accuracy: 0.7900 - val_loss: 0.4989 - val_accuracy: 0.9255 - 48ms/epoch - 2r
Epoch 26/50
21/21 - 0s - loss: 0.5563 - accuracy: 0.8118 - val_loss: 0.4865 - val_accuracy: 0.9255 - 49ms/epoch - 2r
Epoch 27/50
21/21 - 0s - loss: 0.5446 - accuracy: 0.8009 - val_loss: 0.4681 - val_accuracy: 0.9317 - 49ms/epoch - 2r
Epoch 28/50
21/21 - 0s - loss: 0.5624 - accuracy: 0.7900 - val_loss: 0.4593 - val_accuracy: 0.9317 - 49ms/epoch - 2r
Epoch 29/50
21/21 - 0s - loss: 0.5282 - accuracy: 0.8056 - val_loss: 0.4454 - val_accuracy: 0.9317 - 48ms/epoch - 2r
Epoch 30/50
21/21 - 0s - loss: 0.5132 - accuracy: 0.8258 - val_loss: 0.4377 - val_accuracy: 0.9193 - 50ms/epoch - 2r
Epoch 31/50
21/21 - 0s - loss: 0.5174 - accuracy: 0.8009 - val_loss: 0.4245 - val_accuracy: 0.9317 - 48ms/epoch - 2r
Epoch 32/50
21/21 - 0s - loss: 0.4840 - accuracy: 0.8429 - val_loss: 0.4127 - val_accuracy: 0.9379 - 50ms/epoch - 2r
Epoch 33/50
21/21 - 0s - loss: 0.4836 - accuracy: 0.8227 - val_loss: 0.4073 - val_accuracy: 0.9193 - 48ms/epoch - 2r
Epoch 34/50
21/21 - 0s - loss: 0.4907 - accuracy: 0.8118 - val_loss: 0.3966 - val_accuracy: 0.9317 - 51ms/epoch - 2r
Epoch 35/50
21/21 - 0s - loss: 0.4630 - accuracy: 0.8336 - val_loss: 0.3827 - val_accuracy: 0.9379 - 58ms/epoch - 3r
Epoch 36/50
21/21 - 0s - loss: 0.4725 - accuracy: 0.8180 - val_loss: 0.3803 - val_accuracy: 0.9255 - 48ms/epoch - 2r
Epoch 37/50
21/21 - 0s - loss: 0.4510 - accuracy: 0.8351 - val_loss: 0.3718 - val_accuracy: 0.9379 - 48ms/epoch - 2r
Epoch 38/50
21/21 - 0s - loss: 0.4246 - accuracy: 0.8554 - val_loss: 0.3681 - val_accuracy: 0.9193 - 49ms/epoch - 2r
Epoch 39/50
21/21 - 0s - loss: 0.4494 - accuracy: 0.8274 - val_loss: 0.3536 - val_accuracy: 0.9503 - 49ms/epoch - 2r
Epoch 40/50
21/21 - 0s - loss: 0.4186 - accuracy: 0.8367 - val_loss: 0.3565 - val_accuracy: 0.9130 - 50ms/epoch - 2r
Epoch 41/50
21/21 - 0s - loss: 0.4276 - accuracy: 0.8289 - val_loss: 0.3509 - val_accuracy: 0.9193 - 48ms/epoch - 2r
Epoch 42/50
21/21 - 0s - loss: 0.4126 - accuracy: 0.8523 - val_loss: 0.3399 - val_accuracy: 0.9379 - 48ms/epoch - 2r
Epoch 43/50
21/21 - 0s - loss: 0.4171 - accuracy: 0.8227 - val_loss: 0.3375 - val_accuracy: 0.9317 - 48ms/epoch - 2r
Epoch 44/50
21/21 - 0s - loss: 0.4083 - accuracy: 0.8476 - val_loss: 0.3303 - val_accuracy: 0.9379 - 50ms/epoch - 2r
Epoch 45/50
```

```
21/21 - 0s - loss: 0.4225 - accuracy: 0.8289 - val_loss: 0.3299 - val_accuracy: 0.9193 - 49ms/epoch - 2r
Epoch 46/50
21/21 - 0s - loss: 0.4080 - accuracy: 0.8336 - val_loss: 0.3233 - val_accuracy: 0.9379 - 46ms/epoch - 2r
Epoch 47/50
21/21 - 0s - loss: 0.4011 - accuracy: 0.8351 - val_loss: 0.3225 - val_accuracy: 0.9130 - 51ms/epoch - 2r
Epoch 48/50
21/21 - 0s - loss: 0.4106 - accuracy: 0.8305 - val_loss: 0.3163 - val_accuracy: 0.9255 - 49ms/epoch - 2r
Epoch 49/50
21/21 - 0s - loss: 0.3918 - accuracy: 0.8554 - val_loss: 0.3102 - val_accuracy: 0.9379 - 51ms/epoch - 2r
Epoch 50/50
21/21 - 0s - loss: 0.3759 - accuracy: 0.8647 - val_loss: 0.3098 - val_accuracy: 0.9317 - 49ms/epoch - 2r
```

```r
plot(history_ann_1) # Mostramos los gáficos de perdida y precisión.
```




```r
set.seed(seed)
predicciones_ann_1 <- predict(modelo_ann_1, test_data_ann)
```

```
13/13 - 0s - 64ms/epoch - 5ms/step
```

```r
y_pred_ann_1 <- as.factor(max.col(predicciones_ann_1) - 1)
matriz_confusion_ann_1 <- confusionMatrix(y_pred_ann_1, as.factor(test_label_ann))
matriz_confusion_ann_1
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2  3
         0 69  0 17  0
```

```
         1  0 80  2  0
         2 42  0 87  0
         3  0  0  0 99

Overall Statistics

               Accuracy : 0.846
                 95% CI : (0.8066, 0.8801)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.794

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3
Sensitivity            0.6216   1.0000   0.8208     1.00
Specificity            0.9404   0.9937   0.8552     1.00
Pos Pred Value         0.8023   0.9756   0.6744     1.00
Neg Pred Value         0.8645   1.0000   0.9288     1.00
Prevalence             0.2803   0.2020   0.2677     0.25
Detection Rate         0.1742   0.2020   0.2197     0.25
Detection Prevalence   0.2172   0.2071   0.3258     0.25
Balanced Accuracy      0.7810   0.9968   0.8380     1.00
```

```
# Modelo ann con dos capas ocultas.
modelo_ann_2 <- keras_model_sequential() %>%
  layer_dense(units = 25, activation = 'relu', input_shape = ncol(train_data_ann)) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 15, activation = 'relu') %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 4, activation = 'softmax')
summary(modelo_ann_2)
```

```
Model: "sequential_1"
________________________________________________________________________________
 Layer (type)                       Output Shape                    Param #
================================================================================
 dense_4 (Dense)                    (None, 25)                      1150
 dropout_2 (Dropout)                (None, 25)                      0
 dense_3 (Dense)                    (None, 15)                      390
 dropout_1 (Dropout)                (None, 15)                      0
 dense_2 (Dense)                    (None, 4)                       64
================================================================================
Total params: 1604 (6.27 KB)
Trainable params: 1604 (6.27 KB)
Non-trainable params: 0 (0.00 Byte)
________________________________________________________________________________
```

```
set.seed(seed)

# Compilamos el modelo_ann_2:
modelo_ann_2 %>% keras::compile(
```

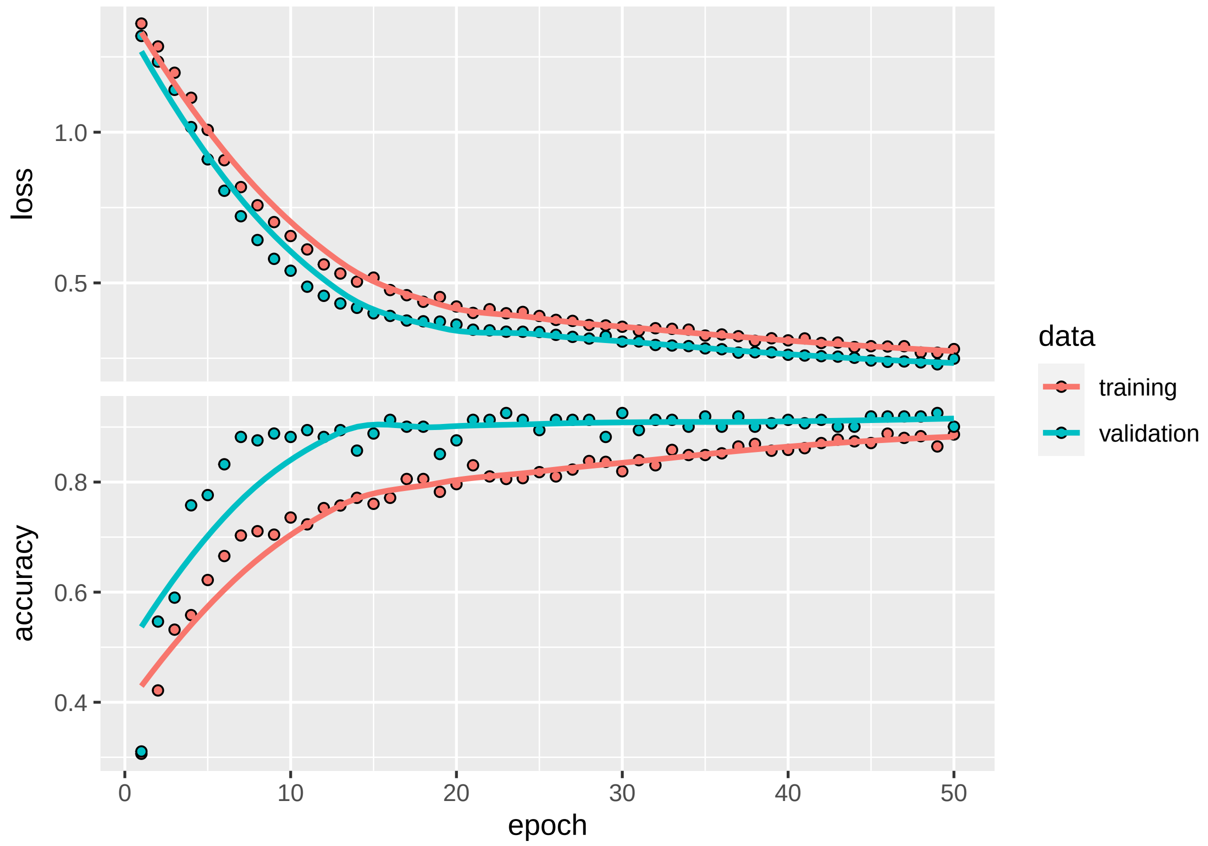
```
  loss = 'sparse_categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
# Entrenamos el modelo_ann_2 y almacenamos la información en la variable "history__ann_2":
history_ann_2 <- modelo_ann_2 %>% fit(
  x = train_data_ann,
  y = train_label_ann,
  epochs = 50,
  validation_split = 0.2,
  verbose = 2
)
```

```
Epoch 1/50
21/21 - 1s - loss: 1.3605 - accuracy: 0.3064 - val_loss: 1.3194 - val_accuracy: 0.3106 - 684ms/epoch - 3
Epoch 2/50
21/21 - 0s - loss: 1.2847 - accuracy: 0.4215 - val_loss: 1.2343 - val_accuracy: 0.5466 - 48ms/epoch - 2
Epoch 3/50
21/21 - 0s - loss: 1.1973 - accuracy: 0.5319 - val_loss: 1.1406 - val_accuracy: 0.5901 - 50ms/epoch - 2
Epoch 4/50
21/21 - 0s - loss: 1.1143 - accuracy: 0.5583 - val_loss: 1.0170 - val_accuracy: 0.7578 - 50ms/epoch - 2
Epoch 5/50
21/21 - 0s - loss: 1.0077 - accuracy: 0.6221 - val_loss: 0.9099 - val_accuracy: 0.7764 - 55ms/epoch - 3
Epoch 6/50
21/21 - 0s - loss: 0.9071 - accuracy: 0.6656 - val_loss: 0.8057 - val_accuracy: 0.8323 - 49ms/epoch - 2
Epoch 7/50
21/21 - 0s - loss: 0.8179 - accuracy: 0.7030 - val_loss: 0.7213 - val_accuracy: 0.8820 - 51ms/epoch - 2
Epoch 8/50
21/21 - 0s - loss: 0.7576 - accuracy: 0.7107 - val_loss: 0.6423 - val_accuracy: 0.8758 - 54ms/epoch - 3
Epoch 9/50
21/21 - 0s - loss: 0.7018 - accuracy: 0.7045 - val_loss: 0.5799 - val_accuracy: 0.8882 - 48ms/epoch - 2
Epoch 10/50
21/21 - 0s - loss: 0.6556 - accuracy: 0.7356 - val_loss: 0.5404 - val_accuracy: 0.8820 - 52ms/epoch - 2
Epoch 11/50
21/21 - 0s - loss: 0.6111 - accuracy: 0.7232 - val_loss: 0.4874 - val_accuracy: 0.8944 - 50ms/epoch - 2
Epoch 12/50
21/21 - 0s - loss: 0.5613 - accuracy: 0.7527 - val_loss: 0.4571 - val_accuracy: 0.8820 - 48ms/epoch - 2
Epoch 13/50
21/21 - 0s - loss: 0.5311 - accuracy: 0.7574 - val_loss: 0.4317 - val_accuracy: 0.8944 - 53ms/epoch - 3
Epoch 14/50
21/21 - 0s - loss: 0.5041 - accuracy: 0.7714 - val_loss: 0.4173 - val_accuracy: 0.8571 - 49ms/epoch - 2
Epoch 15/50
21/21 - 0s - loss: 0.5175 - accuracy: 0.7605 - val_loss: 0.3990 - val_accuracy: 0.8882 - 50ms/epoch - 2
Epoch 16/50
21/21 - 0s - loss: 0.4763 - accuracy: 0.7714 - val_loss: 0.3904 - val_accuracy: 0.9130 - 48ms/epoch - 2
Epoch 17/50
21/21 - 0s - loss: 0.4593 - accuracy: 0.8056 - val_loss: 0.3751 - val_accuracy: 0.9006 - 52ms/epoch - 2
Epoch 18/50
21/21 - 0s - loss: 0.4373 - accuracy: 0.8056 - val_loss: 0.3727 - val_accuracy: 0.9006 - 48ms/epoch - 2
Epoch 19/50
21/21 - 0s - loss: 0.4531 - accuracy: 0.7823 - val_loss: 0.3716 - val_accuracy: 0.8509 - 50ms/epoch - 2
Epoch 20/50
21/21 - 0s - loss: 0.4217 - accuracy: 0.7963 - val_loss: 0.3619 - val_accuracy: 0.8758 - 52ms/epoch - 2
Epoch 21/50
```

```
21/21 - 0s - loss: 0.4003 - accuracy: 0.8305 - val_loss: 0.3447 - val_accuracy: 0.9130 - 51ms/epoch - 2m
Epoch 22/50
21/21 - 0s - loss: 0.4127 - accuracy: 0.8103 - val_loss: 0.3427 - val_accuracy: 0.9130 - 49ms/epoch - 2m
Epoch 23/50
21/21 - 0s - loss: 0.3993 - accuracy: 0.8056 - val_loss: 0.3381 - val_accuracy: 0.9255 - 49ms/epoch - 2m
Epoch 24/50
21/21 - 0s - loss: 0.4036 - accuracy: 0.8072 - val_loss: 0.3379 - val_accuracy: 0.9130 - 49ms/epoch - 2m
Epoch 25/50
21/21 - 0s - loss: 0.3903 - accuracy: 0.8180 - val_loss: 0.3372 - val_accuracy: 0.8944 - 50ms/epoch - 2m
Epoch 26/50
21/21 - 0s - loss: 0.3772 - accuracy: 0.8103 - val_loss: 0.3276 - val_accuracy: 0.9130 - 53ms/epoch - 3m
Epoch 27/50
21/21 - 0s - loss: 0.3740 - accuracy: 0.8227 - val_loss: 0.3209 - val_accuracy: 0.9130 - 49ms/epoch - 2m
Epoch 28/50
21/21 - 0s - loss: 0.3602 - accuracy: 0.8383 - val_loss: 0.3154 - val_accuracy: 0.9130 - 51ms/epoch - 2m
Epoch 29/50
21/21 - 0s - loss: 0.3589 - accuracy: 0.8367 - val_loss: 0.3234 - val_accuracy: 0.8820 - 50ms/epoch - 2m
Epoch 30/50
21/21 - 0s - loss: 0.3545 - accuracy: 0.8196 - val_loss: 0.3051 - val_accuracy: 0.9255 - 50ms/epoch - 2m
Epoch 31/50
21/21 - 0s - loss: 0.3416 - accuracy: 0.8398 - val_loss: 0.3057 - val_accuracy: 0.8944 - 103ms/epoch - 5
Epoch 32/50
21/21 - 0s - loss: 0.3496 - accuracy: 0.8305 - val_loss: 0.2939 - val_accuracy: 0.9130 - 53ms/epoch - 3m
Epoch 33/50
21/21 - 0s - loss: 0.3476 - accuracy: 0.8585 - val_loss: 0.2922 - val_accuracy: 0.9130 - 51ms/epoch - 2m
Epoch 34/50
21/21 - 0s - loss: 0.3450 - accuracy: 0.8491 - val_loss: 0.2902 - val_accuracy: 0.9006 - 49ms/epoch - 2m
Epoch 35/50
21/21 - 0s - loss: 0.3252 - accuracy: 0.8491 - val_loss: 0.2827 - val_accuracy: 0.9193 - 51ms/epoch - 2m
Epoch 36/50
21/21 - 0s - loss: 0.3289 - accuracy: 0.8523 - val_loss: 0.2798 - val_accuracy: 0.9006 - 50ms/epoch - 2m
Epoch 37/50
21/21 - 0s - loss: 0.3231 - accuracy: 0.8647 - val_loss: 0.2685 - val_accuracy: 0.9193 - 49ms/epoch - 2m
Epoch 38/50
21/21 - 0s - loss: 0.3077 - accuracy: 0.8694 - val_loss: 0.2697 - val_accuracy: 0.9006 - 53ms/epoch - 3m
Epoch 39/50
21/21 - 0s - loss: 0.3163 - accuracy: 0.8569 - val_loss: 0.2696 - val_accuracy: 0.9068 - 52ms/epoch - 2m
Epoch 40/50
21/21 - 0s - loss: 0.3094 - accuracy: 0.8585 - val_loss: 0.2615 - val_accuracy: 0.9130 - 53ms/epoch - 3m
Epoch 41/50
21/21 - 0s - loss: 0.3157 - accuracy: 0.8616 - val_loss: 0.2590 - val_accuracy: 0.9068 - 51ms/epoch - 2m
Epoch 42/50
21/21 - 0s - loss: 0.3006 - accuracy: 0.8709 - val_loss: 0.2569 - val_accuracy: 0.9130 - 51ms/epoch - 2m
Epoch 43/50
21/21 - 0s - loss: 0.3021 - accuracy: 0.8771 - val_loss: 0.2553 - val_accuracy: 0.9006 - 48ms/epoch - 2m
Epoch 44/50
21/21 - 0s - loss: 0.2870 - accuracy: 0.8740 - val_loss: 0.2516 - val_accuracy: 0.9006 - 49ms/epoch - 2m
Epoch 45/50
21/21 - 0s - loss: 0.2899 - accuracy: 0.8709 - val_loss: 0.2425 - val_accuracy: 0.9193 - 51ms/epoch - 2m
Epoch 46/50
21/21 - 0s - loss: 0.2890 - accuracy: 0.8880 - val_loss: 0.2381 - val_accuracy: 0.9193 - 50ms/epoch - 2m
Epoch 47/50
21/21 - 0s - loss: 0.2899 - accuracy: 0.8802 - val_loss: 0.2394 - val_accuracy: 0.9193 - 54ms/epoch - 3m
Epoch 48/50
```

```
21/21 - 0s - loss: 0.2691 - accuracy: 0.8834 - val_loss: 0.2363 - val_accuracy: 0.9193 - 49ms/epoch - 2r
Epoch 49/50
21/21 - 0s - loss: 0.2681 - accuracy: 0.8647 - val_loss: 0.2298 - val_accuracy: 0.9255 - 50ms/epoch - 2r
Epoch 50/50
21/21 - 0s - loss: 0.2804 - accuracy: 0.8865 - val_loss: 0.2487 - val_accuracy: 0.9006 - 52ms/epoch - 2r
```

```r
plot(history_ann_2)
```




```r
set.seed(seed)
predicciones_ann_2 <- predict(modelo_ann_2, test_data_ann)
```

```
13/13 - 0s - 47ms/epoch - 4ms/step
```

```r
y_pred_ann_2 <- as.factor(max.col(predicciones_ann_2) - 1)
matriz_confusion_ann_2 <- confusionMatrix(y_pred_ann_2, as.factor(test_label_ann))
matriz_confusion_ann_2
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1  2  3
         0 63  0 14  0
         1  0 80  0  0
         2 48  0 92  0
         3  0  0  0 99

Overall Statistics
```

```
               Accuracy : 0.8434
                 95% CI : (0.8038, 0.8778)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7906

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 0 Class: 1 Class: 2 Class: 3
Sensitivity            0.5676    1.000   0.8679     1.00
Specificity            0.9509    1.000   0.8345     1.00
Pos Pred Value         0.8182    1.000   0.6571     1.00
Neg Pred Value         0.8495    1.000   0.9453     1.00
Prevalence             0.2803    0.202   0.2677     0.25
Detection Rate         0.1591    0.202   0.2323     0.25
Detection Prevalence   0.1944    0.202   0.3535     0.25
Balanced Accuracy      0.7592    1.000   0.8512     1.00
```

Table 5: Comparación de Modelos ('ANN')

Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
ANN One layer	0.846	0.794	0.154	0.622	1	0.821	1
ANN Two layers	0.843	0.791	0.157	0.568	1	0.868	1

Al comparar los modelos observamos que el modelo obtenido aplicando solo una capa oculta tiene una mayor precisión , el valor kappa también es superior en el modelo con una capa oculta y el valor de error es superior en el modelo con dos capas .

Los valores de sensibilidad para el modelo con mejor precisión son 62.2% para la clase AFF, 100% para la clase ARR, 82.1% para la clase CHF y 100% para la última clase NSR.

Si nuestros modelos son muy similares, seguiremos el principio de parsimonia y elegiremos el modelo mas sencillo.

Support Vector Machine

Para implementar el algoritmo **upport Vector Machine (SVM)** con kernel lineal y RBF usamos la funcion `ksvm` del paquete *'kernlab'* (Karatzoglou, Smola, and Hornik 2023). Para el kernel lineal usamos la opción **"vanilladot"** y **"rbfdot"** para RBF. Luego mostramos las matrices de confusión de cada modelo con la función `confusionMatrix()`del paquete *'caret'*.

```r
set.seed(seed)

# Convertimos las clases a tipo factor para poder implementar el algoritmo SVM.
train_data$ECG_signal <- as.factor(train_data$ECG_signal)
test_data$ECG_signal <- as.factor(test_data$ECG_signal)

# Modelo SVM con kernel lineal.
svm_linear_model <- kernlab::ksvm(ECG_signal ~ ., data = train_data, kernel = "vanilladot")
```

```
 Setting default kernel parameters
```

```r
svm_linear_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 110

Objective Function Value : -0.6662 -50.3855 -0.2015 -0.8317 -0.0483 -0.1472
Training error : 0.014925
```

```r
predictions_linear <- predict(svm_linear_model, newdata = test_data)

# Modelo SVM con kernel rbf>
svm_rbf_model <- kernlab::ksvm(ECG_signal ~ ., data = train_data, kernel = "rbfdot")
svm_rbf_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0259397406256988

Number of Support Vectors : 269

Objective Function Value : -20.1491 -113.2324 -8.1272 -19.8548 -5.2133 -6.7519
Training error : 0.021144
```

```r
predictions_rbf <- predict(svm_rbf_model, newdata = test_data)

#Matrices de confusión para los modelos.
linear_model <- caret::confusionMatrix(predictions_linear, test_data$ECG_signal)
rbf_model <- caret::confusionMatrix(predictions_rbf, test_data$ECG_signal)
# Mostramos los datos.
linear_model
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  96   0   1   1
       ARR   0  80   0   0
       CHF  15   0 105   0
       NSR   0   0   0  98

Overall Statistics

               Accuracy : 0.9571
                 95% CI : (0.9322, 0.9748)
```

```
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9425

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8649      1.000     0.9906     0.9899
Specificity              0.9930      1.000     0.9483     1.0000
Pos Pred Value           0.9796      1.000     0.8750     1.0000
Neg Pred Value           0.9497      1.000     0.9964     0.9966
Prevalence               0.2803      0.202     0.2677     0.2500
Detection Rate           0.2424      0.202     0.2652     0.2475
Detection Prevalence     0.2475      0.202     0.3030     0.2475
Balanced Accuracy        0.9289      1.000     0.9694     0.9949
```

```
rbf_model
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF  96   0   4   0
       ARR   0  80   0   1
       CHF  15   0 102   0
       NSR   0   0   0  98

Overall Statistics

               Accuracy : 0.9495
                 95% CI : (0.9231, 0.9689)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9324

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.8649     1.0000     0.9623     0.9899
Specificity              0.9860     0.9968     0.9483     1.0000
Pos Pred Value           0.9600     0.9877     0.8718     1.0000
Neg Pred Value           0.9493     1.0000     0.9857     0.9966
Prevalence               0.2803     0.2020     0.2677     0.2500
Detection Rate           0.2424     0.2020     0.2576     0.2475
Detection Prevalence     0.2525     0.2045     0.2955     0.2475
Balanced Accuracy        0.9254     0.9984     0.9553     0.9949
```

Table 6: Comparación de Modelos ('SVM')

Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
SVM Lineal	0.957	0.943	0.043	0.865	1	0.991	0.99
SVM gaussiano	0.949	0.932	0.051	0.865	1	0.962	0.99

Al comparar los modelos de la tabla anterior observamos que el modelo obtenido con SVM lineal tiene una mayor precisión , el valor kappa también es superior en el modelo lineal y el valor de error es superior en el modelo radial .

Los valores de sensibilidad para el modelo con mejor precisión son 86.5% para la clase AFF, 100% para la clase ARR, 99.1% para la clase CHF y 99% para la última clase NSR.

Si ambos modelos son muy similares, optaremos por el modelo más sencillo.

Árbol de Clasificación

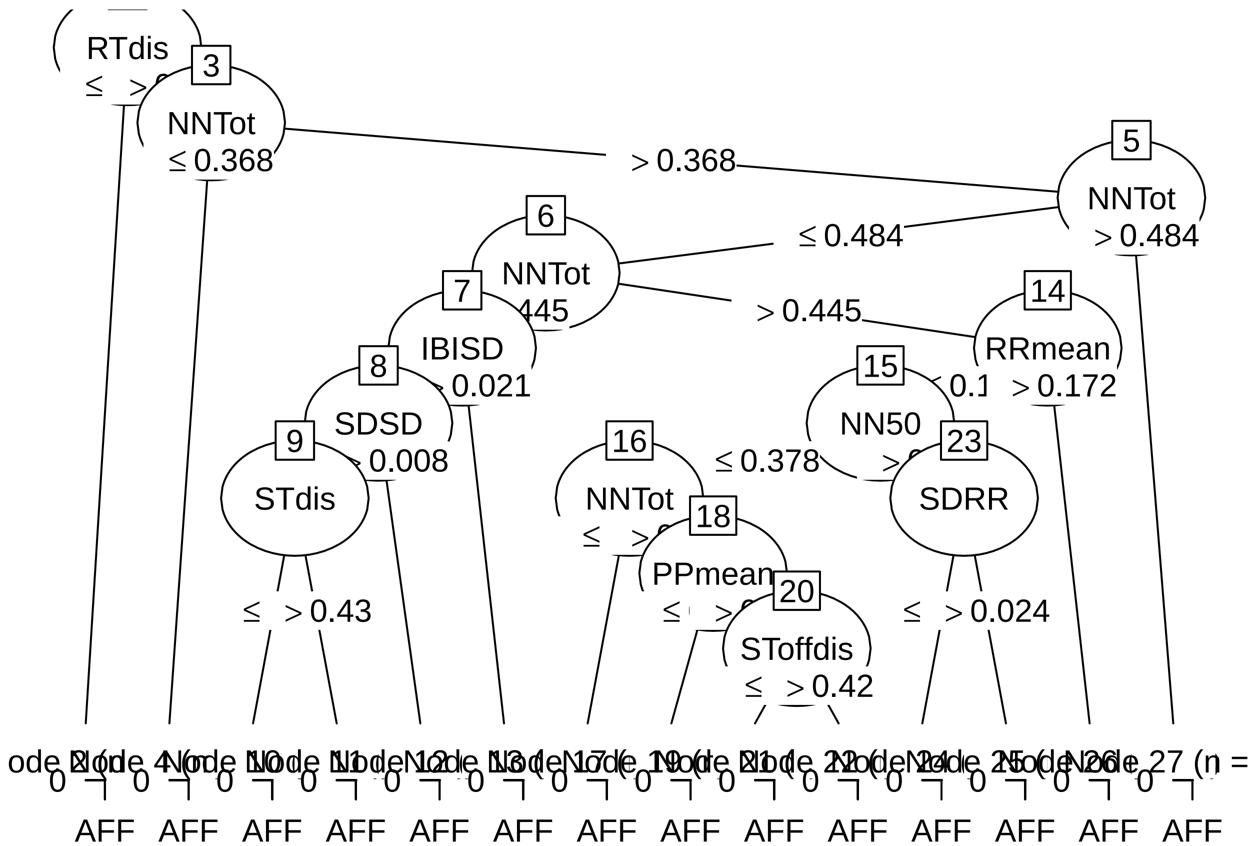
En el siguiente apartado, implementaremos el algoritmo de **Árbol de clasificación** con el paquete *`C50`* (Kuhn and Quinlan 2023) y la función `C5.0()`. En este caso, exploraremos la implementación de dos modelos: uno sin activar la opción de boosting y otro modelo activando el boosting usando un valor de *trials* igual a 10. También incluiremos las figuras que muestren de forma visual el árbol de clasificación. Por ultimo, se mostrará una tabla resumen de las matrices de confusión de cada modelo.

```r
set.seed(seed)
# Modelo sin boosting.
model_tree_1 <- C5.0(train_data[,-1], train_data$ECG_signal, trials = 1)
summary(model_tree_1)
```

```

Call:
C5.0.default(x = train_data[, -1], y = train_data$ECG_signal, trials = 1)


C5.0 [Release 2.07 GPL Edition]     Sun Jan 28 10:26:35 2024
-------------------------------

Class specified by attribute `outcome'

Read 804 cases (46 attributes) from undefined.data

Decision tree:

RTdis <= 0.1667871: NSR (201)
RTdis > 0.1667871:
:...NNTot <= 0.3677419: ARR (221/1)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (136)
        NNTot <= 0.483871:
        :...NNTot <= 0.4451613:
            :...IBISD > 0.0210566: AFF (18)
            :   IBISD <= 0.0210566:
            :   :...SDSD > 0.007752874: CHF (3)
            :       SDSD <= 0.007752874:
```

```
:       :...STdis <= 0.4298038: AFF (4)
:           STdis > 0.4298038: CHF (3)
NNTot > 0.4451613:
:...RRmean > 0.1718247: AFF (6)
    RRmean <= 0.1718247:
    :...NN50 > 0.3783784:
        :...SDRR <= 0.02383286: CHF (8)
        :   SDRR > 0.02383286: AFF (18/1)
        NN50 <= 0.3783784:
        :...NNTot <= 0.4645161: CHF (143/2)
            NNTot > 0.4645161:
            :...PPmean <= 0.1315211: CHF (33/1)
                PPmean > 0.1315211:
                :...SToffdis <= 0.4203458: AFF (6/1)
                    SToffdis > 0.4203458: CHF (4)


Evaluation on training data (804 cases):

	    Decision Tree
	  ----------------
	  Size      Errors

	    14    6( 0.7%)   <<


	   (a)   (b)   (c)   (d)    <-classified as
	  ----  ----  ----  ----
	   186           3          (a): class AFF
	         220                (b): class ARR
	     2     1   191          (c): class CHF
	                     201    (d): class NSR


	Attribute usage:

	100.00% RTdis
	 75.00% NNTot
	 27.11% RRmean
	 26.37% NN50
	  5.35% PPmean
	  3.48% IBISD
	  3.23% SDRR
	  1.24% SToffdis
	  1.24% SDSD
	  0.87% STdis


Time: 0.0 secs
```

```r
plot(model_tree_1)
```




```r
set.seed(seed)
pred_tree_1 <- predict(model_tree_1, test_data)
matriz_confusion_tree_1 <- confusionMatrix(pred_tree_1, test_data$ECG_signal)
matriz_confusion_tree_1
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF 103   1   7   1
       ARR   1  79   2   0
       CHF   7   0  97   0
       NSR   0   0   0  98

Overall Statistics

               Accuracy : 0.952
                 95% CI : (0.9261, 0.9709)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9357

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

```
                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.9279     0.9875     0.9151     0.9899
Specificity              0.9684     0.9905     0.9759     1.0000
Pos Pred Value           0.9196     0.9634     0.9327     1.0000
Neg Pred Value           0.9718     0.9968     0.9692     0.9966
Prevalence               0.2803     0.2020     0.2677     0.2500
Detection Rate           0.2601     0.1995     0.2449     0.2475
Detection Prevalence     0.2828     0.2071     0.2626     0.2475
Balanced Accuracy        0.9482     0.9890     0.9455     0.9949
```

```r
set.seed(seed)
# Modelo aplicando boosting (trials = 10).
model_tree_2 <- C5.0(train_data[,-1], train_data$ECG_signal, trials = 10)
summary(model_tree_2)
```

```
Call:
C5.0.default(x = train_data[, -1], y = train_data$ECG_signal, trials = 10)


C5.0 [Release 2.07 GPL Edition]     Sun Jan 28 10:26:36 2024
-------------------------------

Class specified by attribute `outcome'

Read 804 cases (46 attributes) from undefined.data

-----  Trial 0:  -----

Decision tree:

RTdis <= 0.1667871: NSR (201)
RTdis > 0.1667871:
:...NNTot <= 0.3677419: ARR (221/1)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (136)
        NNTot <= 0.483871:
        :...NNTot <= 0.4451613:
            :...IBISD > 0.0210566: AFF (18)
            :   IBISD <= 0.0210566:
            :   :...SDSD > 0.007752874: CHF (3)
            :       SDSD <= 0.007752874:
            :       :...STdis <= 0.4298038: AFF (4)
            :           STdis > 0.4298038: CHF (3)
            NNTot > 0.4451613:
            :...RRmean > 0.1718247: AFF (6)
                RRmean <= 0.1718247:
                :...NN50 > 0.3783784:
                    :...SDRR <= 0.02383286: CHF (8)
                    :   SDRR > 0.02383286: AFF (18/1)
                    NN50 <= 0.3783784:
                    :...NNTot <= 0.4645161: CHF (143/2)
                        NNTot > 0.4645161:
                        :...PPmean <= 0.1315211: CHF (33/1)
```

```
                            PPmean > 0.1315211:
                            :...SToffdis <= 0.4203458: AFF (6/1)
                                SToffdis > 0.4203458: CHF (4)

-----  Trial 1:  -----

Decision tree:

RTdis <= 0.1667871: NSR (151.1)
RTdis > 0.1667871:
:...IBIM > 0.1452918: ARR (151.9)
    IBIM <= 0.1452918:
    :...STdis > 0.5162575: ARR (15/2.3)
        STdis <= 0.5162575:
        :...NNTot > 0.483871: AFF (101.5)
            NNTot <= 0.483871:
            :...PTdis <= 0.1910204:
                :...QTseg <= 0.03476528: CHF (8.3)
                :   QTseg > 0.03476528: AFF (72.5/3)
                PTdis > 0.1910204:
                :...IBISD <= 0.03364929: CHF (232.1/9.8)
                    IBISD > 0.03364929: AFF (71.6/12)

-----  Trial 2:  -----

Decision tree:

RTdis <= 0.1667871: NSR (114.7)
RTdis > 0.1667871:
:...IBIM > 0.1452918: ARR (115.2)
    IBIM <= 0.1452918:
    :...RTdis > 0.5952742: ARR (14.5/0.6)
        RTdis <= 0.5952742:
        :...NNTot > 0.483871: AFF (83)
            NNTot <= 0.483871:
            :...PonTdis > 0.5545884: AFF (36.3)
                PonTdis <= 0.5545884:
                :...STdis > 0.5656004: AFF (13.6/1.7)
                    STdis <= 0.5656004:
                    :...PTdis <= 0.1910134: AFF (26.4)
                        PTdis > 0.1910134:
                        :...RRmean > 0.1619879: AFF (36.9/6.5)
                            RRmean <= 0.1619879:
                            :...Tseg > 0.7919723: AFF (34.6/8.2)
                                Tseg <= 0.7919723:
                                :...STslope <= 0.4247093: CHF (316.9/38.1)
                                    STslope > 0.4247093: AFF (11.9)

-----  Trial 3:  -----

Decision tree:

RTdis <= 0.1667871: NSR (88.1)
RTdis > 0.1667871:
```

```
:...NNTot <= 0.3677419:
    :...SDRR <= 0.002095731: CHF (19.8)
    :   SDRR > 0.002095731: ARR (114)
    NNTot > 0.3677419:
    :...SDSD <= 0.001280905: CHF (68.5/2.2)
        SDSD > 0.001280905:
        :...Tseg > 0.79299: CHF (25.5/0.4)
            Tseg <= 0.79299:
            :...QToffdis <= 0.1960591: CHF (24.5/0.9)
                QToffdis > 0.1960591:
                :...NN50 > 0.4594595: AFF (126.2/2.6)
                    NN50 <= 0.4594595:
                    :...SDSD <= 0.004591499:
                        :...PonTdis <= 0.2028075: CHF (5/0.4)
                        :   PonTdis > 0.2028075: AFF (136.7/6.6)
                        SDSD > 0.004591499:
                        :...Pseg > 0.6403654: CHF (57.8)
                            Pseg <= 0.6403654:
                            :...RRTot > 0.3607595: CHF (34.5/0.9)
                                RRTot <= 0.3607595:
                                :...QToffdis <= 0.5868583: AFF (89.3/10.1)
                                    QToffdis > 0.5868583: CHF (14.2)

-----  Trial 4:  -----

Decision tree:

RTdis <= 0.1667871: NSR (66.8)
RTdis > 0.1667871:
:...NNTot <= 0.3677419:
    :...Pseg <= 0.0873682: CHF (15.4/0.3)
    :   Pseg > 0.0873682: ARR (86)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (76.6)
        NNTot <= 0.483871:
        :...NNTot <= 0.4451613:
            :...SToffdis > 0.5884368: CHF (3.9)
            :   SToffdis <= 0.5884368:
            :   :...NNTot <= 0.3741935: CHF (4.2)
            :       NNTot > 0.3741935: AFF (83.3/4.5)
            NNTot > 0.4451613:
            :...PTdis <= 0.1910134: AFF (18.9)
                PTdis > 0.1910134:
                :...STseg > 0.6004385: AFF (35.8/1.7)
                    STseg <= 0.6004385:
                    :...NNTot <= 0.4645161:
                        :...IBISD <= 0.03453309: CHF (205.2)
                        :   IBISD > 0.03453309:
                        :   :...Pseg <= 0.5760706: AFF (36.8/6.9)
                        :       Pseg > 0.5760706: CHF (42.5)
                        NNTot > 0.4645161:
                        :...QRseg > 0.5539283: CHF (24.4)
                            QRseg <= 0.5539283:
                            :...PonToffdis <= 0.2519847: CHF (22.2)
```

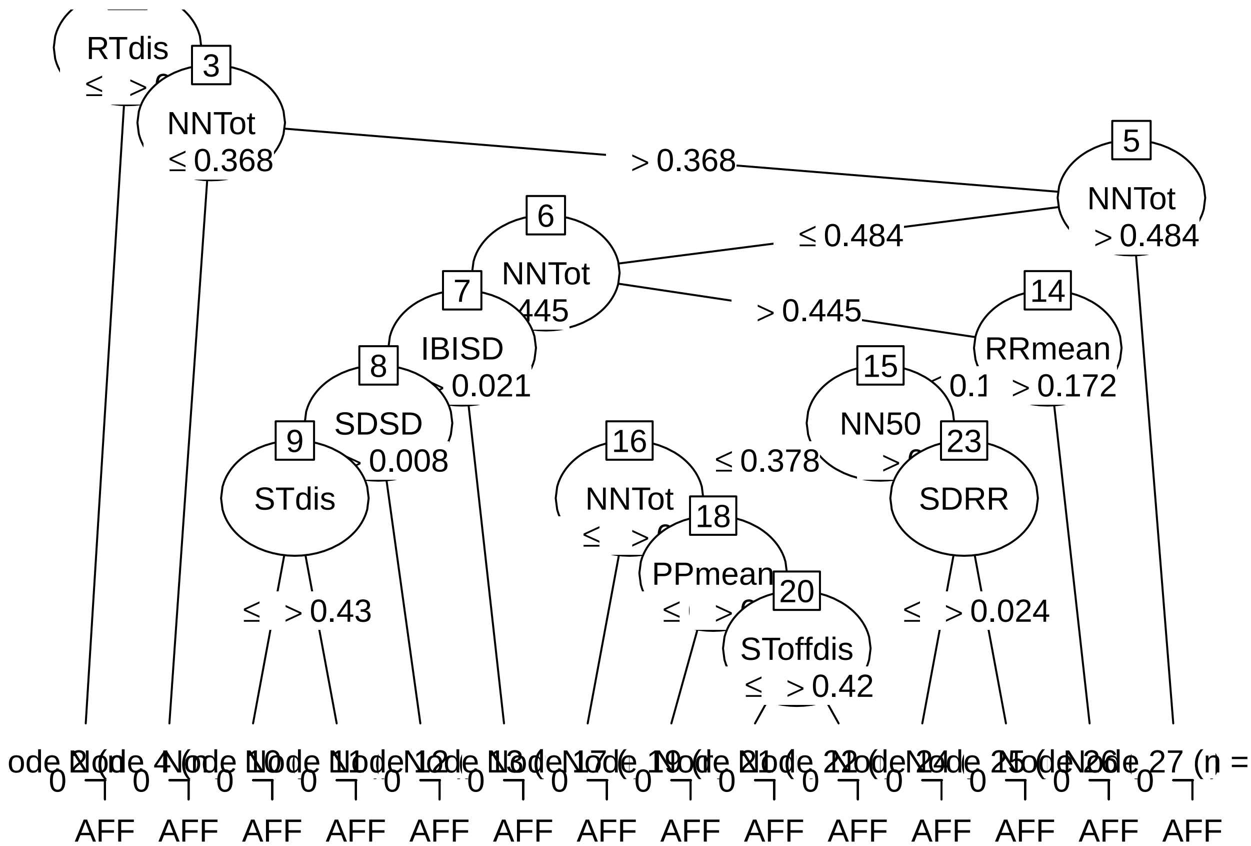
```
                                PonToffdis > 0.2519847:
                                :...QTseg <= 0.1178388: CHF (13.9)
                                    QTseg > 0.1178388: AFF (68.2/11.6)

-----  Trial 5:  -----

Decision tree:

RTdis <= 0.1667871: NSR (50.6)
RTdis > 0.1667871:
:...NNTot <= 0.3677419:
    :...SDRR <= 0.002095731: CHF (11.4)
    :   SDRR > 0.002095731: ARR (79)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (58.1)
        NNTot <= 0.483871:
        :...PTdis > 0.5577416: AFF (18.6)
            PTdis <= 0.5577416:
            :...SDSD > 0.02954469:
                :...NN50 <= 0.3513514: CHF (23.9)
                :   NN50 > 0.3513514: AFF (62)
                SDSD <= 0.02954469:
                :...PTdis <= 0.1910204: AFF (45.2/16.6)
                    PTdis > 0.1910204:
                    :...PPmean > 0.1623282: AFF (27.9/11.5)
                        PPmean <= 0.1623282:
                        :...SDRR <= 0.02482285: CHF (346.5/17.1)
                            SDRR > 0.02482285:
                            :...PonRdis <= 0.3855436: CHF (68.7/10.9)
                                PonRdis > 0.3855436: AFF (12.1/0.3)

-----  Trial 6:  -----

Decision tree:

RTdis <= 0.1667871: NSR (38.9)
RTdis > 0.1667871:
:...NNTot <= 0.3677419: ARR (69.5/8.8)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (44.7)
        NNTot <= 0.483871:
        :...STseg > 0.7408792: AFF (18.1)
            STseg <= 0.7408792:
            :...NN50 > 0.3783784:
                :...SDRR <= 0.03530982: CHF (40.6/4.5)
                :   SDRR > 0.03530982: AFF (68.3)
                NN50 <= 0.3783784:
                :...STslope > 0.4247093: AFF (18.1)
                    STslope <= 0.4247093:
                    :...PonTdis > 0.5528697: AFF (15/0.2)
                        PonTdis <= 0.5528697:
                        :...STslope <= 0.2456336: CHF (403.2/30.9)
                            STslope > 0.2456336:
                            :...QTdis <= 0.2026339: AFF (21.6/0.6)
```

```
                                QTdis > 0.2026339: CHF (66.2/9.1)

-----  Trial 7:  -----

Decision tree:

RTdis <= 0.1667871: NSR (29.9)
RTdis > 0.1667871:
:...IBIM > 0.1452918: ARR (38)
    IBIM <= 0.1452918:
    :...NNTot <= 0.3741935: CHF (28.9/8.6)
        NNTot > 0.3741935:
        :...RRTot <= 0.221519: AFF (69.3)
            RRTot > 0.221519:
            :...SDSD <= 0.001218809: CHF (145.6/1.7)
                SDSD > 0.001218809:
                :...QRdis > 0.5042374: CHF (69.1/4.9)
                    QRdis <= 0.5042374:
                    :...PQdis > 0.589033: CHF (26.6/1.6)
                        PQdis <= 0.589033:
                        :...PQslope > 0.8777664: AFF (83.8/3.4)
                            PQslope <= 0.8777664:
                            :...Tseg <= 0.3603433: CHF (43.1/0.9)
                                Tseg > 0.3603433:
                                :...NNTot <= 0.4451613: AFF (54.4/1.7)
                                    NNTot > 0.4451613:
                                    :...NNTot > 0.4645161:
                                        :...STseg <= 0.4583217: CHF (13.5/1.7)
                                        :   STseg > 0.4583217: AFF (84.7/7.4)
                                        NNTot <= 0.4645161:
                                        :...NN50 > 0.5675676: AFF (10.3)
                                            NN50 <= 0.5675676:
                                            :...STslope <= 0.2575125: CHF (98.2/3.3)
                                                STslope > 0.2575125: AFF (8.8/0.4)

-----  Trial 8:  -----

Decision tree:

SToffdis <= 0.1922232: NSR (22.1/0.1)
SToffdis > 0.1922232:
:...NNTot <= 0.3677419:
    :...SDSD <= 0.001366355: CHF (14)
    :   SDSD > 0.001366355: ARR (93.5)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (86.8/0.8)
        NNTot <= 0.483871:
        :...RRmean > 0.1718247: AFF (46.6)
            RRmean <= 0.1718247:
            :...SDSD <= 0.001303661: CHF (115.1)
                SDSD > 0.001303661:
                :...PTdis > 0.5577416: AFF (18.8)
                    PTdis <= 0.5577416:
                    :...SDSD <= 0.001511497: AFF (24.6)
```

```
                        SDSD > 0.001511497:
                        :...QRdis > 0.4762217: CHF (67.5)
                            QRdis <= 0.4762217:
                            :...pNN50 > 0.3835723: AFF (69.7/7.8)
                                pNN50 <= 0.3835723:
                                :...QRdis > 0.4587311: AFF (16.6)
                                    QRdis <= 0.4587311:
                                    :...NNTot <= 0.4451613: AFF (49/12.4)
                                        NNTot > 0.4451613:
                                        :...PTdis <= 0.1910134: AFF (14.2)
                                            PTdis > 0.1910134: CHF (165.5/13.2)

-----  Trial 9:  -----

Decision tree:

RTdis <= 0.1667871: NSR (93.2)
RTdis > 0.1667871:
:...NNTot <= 0.3677419:
    :...SDSD <= 0.001366355: CHF (10.7)
    :   SDSD > 0.001366355: ARR (71.1)
    NNTot > 0.3677419:
    :...NNTot > 0.483871: AFF (76.4)
        NNTot <= 0.483871:
        :...PonRdis <= 0.1391179: AFF (50.5/1.1)
            PonRdis > 0.1391179:
            :...IBISD <= 0.02059357:
                :...IBIM <= 0.1222248: CHF (268.9/25.2)
                :   IBIM > 0.1222248: AFF (12.7)
                IBISD > 0.02059357:
                :...PonQdis <= 0.1980509: CHF (43.7/1)
                    PonQdis > 0.1980509:
                    :...Pseg > 0.6370179: CHF (35.2/5.8)
                        Pseg <= 0.6370179:
                        :...RToffdis <= 0.2252389: CHF (5.8)
                            RToffdis > 0.2252389:
                            :...Pseg <= 0.3462449: CHF (9.9/1)
                                Pseg > 0.3462449: AFF (125.7/8.1)


Evaluation on training data (804 cases):

Trial       Decision Tree
-----     ----------------
      Size      Errors

   0        14    6( 0.7%)
   1         8   36( 4.5%)
   2        11   40( 5.0%)
   3        13   55( 6.8%)
   4        16   14( 1.7%)
   5        12   27( 3.4%)
   6        11   18( 2.2%)
   7        15   59( 7.3%)
```

```
   8        14   17( 2.1%)
   9        12   21( 2.6%)
boost             0( 0.0%)   <<


       (a)   (b)   (c)   (d)    <-classified as
      ----  ----  ----  ----
       189                      (a): class AFF
             220                (b): class ARR
                   194          (c): class CHF
                         201    (d): class NSR


    Attribute usage:

    100.00% RTdis
    100.00% SToffdis
     75.87% NNTot
     75.00% IBIM
     75.00% SDSD
     52.11% SDRR
     49.88% STdis
     49.50% Pseg
     47.39% RRTot
     46.89% Tseg
     46.89% NN50
     37.94% QToffdis
     36.82% STseg
     35.70% PonTdis
     34.08% QRdis
     30.85% PTdis
     30.72% RRmean
     30.60% PonRdis
     30.60% IBISD
     30.47% STslope
     29.85% PQdis
     27.86% PQslope
     24.25% PPmean
     15.92% pNN50
     11.82% PonQdis
      6.97% RToffdis
      6.34% QRseg
      5.60% QTseg
      4.85% PonToffdis
      4.73% QTdis


Time: 0.2 secs
```

```r
plot(model_tree_2)
```




```r
set.seed(seed)
pred_tree_2 <- predict(model_tree_2, test_data)
matriz_confusion_tree_2 <- confusionMatrix(pred_tree_2, test_data$ECG_signal)
matriz_confusion_tree_2
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF 104   1   2   1
       ARR   1  79   2   0
       CHF   6   0 102   0
       NSR   0   0   0  98

Overall Statistics

               Accuracy : 0.9672
                 95% CI : (0.9445, 0.9824)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.956

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

```
                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.9369     0.9875     0.9623     0.9899
Specificity              0.9860     0.9905     0.9793     1.0000
Pos Pred Value           0.9630     0.9634     0.9444     1.0000
Neg Pred Value           0.9757     0.9968     0.9861     0.9966
Prevalence               0.2803     0.2020     0.2677     0.2500
Detection Rate           0.2626     0.1995     0.2576     0.2475
Detection Prevalence     0.2727     0.2071     0.2727     0.2475
Balanced Accuracy        0.9615     0.9890     0.9708     0.9949
```

Table 7: Comparación de Modelos ('Árbol de clasificación')

	Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
2	Tree with boosting	0.967	0.956	0.033	0.937	0.988	0.962	0.99
1	Tree without boosting	0.952	0.936	0.048	0.928	0.988	0.915	0.99

Si comparamos ambos modelos observamos que el modelo obtenido activando boosting tiene una mayor precisión , el valor kappa es superior en el modelo con boosting y el valor de error es superior en el modelo con boosting .

Los valores de sensibilidad para el modelo con mejor precisión son 93.7% para la clase AFF, 98.8% para la clase ARR, 96.2% para la clase CHF y 99% para la última clase NSR.

Si las características de los modelos de árbol de clasificación son similares, optaremos por elegir la opción más sencilla.

Random Forest

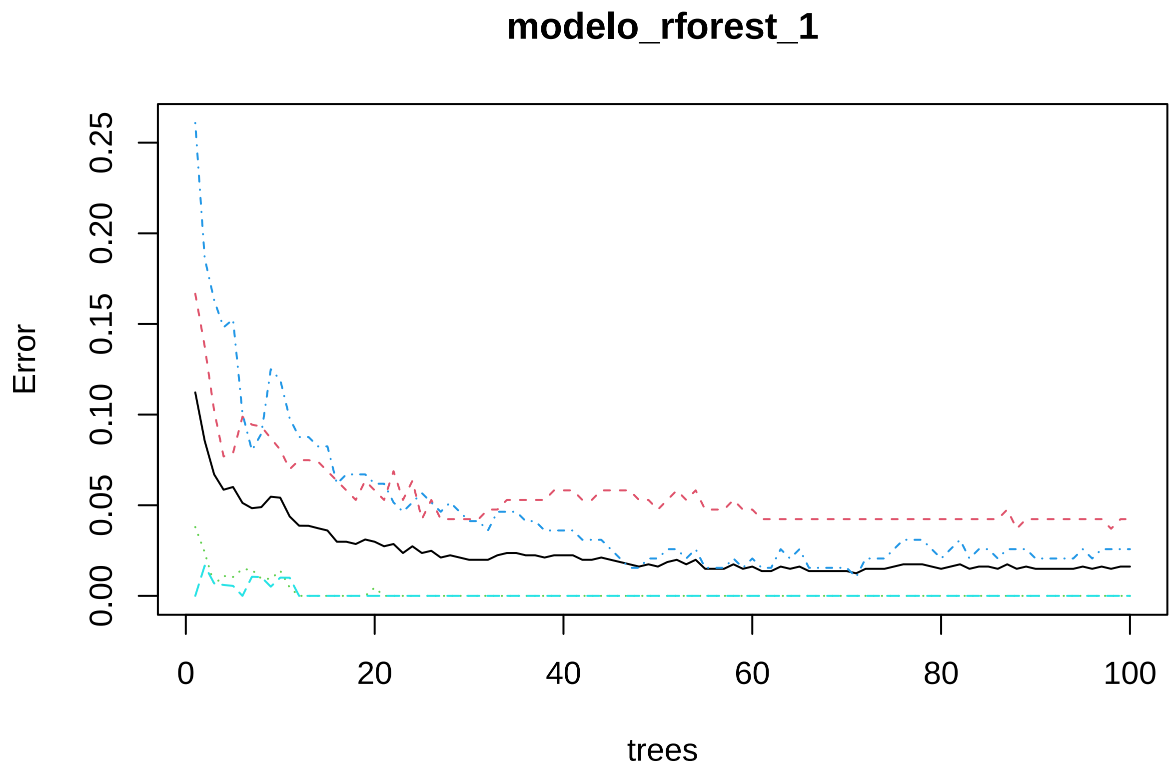
En este último apartado implementaremos el último algoritmo de la práctica evaluable. Se trata del algoritmo **Random Forest**. Para implementar los dos últimos modelos, usaremos la función `randomForest` del paquete *`randomForest`* (Liaw and Wiener 2002). Se presentará de forma gráfica la evolución del error de nuestras clases a medida que se construyen los árboles. Además, se presentará una tabla con las métricas de ambos modelos derivadas de las matrices de confusión.

```
set.seed(seed)

modelo_rforest_1 <- randomForest(ECG_signal ~ ., data = train_data, ntree = 100)
modelo_rforest_1
```

```
Call:
 randomForest(formula = ECG_signal ~ ., data = train_data, ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 6

        OOB estimate of  error rate: 1.62%
Confusion matrix:
    AFF ARR CHF NSR class.error
AFF 181   0   8   0  0.04232804
ARR   0 220   0   0  0.00000000
```

```
CHF    4    1 189    0   0.02577320
NSR    0    0    0 201   0.00000000
```

```r
plot(modelo_rforest_1)
```




```r
set.seed(seed)
pred_rforest_1 <- predict(modelo_rforest_1, newdata = test_data)
matriz_confusion_rforest_1 <- confusionMatrix(pred_rforest_1 , test_data$ECG_signal)
matriz_confusion_rforest_1
```

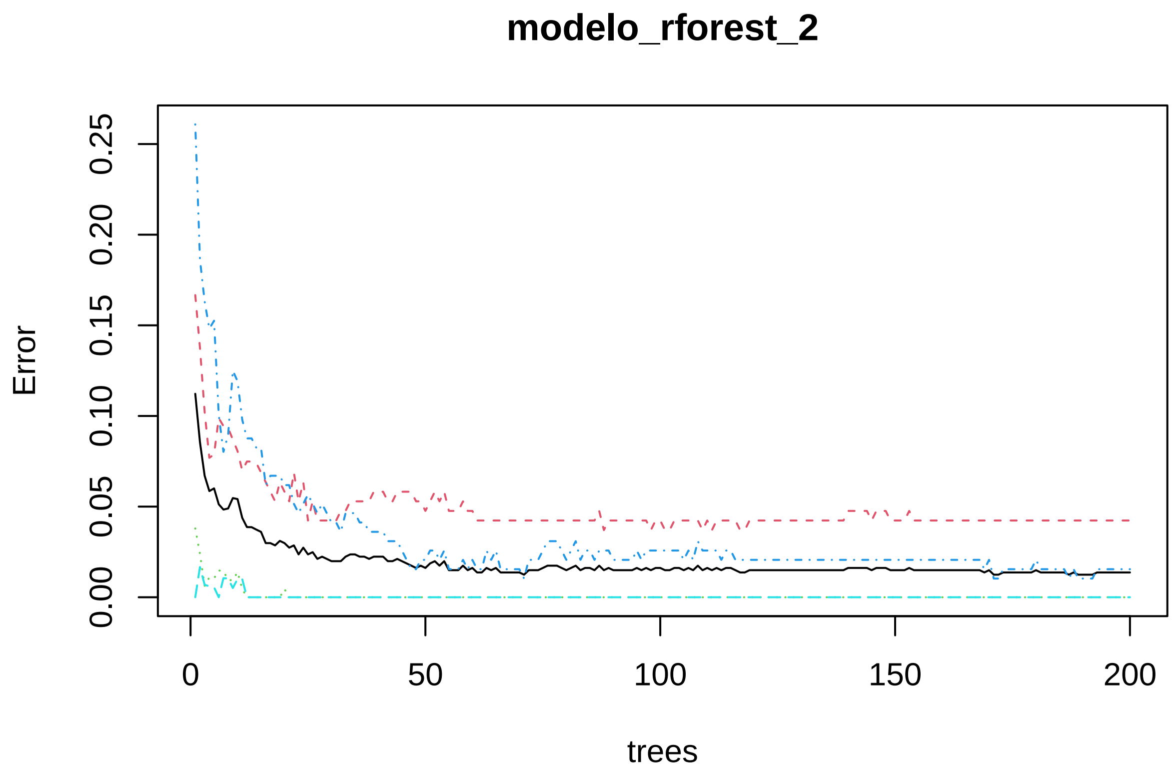
```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF 106   0   1   0
       ARR   0  80   1   0
       CHF   5   0 104   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9823
                 95% CI : (0.9639, 0.9929)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9763
```

```
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity              0.9550     1.0000     0.9811       1.00
Specificity              0.9965     0.9968     0.9828       1.00
Pos Pred Value           0.9907     0.9877     0.9541       1.00
Neg Pred Value           0.9827     1.0000     0.9930       1.00
Prevalence               0.2803     0.2020     0.2677       0.25
Detection Rate           0.2677     0.2020     0.2626       0.25
Detection Prevalence     0.2702     0.2045     0.2753       0.25
Balanced Accuracy        0.9757     0.9984     0.9819       1.00
```

```r
set.seed(seed)

modelo_rforest_2 <- randomForest(ECG_signal ~ ., data = train_data, ntree = 200)
modelo_rforest_2
```

```
Call:
 randomForest(formula = ECG_signal ~ ., data = train_data, ntree = 200)
               Type of random forest: classification
                     Number of trees: 200
No. of variables tried at each split: 6

        OOB estimate of  error rate: 1.37%
Confusion matrix:
    AFF ARR CHF NSR class.error
AFF 181   0   8   0  0.04232804
ARR   0 220   0   0  0.00000000
CHF   2   1 191   0  0.01546392
NSR   0   0   0 201  0.00000000
```

```r
plot(modelo_rforest_2)
```




```
set.seed(seed)
pred_rforest_2 <- predict(modelo_rforest_2, newdata = test_data)
matriz_confusion_rforest_2 <- confusionMatrix(pred_rforest_2 , test_data$ECG_signal)
matriz_confusion_rforest_2
```

```
Confusion Matrix and Statistics

          Reference
Prediction AFF ARR CHF NSR
       AFF 105   0   1   0
       ARR   0  80   1   0
       CHF   6   0 104   0
       NSR   0   0   0  99

Overall Statistics

               Accuracy : 0.9798
                 95% CI : (0.9606, 0.9912)
    No Information Rate : 0.2803
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9729

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

```
                      Class: AFF Class: ARR Class: CHF Class: NSR
Sensitivity               0.9459     1.0000     0.9811       1.00
Specificity               0.9965     0.9968     0.9793       1.00
Pos Pred Value            0.9906     0.9877     0.9455       1.00
Neg Pred Value            0.9793     1.0000     0.9930       1.00
Prevalence                0.2803     0.2020     0.2677       0.25
Detection Rate            0.2652     0.2020     0.2626       0.25
Detection Prevalence      0.2677     0.2045     0.2778       0.25
Balanced Accuracy         0.9712     0.9984     0.9802       1.00
```

Table 8: Comparación de Modelos ('Ramdon Forest')

Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
rForest n=100	0.982	0.976	0.018	0.955	1	0.981	1
rForest n=200	0.980	0.973	0.020	0.946	1	0.981	1

Comparando los dos ultimos modelos observamos que el modelo obtenido con un número de árboles igual a 100 tiene una mayor precisión , el valor kappa también es superior en el modelo con un número de árboles igual a 100 y el valor de error es superior en el modelo con un número de árboles igual a 100 .

Los valores de sensibilidad para el modelo con mejor precisión son 95.5% para la clase AFF, 100% para la clase ARR, 98.1% para la clase CHF y 100% para la última clase NSR.

Como en los modelos anteriores, si obtenemos métricas muy similares, volveremos a elegir el más sencillo.

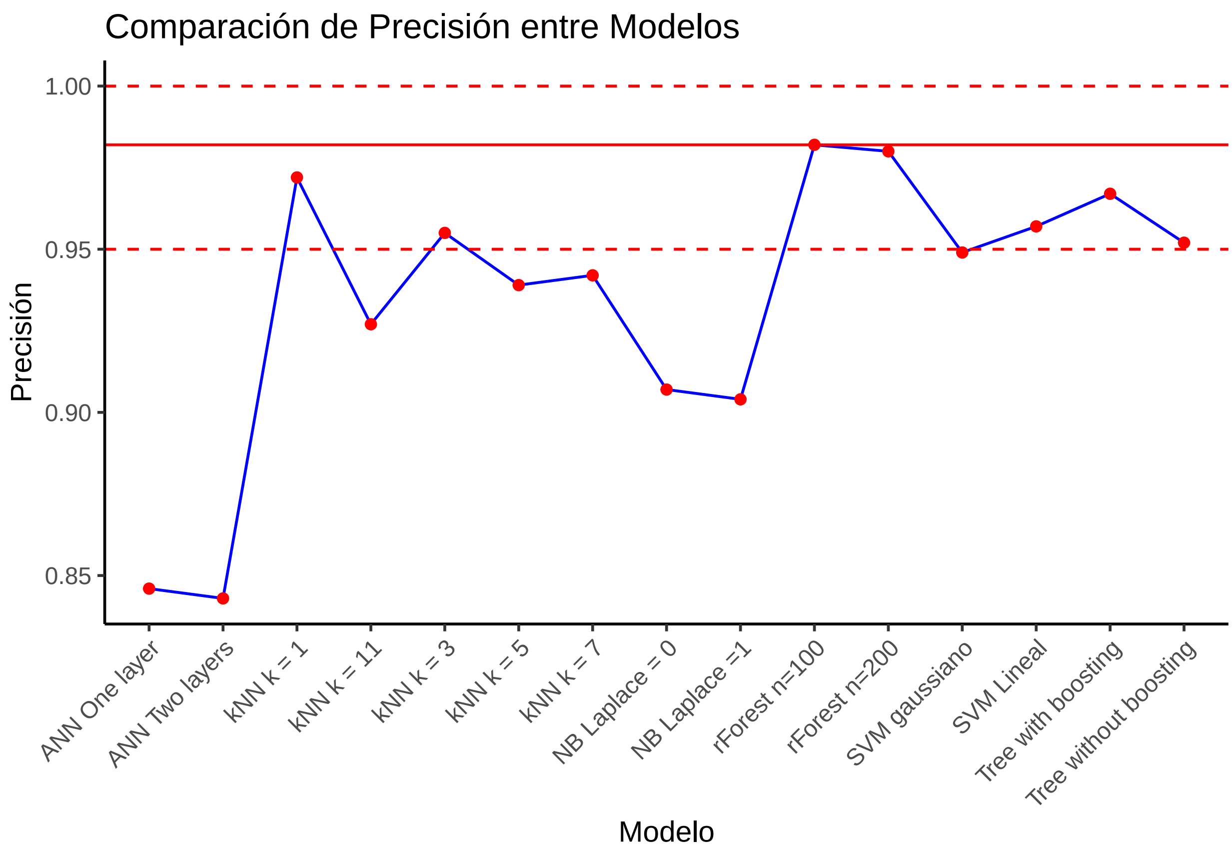
Discusión y Conclusión

En la siguiente tabla se presentan todos los modelos implementados en esta actividad, tenemos 15 modelos. Se incluyen las métricas de precisión (Accuracy), el valor Kappa, el ratio de error (Error_rate) y la sensibilidad de cada clase de estudio. Cada modelo se ha ordenado por su valor de precisión de mayor a menor.

Table 9: Modelos

	Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
15	rForest n=100	0.982	0.976	0.018	0.955	1.000	0.981	1.00
25	rForest n=200	0.980	0.973	0.020	0.946	1.000	0.981	1.00
1	kNN k = 1	0.972	0.963	0.028	0.919	1.000	0.981	1.00
24	Tree with boosting	0.967	0.956	0.033	0.937	0.988	0.962	0.99
13	SVM Lineal	0.957	0.943	0.043	0.865	1.000	0.991	0.99
2	kNN k = 3	0.955	0.939	0.045	0.865	1.000	0.972	1.00
14	Tree without boosting	0.952	0.936	0.048	0.928	0.988	0.915	0.99
23	SVM gaussiano	0.949	0.932	0.051	0.865	1.000	0.962	0.99
4	kNN k = 7	0.942	0.922	0.058	0.829	1.000	0.962	1.00
3	kNN k = 5	0.939	0.919	0.061	0.820	1.000	0.962	1.00
5	kNN k = 11	0.927	0.902	0.073	0.829	1.000	0.906	1.00
11	NB Laplace = 0	0.907	0.875	0.093	0.892	0.900	0.858	0.98
21	NB Laplace =1	0.904	0.871	0.096	0.892	0.900	0.849	0.98

	Modelo	Accuracy	Kappa	Error_rate	Sensitivity_AFF	Sensitivity_ARR	Sensitivity_CHF	Sensitivity_NSR
12	ANN One layer	0.846	0.794	0.154	0.622	1.000	0.821	1.00
22	ANN Two layers	0.843	0.791	0.157	0.568	1.000	0.868	1.00



Para la clasificación de dolencias cardiacas hemos explorado distintos tipos de algoritmos. En la tabla anterior, podemos comparar las métricas de cada uno. Los mejores tres modelos se corresponden, de más preciso a menos, con rForest n=100, rForest n=200 y kNN k = 1. La precisión de los modelos varía desde 98.2% hasta 84.3%, lo que nos da una diferencia de un 13.9% entre el primero y el último. Sin embargo, entre el primer modelo y el segundo tenemos una diferencia del 0.2% y una diferencia con el tercero del 1%.

El modelo rForest n=100 tiene las mejores métricas en cuanto a precisión (98.2%), valor Kappa (97.6%), y ratio de error 1.8%. Los valores de sensibilidad para el modelo rForest n=100 con la mejor precisión (98.2%) son 95.5% para la clase AFF, 100% para la clase ARR, 98.1% para la clase CHF, y 100% para la última clase NSR.

A la hora de elegir un modelo, tenemos que tener en cuenta las métricas. Aun así, si nuestros modelos muestran valores muy similares, podemos establecer un valor de corte (cutoff). En nuestro caso, vamos a seleccionar los modelos que tengan una precisión del 95% o más:

Table 10: Modelos Seleccionados

x
rForest n=100
rForest n=200
kNN k = 1

x
Tree with boosting
SVM Lineal
kNN k = 3
Tree without boosting

De los modelos seleccionados, los dos primeros destacan por sus métricas. Como resultado final de esta práctica evaluable, elegimos estos modelos, rForest n=100 y rForest n=200, como los mejores para clasificar las dolencias cardiacas. Si los modelos coinciden en que son del mismo tipo de algoritmo, nos quedaremos únicamente con el más sencillo. De forma adicional, podríamos incluir el tercer tipo de modelo (kNN k = 1).

Referencias

Allaire, JJ, and François Chollet. 2023. *Keras: R Interface to 'Keras'*. https://CRAN.R-project.org/package=keras.

Karatzoglou, Alexandros, Alex Smola, and Kurt Hornik. 2023. *Kernlab: Kernel-Based Machine Learning Lab*. https://CRAN.R-project.org/package=kernlab.

Kuhn, Max, and Ross Quinlan. 2023. *C50: C5.0 Decision Trees and Rule-Based Models*. https://CRAN.R-project.org/package=C50.

Kuhn, and Max. 2008. "Building Predictive Models in r Using the Caret Package." *Journal of Statistical Software* 28 (5): 1–26. https://doi.org/10.18637/jss.v028.i05.

Lantz, Brett. 2019. *Machine Learning with r: Expert Techniques for Predictive Modeling*. Packt publishing ltd.

Liaw, Andy, and Matthew Wiener. 2002. "Classification and Regression by randomForest." *R News* 2 (3): 18–22. https://CRAN.R-project.org/doc/Rnews/.

Meyer, David, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. 2023. *E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. https://CRAN.R-project.org/package=e1071.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with s*. Fourth. New York: Springer. https://www.stats.ox.ac.uk/pub/MASS4/.