



UNIVERSIDAD CARLOS III DE MADRID

ANÁLISIS MULTIVARIANTE, GRADO EN ESTADÍSTICA Y EMPRESA

Práctica I: Inferencia multivariante aplicada a datos de Airbnb en Madrid

Jorge Salas y Marc Pastor

Índice

1	Objetivos	3
2	Problema	3
3	Datos	3
4	Desarrollo de la práctica	3
4.1	Carga de los datos	3
4.2	4.2. EDA en R	7
4.2.1	4.2.1. EDA con <code>skimr</code>	7
4.2.2	4.2.2. EDA con <code>DataExplorer</code>	9
4.3	4.3. EDA en <code>Python</code>	10
4.3.1	4.3.1. Estructura del data-set	10
4.3.2	4.3.2. Resumen Estadístico Descriptivo Básico	14
4.3.3	4.3.3. Análisis gráfico general	18
4.3.4	4.3.4. Análisis de la relación entre los predictores categoricos y la respuesta	24
4.3.5	4.3.5. Análisis de la relación entre los predictores cuantitativos y la respuesta	34
4.4	4.4. Arboles de clasificación en R	42
4.4.1	4.4.1. Algoritmo <code>rpart</code> con R	42
4.4.2	4.4.2. Algoritmo C5.0 con R	48
4.4.3	4.4.3. Algoritmo CART en R con <code>mlr3</code>	58
4.4.4	4.4.4. Algoritmo C5.0 en R con <code>mlr3</code>	61
4.4.5	4.4.5. Comparación entre R y <code>mlr3</code>	63
4.4.6	4.4.6. Comparación de resultados	64
4.5	4.5. Arboles de clasificación en <code>Python</code>	65
4.5.1	4.5.1. Arboles de clasificación: <i>teoría</i>	65
4.5.2	Definicion formal de los arboles de clasificación:	66

1 Objetivos

- Conocer un contexto de aplicación real.
- Ejercitarnos en el análisis de datos y la implementación de algoritmos de clasificación, para adquirir criterio en la aplicación de los mismos.
- Evaluar las capacidades de R y mlr para la implementación.

2 Problema

- Resolver un problema de clasificación para el diagnóstico de pacientes hepáticos.

3 Datos

- Usaremos el conjunto de datos “Indian Liver Patient Dataset”: Los pacientes con enfermedades del hígado han ido aumentando continuamente debido al consumo excesivo de alcohol, inhalación de gases nocivos, ingesta de alimentos contaminados, encurtidos y drogas.
- Este conjunto de datos se utilizó para evaluar los algoritmos de predicción en un esfuerzo por reducir la carga para los médicos. Este conjunto de datos contiene 416 registros de pacientes hepáticos y 167 registros de pacientes no hepáticos recopilados en el noreste de Andhra Pradesh, India.
- La variable de respuesta es “diseased” (personas que tienen enfermedad del hígado)
- El data set encuentra en la librería “mlr3data”. `data(“ilpd”, package = “mlr3data”)`

4 Desarrollo de la práctica

Vamos a desarrollar esta práctica en los lenguajes de programación R y Python simultáneamente.

En general lo haremos a través el lenguaje Python utilizando un paquete llamado `rpy2` que permite ejecutar código R desde Python

4.1 Carga de los datos

Empezamos la práctica cargando los datos, tanto en R como en Python :

Importamos en Python la librería `rpy2` que nos será esencial para trabajar con R y Python simultáneamente desde el mismo entorno:

```
import warnings
warnings.filterwarnings("ignore")
```

```
import rpy2

%load_ext rpy2.ipython
```

```
%%R

# install.packages("mlr3data")
# install.packages("mlr3")
```

NULL

Cargamos los datos en R

```
%%R

data("ilpd", package = "mlr3data")

head(ilpd,5)
```

	age	gender	total_bilirubin	direct_bilirubin	alkaline_phosphatase
1	65	Female	0.7	0.1	187
2	62	Male	10.9	5.5	699
3	62	Male	7.3	4.1	490
4	58	Male	1.0	0.4	182
5	72	Male	3.9	2.0	195

	alanine_transaminase	aspartate_transaminase	total_protein	albumin
1	16	18	6.8	3.3
2	64	100	7.5	3.2
3	60	68	7.0	3.3
4	14	20	6.8	3.4
5	27	59	7.3	2.4

	albumin_globulin_ratio	diseased
1	0.90	yes
2	0.74	yes
3	0.89	yes
4	1.00	yes
5	0.40	yes

Cargamos los datos en Python

```
import pandas as pd

Data_Python = pd.read_csv('indian_liver_patient.csv')
```

```
Data_Python = Data_Python.rename({'Dataset': 'Diseased'}, axis=1)
```

```
Data_Python.head()
```

```
Age
Gender
Total_Bilirubin
Direct_Bilirubin
Alkaline_Phosphotase
Alamine_Aminotransferase
Aspartate_Aminotransferase
Total_Protiens
Albumin
Albumin_and_Globulin_Ratio
Diseased
0
65
Female
0.7
0.1
187
16
18
6.8
3.3
0.90
1
1
62
Male
10.9
5.5
699
64
100
```

7.5
3.2
0.74
1
2
62
Male
7.3
4.1
490
60
68
7.0
3.3
0.89
1
3
58
Male
1.0
0.4
182
14
20
6.8
3.4
1.00
1
4
72
Male
3.9
2.0
195
27

59

7.3

2.4

0.40

1

Describiremos cada una de las variables:

- age: edad del paciente. A los pacientes que exceden 89 son listados con la edad 90
- gender: género del paciente.
- total_bilirubin: Total de bilirubina.
- direct_bilirubin: Bilirubina directa.
- alkaline_phosphatase: Fosfatasa alcalina.
- alanine_transaminase: alanina aminotransferasa o transaminasa glutámico pirúvica.
- aspartate_transaminase: aspartato aminotransferasa.
- total_protein: proteínas totales.
- albumin: albúmina.
- albumin_globulin_ratio: albúmina y globulina ratio.
- diseased: Si tienen (1) o no (2) enfermedad en el hígado.

Ahora que ya tenemos cargados los datos y hemos visto la apariencia de los mismo procedemos a hacer un EDA (exploratory data analysis).

4.2 4.2. EDA en R

4.2.1 4.2.1. EDA con skimr

Haremos el EDA con la librería `skimr`, como se pide en el enunciado de la práctica.

```
%R
# install.packages('skimr')

library(skimr) # Cargamos librería

skim(ilpd) # EDA con librería
```

```
-- Data Summary -----
                                Values
Name                           ilpd
Number of rows                 583
Number of columns              11
-----
Column type frequency:
  factor                2
  numeric               9
```

```
-----
Group variables          None
```

```
-- Variable type: factor -----
skim_variable n_missing complete_rate ordered n_unique top_counts
1 gender      0          1 FALSE          2 Mal: 441, Fem: 142
2 diseased    0          1 FALSE          2 yes: 416, no: 167
```

```
-- Variable type: numeric -----
skim_variable      n_missing complete_rate      mean      sd      p0      p25
1 age              0          1  44.7      16.2      4      33
2 total_bilirubin  0          1   3.30      6.21     0.4     0.8
3 direct_bilirubin 0          1   1.49      2.81     0.1     0.2
4 alkaline_phosphatase 0          1 291.      243.     63     176.
5 alanine_transaminase 0          1  80.7     183.     10      23
6 aspartate_transaminase 0          1 110.      289.     10      25
7 total_protein    0          1   6.48      1.09     2.7     5.8
8 albumin          0          1   3.14      0.796    0.9     2.6
9 albumin_globulin_ratio 0          1   0.947     0.318    0.3     0.7
      p50      p75      p100 hist
1  45        58        90  <U+2582><U+2586><U+2587><U+2585><U+2581>
2   1         2.6       75  <U+2587><U+2581><U+2581><U+2581><U+2581>
3  0.3        1.3      19.7 <U+2587><U+2581><U+2581><U+2581><U+2581>
4 208        298      2110 <U+2587><U+2581><U+2581><U+2581><U+2581>
5  35        60.5     2000 <U+2587><U+2581><U+2581><U+2581><U+2581>
6  42         87      4929 <U+2587><U+2581><U+2581><U+2581><U+2581>
7   6.6       7.2       9.6 <U+2581><U+2582><U+2587><U+2587><U+2581>
8   3.1       3.8       5.5 <U+2581><U+2585><U+2587><U+2586><U+2581>
9  0.947      1.1       2.8 <U+2586><U+2587><U+2582><U+2581><U+2581>
```

También haremos uso de la función `str()` que nos da la estructura de nuestro dataset.

```
%%R
```

```
str(ilpd) # Analizamos la estructura de los datos
```

```
'data.frame':  583 obs. of  11 variables:
 $ age          : int  65 62 62 58 72 46 26 29 17 55 ...
 $ gender       : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 2 1 1 2 2 ...
 $ total_bilirubin : num  0.7 10.9 7.3 1 3.9 1.8 0.9 0.9 0.9 0.7 ...
 $ direct_bilirubin : num  0.1 5.5 4.1 0.4 2 0.7 0.2 0.3 0.3 0.2 ...
 $ alkaline_phosphatase : int  187 699 490 182 195 208 154 202 202 290 ...
 $ alanine_transaminase : int  16 64 60 14 27 19 16 14 22 53 ...
 $ aspartate_transaminase: int  18 100 68 20 59 14 12 11 19 58 ...
 $ total_protein    : num  6.8 7.5 7 6.8 7.3 7.6 7 6.7 7.4 6.8 ...
 $ albumin          : num  3.3 3.2 3.3 3.4 2.4 4.4 3.5 3.6 4.1 3.4 ...
 $ albumin_globulin_ratio: num  0.9 0.74 0.89 1 0.4 1.3 1 1.1 1.2 1 ...
 $ diseased        : Factor w/ 2 levels "yes","no": 1 1 1 1 1 1 1 1 2 1 ...
```


Claramente podemos ver como con skimr, las variables que son numéricas las considera todas numéricas, mientras que la función str nos especifica las variables numéricas en si son variables que toman valores reales o enteros.

4.2.2. EDA con DataExplorer

Por último sacaremos un reporte con la librería DataExplorer:

```
%%R
# install.packages('DataExplorer')
# DataExplorer::create_report(ilpd,y="diseased")
```

NULL

Se genera un reporte en HTML que puede ser abierto en el navegador.

Aquí esta la versión en PDF de dicho reporte:

<https://github.com/FabioScielzoOrtiz/Estadistica4all.github.io/blob/main/Notebooks/Aprendizaje%20Automatico/Data%20Profiling%20Report.pdf>

Aunque hemos visto el número de valores ausentes en la salida que nos da skimr, esto se puede hacer a mano como sigue:

```
%%R
library(tidyverse)

ilpd %>% map_dbl(.f = function(x){sum(is.na(x))}) # Número de missing
↪ values
```

age	gender	total_bilirubin
0	0	0
direct_bilirubin	alkaline_phosphatase	alanine_transaminase
0	0	0
aspartate_transaminase	total_protein	albumin
0	0	0
albumin_globulin_ratio	diseased	
0	0	

Podemos sacar las siguientes conclusiones del EDA anterior: - Se dispone de 583 instancias y 11 variables (2 de tipo factor, 5 de tipo numérico y 4 enteras) - No hay ausencia de valores por lo que no habrá que eliminar instancias o al menos los modelos no se verán dificultados por los mismos. - La variable respuesta es diseased que es una de las variables tipo factor que puede tomar valores “yes” o “no”. Se puede ver como esta variable está un poco desbalanceada ya que tenemos muchas más observaciones con valor “yes”(416

observaciones) que con valor “no”(167 observaciones). - En cuanto a las correlaciones se puede ver que son relativamente altas entre los pares que tienen que ver con sustancias similares, como bilirubina directa y bilirubina total. Con respecto a la variable respuesta podemos ver como hay relaciones directas con el resto de variables y una correlacion similar en torno a 0.7.

4.3 4.3. EDA en Python

Ahora vamos a realizar un EDA del data-set pero usando Python

El EDA (Exploratory Data Analysis) en lineas generales va a consistir en:

- Analizar estructura del data-set que tenemos (dimensiones, tipo de variables, valores faltantes, etc)
- Cálculo de estadísticos básicos para cada variable
- Generación de gráficos que aporten información relevante (histogramas, diagramas de barras, scatter plots, box plots, etc)
- Análisis de relaciones entre los predictores y la respuesta.

4.3.1 4.3.1. Estructura del data-set

```
import warnings
warnings.filterwarnings("ignore")
```

```
Data_Python.shape
```

```
(583, 11)
```

Tenemos un data-set con 11 variables y 583 observaciones.

Las variables son:

- 10 predictores (age , gender , Total_Bilirubin , Direct_Bilirubin , Alkaline_Phosphotase , Alamine_Aminotransferase , Aspartate_Aminotransferase , Total_Protiens , Albumin , Albumin_and_Globulin_Ratio)
- 1 respuesta (Diseased)

Las variables **categoricas** del data-set son:

- Diseased y Gender (*binarias*)

Las variables **cuantitativas** del data-set son:

- age , Alkaline_Phosphotase, Alamine_Aminotransferase , Aspartate_Aminotransferase (*discretas*) y Total_Bilirubin , Direct_Bilirubin, Total_Protiens, Albumin, Albumin_and_Globulin_Ratio (*continuas*)

Con el siguiente código podemos ver el tipo de cada una de las variables en Python (que podría no coincidir con el descrito anteriormente, en su caso habría que modificarlo.)

```
Data_Python.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Age                                    583 non-null    int64
1   Gender                                583 non-null    object
2   Total_Bilirubin                       583 non-null    float64
3   Direct_Bilirubin                      583 non-null    float64
4   Alkaline_Phosphotase                  583 non-null    int64
5   Alamine_Aminotransferase              583 non-null    int64
6   Aspartate_Aminotransferase             583 non-null    int64
7   Total_Protiens                        583 non-null    float64
8   Albumin                              583 non-null    float64
9   Albumin_and_Globulin_Ratio            583 non-null    float64
10  Diseased                              583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

En este caso el tipo en Python es correcto para todas las variables salvo para la respuesta (Diseased) ya que Python la considera entera (cuantitativa discreta: int64) cuando realmente es categórica binaria, por ello transformamos su tipo de int64 a object (el tipo clásico de las variables categóricas en Python).

```
Data_Python['Diseased'] = Data_Python['Diseased'].astype('object')
```

Comprobamos que los cambios se han producido correctamente:

```
Data_Python.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Age                                    583 non-null    int64
1   Gender                                583 non-null    object
2   Total_Bilirubin                       583 non-null    float64
3   Direct_Bilirubin                      583 non-null    float64
4   Alkaline_Phosphotase                  583 non-null    int64
5   Alamine_Aminotransferase              583 non-null    int64
6   Aspartate_Aminotransferase             583 non-null    int64
```

```

7   Total_Protiens          583 non-null    float64
8   Albumin                 583 non-null    float64
9   Albumin_and_Globulin_Ratio 583 non-null    float64
10  Diseased                583 non-null    object
dtypes: float64(5), int64(4), object(2)
memory usage: 50.2+ KB

```

Ahora vamos a ver si existe algún valor nulo en el data-set:

```
Data_Python.isnull().sum()
```

```

Age                0
Gender             0
Total_Bilirubin    0
Direct_Bilirubin   0
Alkaline_Phosphotase 0
Alamine_Aminotransferase 0
Aspartate_Aminotransferase 0
Total_Protiens     0
Albumin            0
Albumin_and_Globulin_Ratio 0
Diseased           0
dtype: int64

```

Ninguna de las variables tiene valores faltantes (nulos).

Ahora vamos a ver cual es el rango de las variables categoricas, y posteriormete lo codificaremos en formato estandar {0,1,2,...} , si es que no lo están ya.

```
Data_Python['Gender'].unique()
```

```
array(['Female', 'Male'], dtype=object)
```

```
Data_Python['Diseased'].unique()
```

```
array([1, 2], dtype=object)
```

Vamos a codificar en formato estandar la variable **Gender** tal que: **Female=0** , **Male=1** , y la variable **Diseased** tal que: **1=0** , **2=1**

Para ello vamos a apoyarnos en la libreria **sklearn**, que posteriormente volverá a ser usada. Se podría hacer esto de otras formas, como con un bucle for, pero en casos en los que el número de categorias es alto, la opción aportada por **sklearn** es bastante más eficiente que un bucle.

```
from sklearn.preprocessing import OrdinalEncoder

ord_enc = OrdinalEncoder()
```

```
Data_Python['Gender'] = ord_enc.fit_transform(Data_Python[['Gender']])
Data_Python['Diseased'] = ord_enc.fit_transform(Data_Python[['Diseased']])
```

Comprobamos que los cambios se han realizado correctamente:

```
Data_Python['Gender'].unique()
```

```
array([0., 1.])
```

```
Data_Python['Diseased'].unique()
```

```
array([0., 1.])
```

Pero cuidado, tras realizar estos cambios tambien se cambia en Python el tipo de las variables codificadas a 'float64', que es un tipo cuantitativo (continuo), por lo que debemos volver a fijar el tipo de Diseased y Gender como 'object' (ya que son categoricas).

```
Data_Python.dtypes
```

```
Age                int64
Gender             float64
Total_Bilirubin    float64
Direct_Bilirubin   float64
Alkaline_Phosphotase  int64
Alamine_Aminotransferase  int64
Aspartate_Aminotransferase int64
Total_Protiens     float64
Albumin            float64
Albumin_and_Globulin_Ratio float64
Diseased           float64
dtype: object
```

```
Data_Python['Diseased'] = Data_Python['Diseased'].astype('object')
Data_Python['Gender'] = Data_Python['Gender'].astype('object')
```

Verificamos que se han realizado correctamente los cambios:

```
Data_Python.dtypes
```

```
Age                int64
Gender             object
Total_Bilirubin    float64
Direct_Bilirubin   float64
Alkaline_Phosphotase  int64
Alamine_Aminotransferase  int64
Aspartate_Aminotransferase  int64
Total_Protiens     float64
Albumin            float64
Albumin_and_Globulin_Ratio  float64
Diseased           object
dtype: object
```

4.3.2 4.3.2. Resumen Estadístico Descriptivo Básico

Ahora vamos a hacer una descripción estadística básica de las variables del data-set:

```
Data_Python.describe(include='all') # include='all' para dar un
→ tratamiento diferente a las categoricas que a las cuantitativas
```

```
Age
Gender
Total_Bilirubin
Direct_Bilirubin
Alkaline_Phosphotase
Alamine_Aminotransferase
Aspartate_Aminotransferase
Total_Protiens
Albumin
Albumin_and_Globulin_Ratio
Diseased
count
583.000000
583.0
583.000000
583.000000
583.000000
583.000000
```

583.000000
583.000000
583.000000
583.000000
583.0
unique
NaN
2.0
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
2.0
top
NaN
1.0
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
NaN
0.0
freq
NaN
441.0
NaN
NaN
NaN

NaN
NaN
NaN
NaN
NaN
416.0
mean
44.746141
NaN
3.298799
1.486106
290.576329
80.713551
109.910806
6.483190
3.141852
0.947064
NaN
std
16.189833
NaN
6.209522
2.808498
242.937989
182.620356
288.918529
1.085451
0.795519
0.318492
NaN
min
4.000000
NaN
0.400000
0.100000

63.000000
10.000000
10.000000
2.700000
0.900000
0.300000
NaN
25%
33.000000
NaN
0.800000
0.200000
175.500000
23.000000
25.000000
5.800000
2.600000
0.700000
NaN
50%
45.000000
NaN
1.000000
0.300000
208.000000
35.000000
42.000000
6.600000
3.100000
0.947064
NaN
75%
58.000000
NaN
2.600000

1.300000
298.000000
60.500000
87.000000
7.200000
3.800000
1.100000
NaN
max
90.000000
NaN
75.000000
19.700000
2110.000000
2000.000000
4929.000000
9.600000
5.500000
2.800000
NaN

4.3.3. Análisis gráfico general

```
import numpy as np

import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
```

```
fig, axs = plt.subplots(3, 3, figsize=(13, 13))

p1 = sns.histplot(data=Data_Python, x="Total_Bilirubin",
    ↳ stat="proportion", bins=15, color="skyblue", ax=axs[0, 0])
p1.set_xticks( range(int(Data_Python['Total_Bilirubin'].min()),
    ↳ int(Data_Python['Total_Bilirubin'].max()) , 10) )
p1.set_yticks( np.arange(0, 1, 0.1) )
```

```

p2 = sns.histplot(data=Data_Python, x="Direct_Bilirubin",
    ↪ stat="proportion", bins=15, color="olive", ax=axes[0, 1])
p2.axes.set(xlabel='Direct_Bilirubin', ylabel=' ')
p2.set_xticks( range(int(Data_Python['Direct_Bilirubin'].min()),
    ↪ int(Data_Python['Direct_Bilirubin'].max()) , 3) )
p2.set_yticks( np.arange(0, 1, 0.1) )

p3 = sns.histplot(data=Data_Python, x="Alkaline_Phosphotase",
    ↪ stat="proportion", bins=15, color="blue", ax=axes[0, 2])
p3.axes.set(xlabel='Alkaline_Phosphotase', ylabel=' ')
p3.set_xticks( range(int(Data_Python['Alkaline_Phosphotase'].min()),
    ↪ int(Data_Python['Alkaline_Phosphotase'].max()) , 320) )
p3.set_yticks( np.arange(0, 1, 0.1) )

p4 = sns.histplot(data=Data_Python, x="Alamine_Aminotransferase",
    ↪ stat="proportion", bins=15, color="teal", ax=axes[1, 0])
p4.axes.set(xlabel='Alamine_Aminotransferase', ylabel=' ')
p4.set_xticks( range(int(Data_Python['Alamine_Aminotransferase'].min()),
    ↪ int(Data_Python['Alamine_Aminotransferase'].max()) , 300) )
p4.set_yticks( np.arange(0, 1, 0.1) )

p5 = sns.histplot(data=Data_Python, x="Aspartate_Aminotransferase",
    ↪ stat="proportion", bins=15, color="purple", ax=axes[1, 1])
p5.axes.set(xlabel='Aspartate_Aminotransferase', ylabel=' ')
p5.set_xticks( range(int(Data_Python['Aspartate_Aminotransferase'].min()),
    ↪ , int(Data_Python['Aspartate_Aminotransferase'].max()) , 850) )
p5.set_yticks( np.arange(0, 1, 0.1) )

p6 = sns.histplot(data=Data_Python, x="Total_Protiens", stat="proportion",
    ↪ bins=15, color="pink", ax=axes[1, 2])
p6.axes.set(xlabel='Total_Protiens', ylabel=' ')
p6.set_xticks( range(int(Data_Python['Total_Protiens'].min()),
    ↪ int(Data_Python['Total_Protiens'].max()+1) , 1) )
p6.set_yticks( np.arange(0, 1, 0.1) )

p7 = sns.histplot(data=Data_Python, x="Albumin", stat="proportion",
    ↪ bins=15, color="orange", ax=axes[2, 0])
p7.axes.set(xlabel='Albumin', ylabel=' ')
p7.set_xticks( range(int(Data_Python['Albumin'].min()),
    ↪ int(Data_Python['Albumin'].max()+1) , 1) )
p7.set_yticks( np.arange(0, 1, 0.1) )

p8 = sns.histplot(data=Data_Python, x="Albumin_and_Globulin_Ratio",
    ↪ stat="proportion", bins=15, color="red", ax=axes[2, 1])
p8.axes.set(xlabel='Albumin_and_Globulin_Ratio', ylabel=' ')
p8.set_xticks( np.arange(0, 2, 0.5) )
p8.set_yticks( np.arange(0, 1, 0.1) )

```

```

p9 = sns.histplot(data=Data_Python, x="Age", stat="proportion", bins=15,
    ↪ color="green", ax=axes[2, 2])
p9.axes.set(xlabel='Age', ylabel=' ')
p9.set_xticks( range(int(Data_Python['Age'].min()) ,
    ↪ int(Data_Python['Age'].max()) , 10) )
p9.set_yticks( np.arange(0, 1, 0.1) )

plt.show()

```

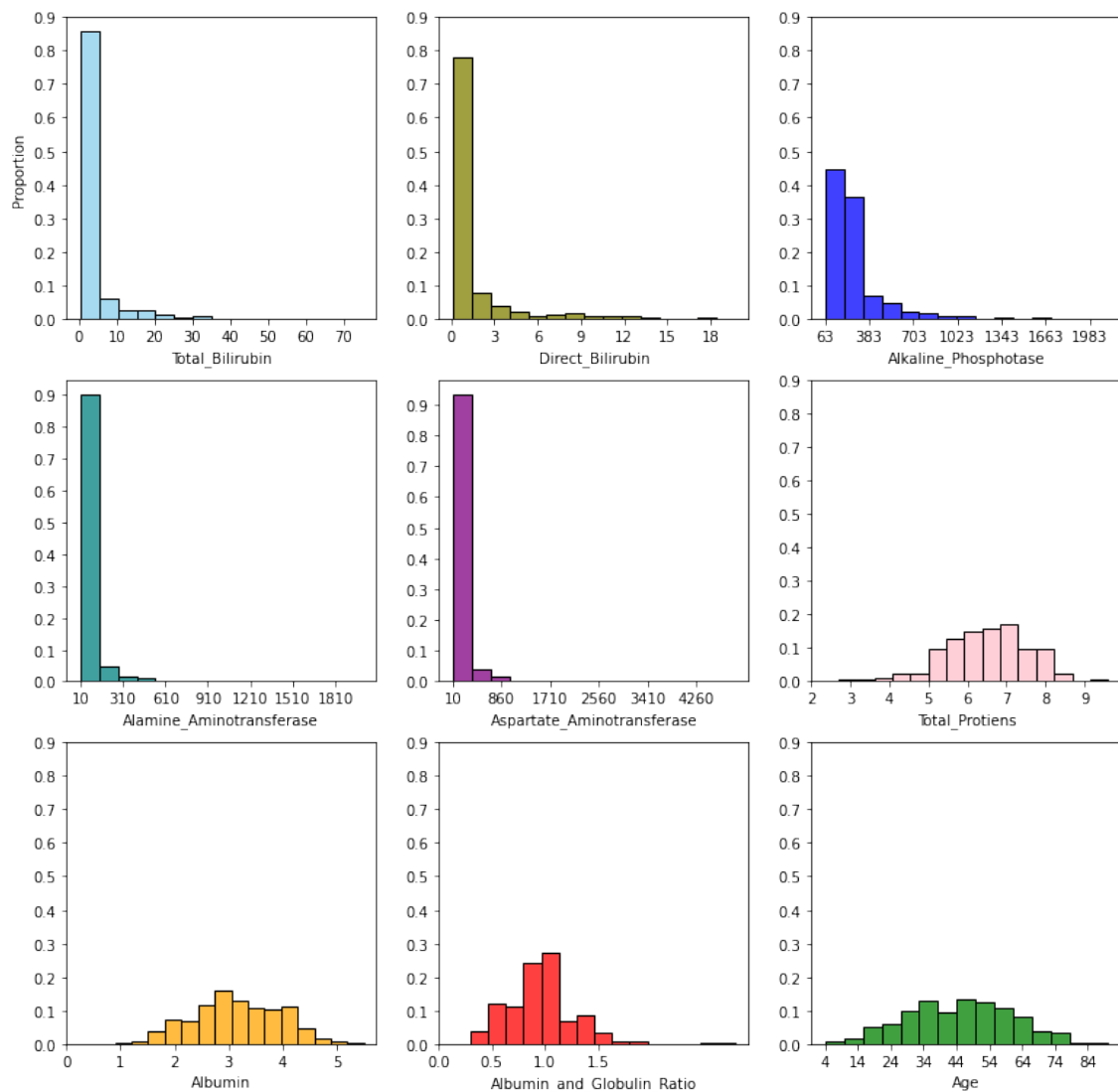


Figure 1: png

4.3.3.1 4.3.3.1. Histogramas para las variables cuantitativas

4.3.3.2 4.3.3.2. Diagramas de barras para las variables categóricas Ahora vamos a realizar una serie de operaciones para generar dos diagramas de barras, uno para la variable Gender y otro para Diseased.

```

proportion_Female = len( Data_Python.loc[ Data_Python['Gender']==0 , :] )
    ↪ / len(Data_Python)
proportion_Male = len( Data_Python.loc[ Data_Python['Gender']==1 , :] ) /
    ↪ len(Data_Python)

proportion_Diseased_yes = len( Data_Python.loc[ Data_Python['Diseased']==0
    ↪ , :] ) / len(Data_Python)
proportion_Diseased_no = len( Data_Python.loc[ Data_Python['Diseased']==1
    ↪ , :] ) / len(Data_Python)

```

```

Data_Python['proportion_Gender'] = 0

for i in range(0, len(Data_Python)):

    if Data_Python['Gender'][i] == 0 :

        Data_Python['proportion_Gender'][i] = proportion_Female

    else :

        Data_Python['proportion_Gender'][i] = proportion_Male

```

```

Data_Python['proportion_Diseased'] = 0

for i in range(0, len(Data_Python)):

    if Data_Python['Diseased'][i] == 0 :

        Data_Python['proportion_Diseased'][i] = proportion_Diseased_yes

    else :

        Data_Python['proportion_Diseased'][i] = proportion_Diseased_no

```

```

fig, axs = plt.subplots(1, 2, figsize=(8, 8))

p1 = sns.barplot(x='Gender', y='proportion_Gender', data=Data_Python,
    ↪ ax=axs[0])
p1.set_yticks( np.arange(0, 0.85, 0.1) )
p1.set_xticklabels(['Female', 'Male'])
p1.axes.set(xlabel='Gender', ylabel='proportion')

p2 = sns.barplot(x='Diseased', y='proportion_Diseased', data=Data_Python,
    ↪ ax=axs[1])
p2.set_yticks( np.arange(0, 0.85, 0.1) )

```

```

p2.set_xticklabels(['Yes', 'No'])
p2.axes.set(xlabel='Diseased', ylabel=' ')

plt.show()

```

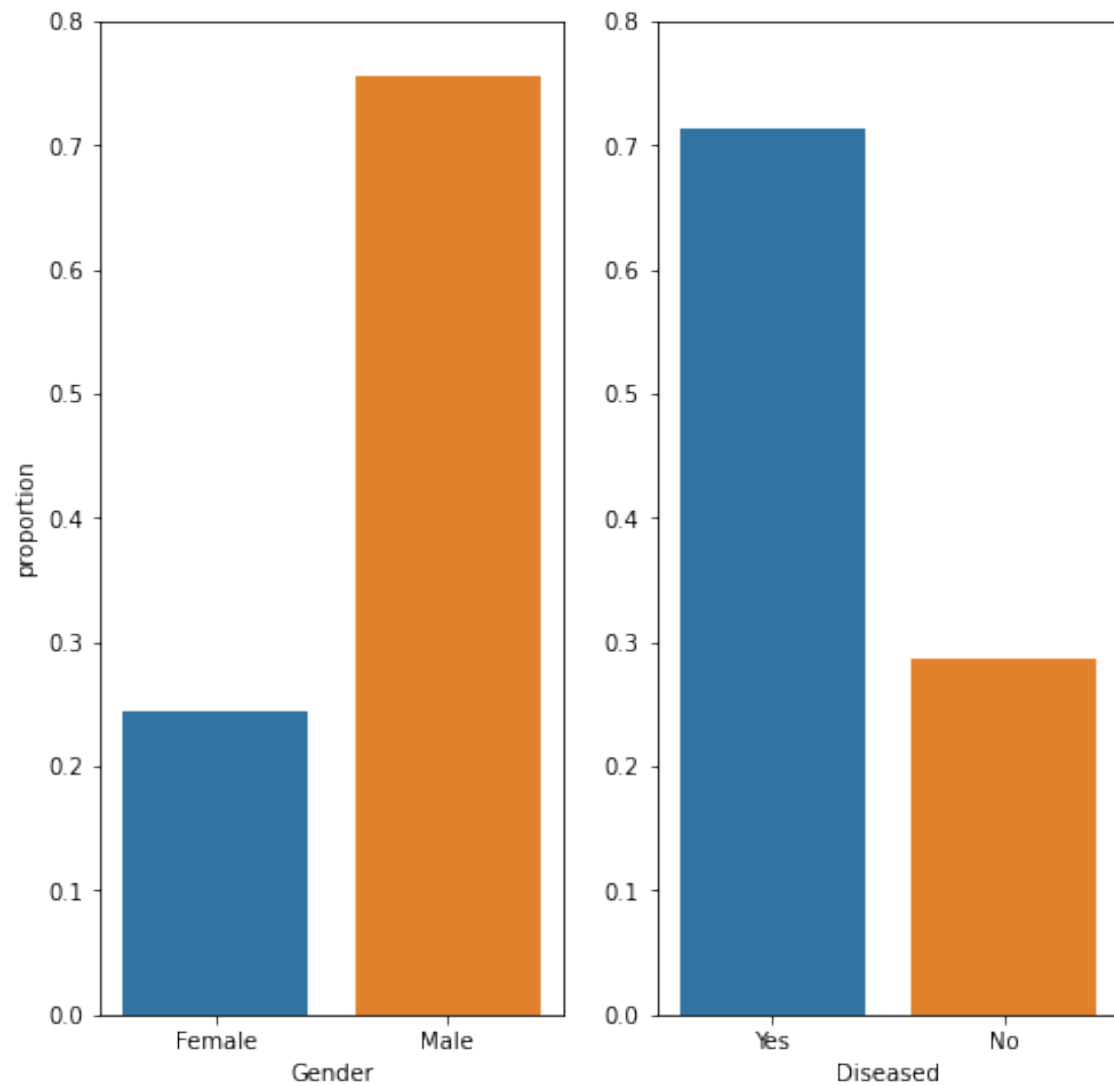


Figure 2: png

```
[ proportion_Female , proportion_Male ]
```

```
[0.24356775300171526, 0.7564322469982847]
```

```
[ proportion_Diseased_yes , proportion_Diseased_no ]
```

```
[0.7135506003430532, 0.2864493996569468]
```

Como puede verse el porcentaje de mujeres en la muestra es del 24.36% , mientras que el de hombres es del 75.64%

Por otro lado el porcentaje de endfermos es del 71.36%, mientras que el de no enfermos es del 28.64%

```
fig, axs = plt.subplots(3, 3, figsize=(13, 13))

p1 = sns.boxplot(x=Data_Python['Age'], color="palegreen", ax=axs[0, 0])
p1.set_xticks( range(int(Data_Python['Age'].min()) ,
    ↪ int(Data_Python['Age'].max()+10) , 10) )

p2 = sns.boxplot(x=Data_Python['Direct_Bilirubin'], color="olive",
    ↪ ax=axs[0, 1])
p2.set_xticks( range(int(Data_Python['Direct_Bilirubin'].min()) ,
    ↪ int(Data_Python['Direct_Bilirubin'].max()) , 10) )

p3 = sns.boxplot(x=Data_Python['Alkaline_Phosphotase'], color="blue",
    ↪ ax=axs[1, 0])
p3.set_xticks( range(int(Data_Python['Alkaline_Phosphotase'].min()) ,
    ↪ int(Data_Python['Alkaline_Phosphotase'].max()) , 300) )

p4 = sns.boxplot(x=Data_Python['Alamine_Aminotransferase'], color="teal",
    ↪ ax=axs[1, 1])
p4.set_xticks( range(int(Data_Python['Alamine_Aminotransferase'].min()) ,
    ↪ int(Data_Python['Alamine_Aminotransferase'].max()) , 500) )

p5 = sns.boxplot(x=Data_Python['Aspartate_Aminotransferase'],
    ↪ color="purple", ax=axs[0, 2])
p5.set_xticks( range(int(Data_Python['Aspartate_Aminotransferase'].min())
    ↪ , int(Data_Python['Aspartate_Aminotransferase'].max()) , 850) )

p6 = sns.boxplot(x=Data_Python['Total_Protiens'], color="pink", ax=axs[1,
    ↪ 2])
p6.set_xticks( range(int(Data_Python['Total_Protiens'].min()) ,
    ↪ int(Data_Python['Total_Protiens'].max()) , 10) )

p7 = sns.boxplot(x=Data_Python['Albumin'], color="orange", ax=axs[2, 2])
p7.set_xticks( range(int(Data_Python['Albumin'].min()) ,
    ↪ int(Data_Python['Albumin'].max()) , 10) )

p8 = sns.boxplot(x=Data_Python['Albumin_and_Globulin_Ratio'], color="red",
    ↪ ax=axs[2, 1])
p8.set_xticks( range(int(Data_Python['Albumin_and_Globulin_Ratio'].min())
    ↪ , int(Data_Python['Albumin_and_Globulin_Ratio'].max()) , 10) )

p9 = sns.boxplot(x=Data_Python['Total_Bilirubin'], color="skyblue",
    ↪ ax=axs[2, 0])
```

```
p9.set_xticks( range(int(Data_Python['Total_Bilirubin'].min()) ,
↪ int(Data_Python['Total_Bilirubin'].max()) , 10) )

plt.show()
```

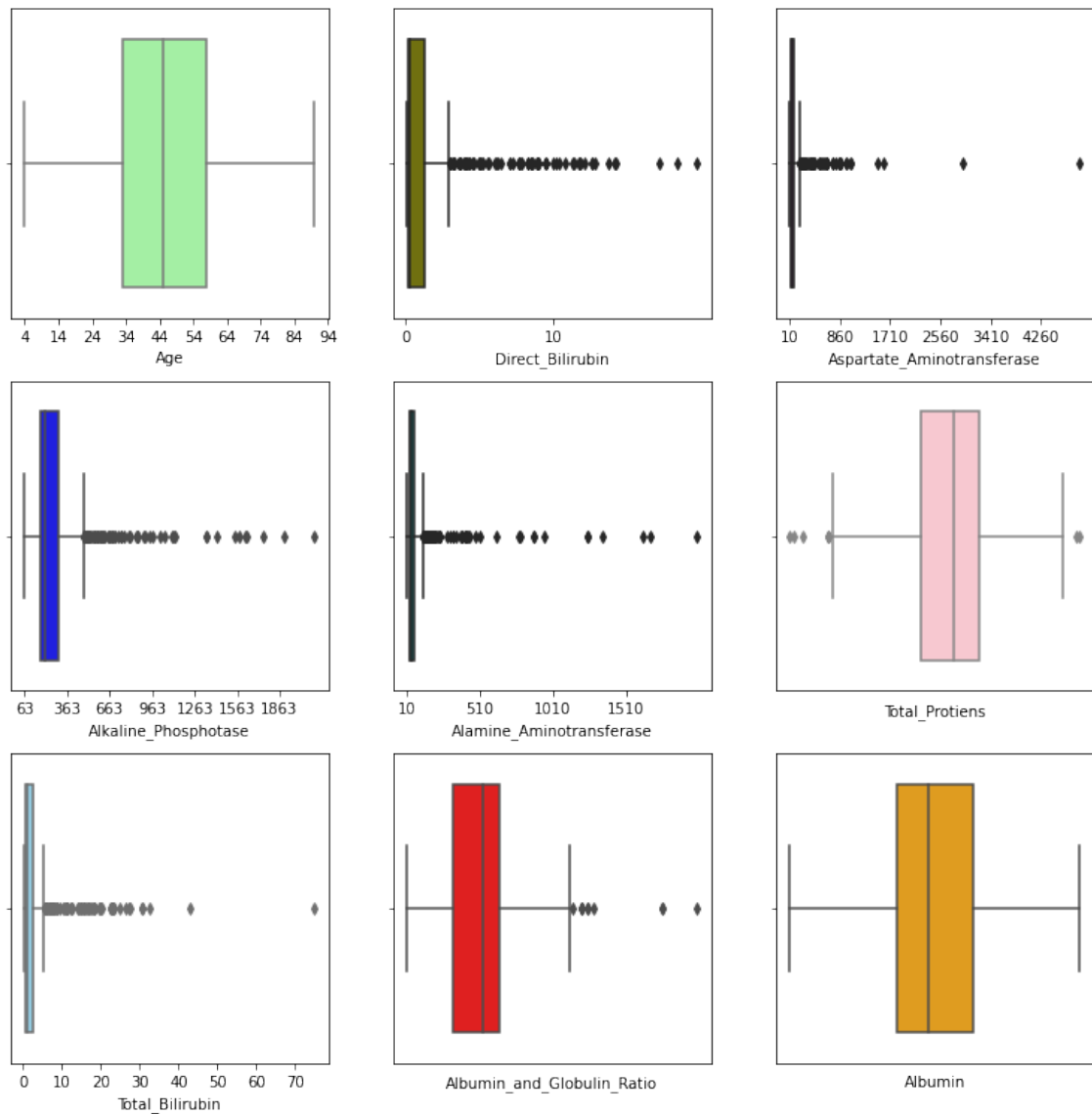


Figure 3: png

4.3.3.3. Box-plots para las variables cuantitativas

4.3.4. Análisis de la relación entre los predictores categoricos y la respuesta

4.3.4.1. Analisis relación entre respuesta (Diseased) y Gender Frecuencia relativa de genero condicionada a enfermedad

Ahora vamos a realizar una serie de operaciones para obtener una tabla de frecuencias relativas de la variable Gender condicionada a la respuesta (Diseased). También obtendremos el gráfico de barras asociado a esta tabla.

```
Df_Diseased_Yes = Data_Python.loc[ Data_Python['Diseased']==0 , :]

proportion_Female_in_Diseased_Yes = len(
    ↳ Df_Diseased_Yes.loc[Df_Diseased_Yes['Gender']==0 , :] ) /
    ↳ len(Df_Diseased_Yes)

#####

Df_Diseased_Yes = Data_Python.loc[ Data_Python['Diseased']==0 , :]

proportion_Male_in_Diseased_Yes = len(
    ↳ Df_Diseased_Yes.loc[Df_Diseased_Yes['Gender']==1 , :] ) /
    ↳ len(Df_Diseased_Yes)

#####

Df_Diseased_No = Data_Python.loc[ Data_Python['Diseased']==1 , :]

proportion_Female_in_Diseased_No = len(
    ↳ Df_Diseased_No.loc[Df_Diseased_No['Gender']==0 , :] ) /
    ↳ len(Df_Diseased_No)

#####

Df_Diseased_No = Data_Python.loc[ Data_Python['Diseased']==1 , :]

proportion_Male_in_Diseased_No = len(
    ↳ Df_Diseased_No.loc[Df_Diseased_No['Gender']==1 , :] ) /
    ↳ len(Df_Diseased_No)
```

Función para calcular tablas de frecuencias relativas condicionadas con dos variables:

```
def Table_Con_Rel_Freq_2_Var_Py (df, var1, p1, p2, var1_name='var1' ,
    ↳ var2_name='var2') :

    table = np.zeros(( p2+1 , p1+1 ))
    table[:] = np.nan

    #####

    for i in range(0, p2+1):
        for j in range(0, p1+1):

            df_new = df.loc[ var1 == j , : ]
```

```

        table[i,j] = len( df_new.loc[ df_new[var2_name] == i , :] ) /
↪ len(df_new)

    table = pd.DataFrame(table)

    return table

```

```

Frec_Relativas_Condicionadas_Gender_in_Diseased =
↪ Table_Con_Rel_Freq_2_Var_Py (Data_Python, Data_Python['Diseased'], 1,
↪ 1, var1_name='Diseased' , var2_name='Gender')

Frec_Relativas_Condicionadas_Gender_in_Diseased.index = ['Female' ,
↪ 'Male']
Frec_Relativas_Condicionadas_Gender_in_Diseased.columns = ['Yes' , 'No']
Frec_Relativas_Condicionadas_Gender_in_Diseased =
↪ Frec_Relativas_Condicionadas_Gender_in_Diseased.style.set_caption("Gender
↪ | Diseased ")

```

```

p1 = sns.countplot(data=Data_Python, x="Diseased", hue="Gender",
↪ palette="husl")
p1.set_xticklabels(['Yes', 'No'])
p1.legend(title='Gender', loc='upper right', labels=['Female', 'Male'])

```

<matplotlib.legend.Legend at 0x2736f2a9750>

Frec_Relativas_Condicionadas_Gender_in_Diseased

Gender | Diseased

Yes

No

Female

0.221154

0.299401

Male

0.778846

0.700599

[proportion_Female_in_Diseased_Yes , proportion_Male_in_Diseased_Yes]

[0.22115384615384615, 0.7788461538461539]

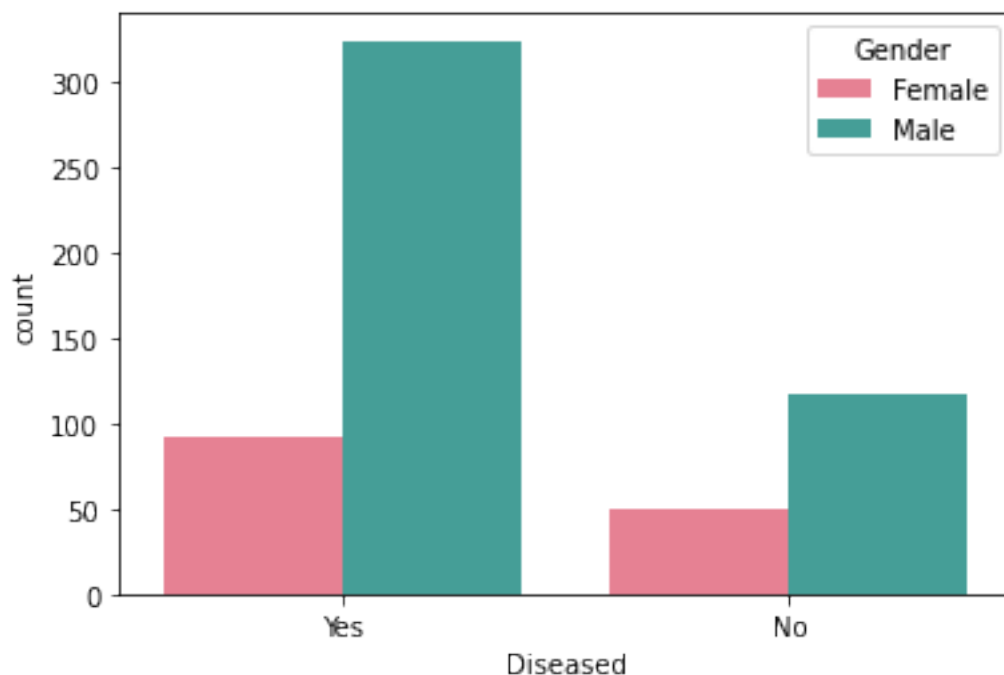


Figure 4: png

```
[proportion_Female_in_Diseased_No , proportion_Male_in_Diseased_No]
```

```
[0.2994011976047904, 0.7005988023952096]
```

Como puede observarse el porcentaje de mujeres dentro del grupo de los enfermos es del 22.12% , mientras que el de hombres es del 77.88%.

Por otro lado el porcentaje de mujeres dentro del grupo de los no enfermos es del 29.94%, mientras que el de hombres es del 70.06%

Frecuencia relativa de enfermedad condicionada al genero

Ahora vamos a realizar una serie de operaciones para obtener una tabla de frecuencias relativas de la variable respuesta (Diseased) condicionada a la variable Gender . También obtendremos el gráfico de barras asociado a esta tabla.

```
Frec_Relativas_Condicionadas_Diseased_in_Gender =
↳ Table_Con_Rel_Freq_2_Var_Py (Data_Python, Data_Python['Gender'], 1, 1,
↳ var1_name='Gender' , var2_name='Diseased')
Frec_Relativas_Condicionadas_Diseased_in_Gender.index = ['Yes' , 'No']
Frec_Relativas_Condicionadas_Diseased_in_Gender.columns = ['Female' ,
↳ 'Male']
Frec_Relativas_Condicionadas_Diseased_in_Gender =
↳ Frec_Relativas_Condicionadas_Diseased_in_Gender.style.set_caption("Diseased
↳ | Gender")
```

```
p = sns.countplot(data=Data_Python, x="Gender", hue="Diseased",
↪ palette="husl")
p.set_xticklabels(['Female', 'Male'])
p.legend(title='Diseased', loc='upper right', labels=['Yes', 'No'])
```

<matplotlib.legend.Legend at 0x2736f2d2bc0>

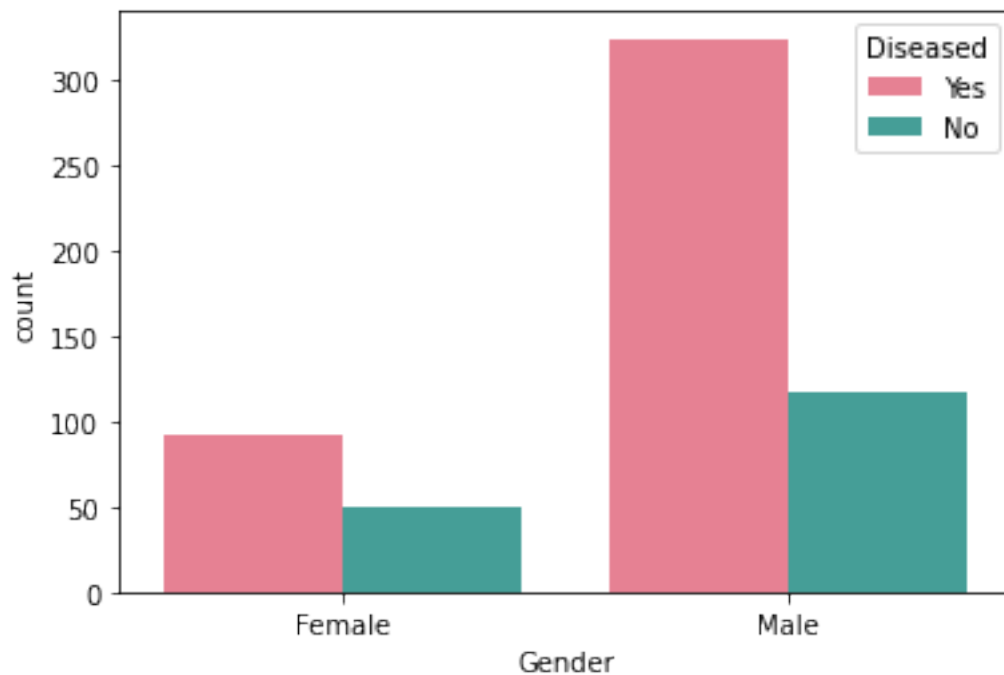


Figure 5: png

Frec_Relativas_Condicionadas_Diseased_in_Gender

Diseased | Gender

Female

Male

Yes

0.647887

0.734694

No

0.352113

0.265306

Como puede observarse el porcentaje de enfermos dentro del grupo de as mujeres es del 64.79% , mientras que el de no enfermos es del 35.21%.

Por otro lado el porcentaje de enfermos dentro del grupo de los hombres es del 73.47%, mientras que el de no enfermos es del 26.53%

4.3.4.2 4.3.4.2. Analisis relación entre respuesta (Diseased) y Grupo de Edad Frecuecnia relativa de enfermedad en funcion del grupo de edad

Tenemos que categorizar la variable cuantitativa Age (edad) , para ello debemos emplear una regla de categorización (mediana, media, cuartiles, Scott ...)

Usaremos la regla de los cuartiles por simplicidad:

```
intervals = np.quantile( Data_Python['Age'] , [0, 0.25, 0.5, 0.75 , 1])
intervals
```

```
array([ 4., 33., 45., 58., 90.])
```

Nos apoyaremos en la funcion `cut()` de la libreria **Pandas** para categorizar la variable Age usando la regla de los cuartiles.

A esta función le das un vector (bins) y construye unos intervalos con los elementos del vector, en este caso (3, 33], (33, 45], (45, 58], (58, 90]. Luego te devuelve a qué intervalo pertenece cada observación de una variable dada (en nuestro caso Age), y también nos permite codificar estos intervalos con la codificación estandar (0,1,2,...), y así obtener una nueva variable que es una versión categorizada de la variable pasada (Age en nuestro caso).

Vamos a restar una cantidad positiva (por ejemplo 1) al mínimo de Age, puesto que ese valor será el extremo inferior del primer intervalo, y dicho intervalo será abierto en ese extremo (por configuración de la función `cut`), por tanto si no restasemos una cantidad positiva, el valor mínimo de Age no estaría en ninguno de los intervalos generados por `cut()`

```
intervals[0] = intervals[0] - 1
intervals
```

```
array([ 3., 33., 45., 58., 90.])
```

```
pd.cut(x=Data_Python['Age'] , bins=intervals , right=True)
```

```
0      (58.0, 90.0]
1      (58.0, 90.0]
2      (58.0, 90.0]
3      (45.0, 58.0]
4      (58.0, 90.0]
...
578    (58.0, 90.0]
579    (33.0, 45.0]
```

```

580    (45.0, 58.0]
581    (3.0, 33.0]
582    (33.0, 45.0]
Name: Age, Length: 583, dtype: category
Categories (4, interval[float64, right]): [(3.0, 33.0] < (33.0, 45.0] < (45.0, 58.0] < (5

```

```
pd.cut(x=Data_Python['Age'] , bins=intervals , labels=False)
```

```

0      3
1      3
2      3
3      2
4      3
..
578    3
579    1
580    2
581    0
582    1
Name: Age, Length: 583, dtype: int64

```

```

Data_Python['Age_cat'] = pd.cut(x=Data_Python['Age'] , bins=intervals ,
↪ labels=False)

```

La nueva variable Age_cat es tal que:

$$Age_cat_i = \begin{cases} 0, & \text{if } Age_i \in [Min(Age), Q(0.25, Age)] \\ 1, & \text{if } Age_i \in (Q(0.25, Age), Q(0.50, Age)] \\ 2, & \text{if } Age_i \in (Q(0.50, Age), Q(0.75, Age)] \\ 3, & \text{if } Age_i \in (Q(0.75, Age), Max(Age)] \end{cases}$$

para $i = 1, \dots, n$

Ahora tenemos una variable que nos indica el grupo de edad de cada individuo. Tenemos tres grupos de edad.

Grupo 0: ≤ 33 años

Grupo 1: entre 33 y 45 años

Grupo 2: entre 45 y 58 años

grupo 3: > 58 años

Ahora vamos a generar una tabla de frecuencias relativas de la variable respuesta (Diseased) condicionada a la nueva variable Grupo de edad (Age_cat), también generaremos su gráfico de barras asociado.

```

Frec_Relativas_Condicionadas_Diseased_in_Aged =
→ Table_Con_Rel_Freq_2_Var_Py (Data_Python, Data_Python['Age_cat'], 3,
→ 1, var1_name='Age_cat' , var2_name='Diseased')
Frec_Relativas_Condicionadas_Diseased_in_Aged.index = ['Yes' , 'No']
Frec_Relativas_Condicionadas_Diseased_in_Aged.columns = ['(3, 33]', '(33,
→ 45]', '(45, 58]', '(58, 90]']
Frec_Relativas_Condicionadas_Diseased_in_Aged =
→ Frec_Relativas_Condicionadas_Diseased_in_Aged.style.set_caption("Diseased
→ | Age Group")

```

```

p = sns.countplot(data=Data_Python, x="Age_cat", hue="Diseased",
→ palette="husl")
p.set_xticklabels(['(3, 33]', '(33, 45]', '(45, 58]', '(58, 90]'])
p.legend(title='Diseased', loc='upper right', labels=['Yes', 'No'] )

```

<matplotlib.legend.Legend at 0x2736ed18ac0>

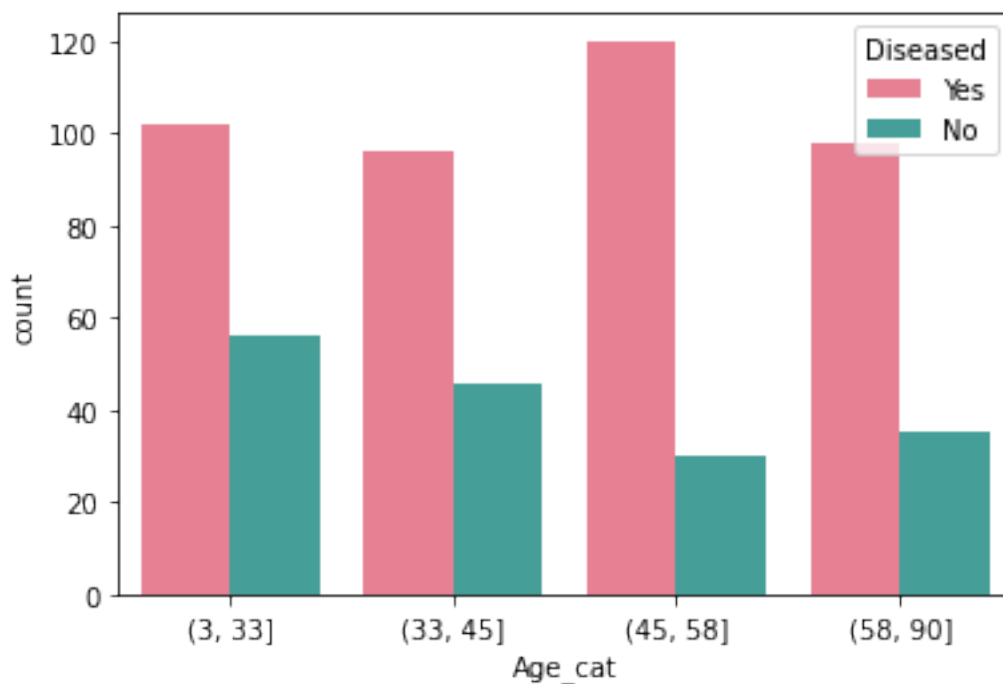


Figure 6: png

```

Frec_Relativas_Condicionadas_Diseased_in_Aged

```

Diseased | Age Group

(3, 33]

(33, 45]

(45, 58]
 (58, 90]
 Yes
 0.645570
 0.676056
 0.800000
 0.736842
 No
 0.354430
 0.323944
 0.200000
 0.263158

Como puede observarse dentro del grupo de los más jóvenes (de edad menor o igual a 33) el porcentaje de enfermos es del 64.56% , este porcentaje aumenta hasta el 67.60% en el grupo de individuos cuya edad esta entre 33 y 45 años, y hasta el 80% en el de los individuos con una edad entre 45 y 58 años, luego pasa a ser del 73.68% en el grupo de edad superior a 58 años.

Frecuecnia relativa de grupo de edad en funcion de enfermedad

Ahora vamos a generar una tabla de frecuencias relativas de la variable grupo de edad (Age_cat) condicionada a la variable respuesta, también generaremos su gráfico de barras asociado.

```
Frec_Relativas_Condicionadas_Age_in_Diseased = Table_Con_Rel_Freq_2_Var_Py
→ (Data_Python, Data_Python['Diseased'], 1, 3, var1_name='Diseased' ,
→ var2_name='Age_cat')
Frec_Relativas_Condicionadas_Age_in_Diseased.index = ['(3, 33]', '(33,
→ 45]', '(45, 58]', '(58, 90]']
Frec_Relativas_Condicionadas_Age_in_Diseased.columns = ['Yes' , 'No']
Frec_Relativas_Condicionadas_Age_in_Diseased =
→ Frec_Relativas_Condicionadas_Age_in_Diseased.style.set_caption("Age
→ Group | Diseased")
```

```
p = sns.countplot(data=Data_Python, x="Diseased", hue="Age_cat",
→ palette="husl")
p.set_xticklabels(['Yes', 'No'])
p.legend(title='Group Age', loc='upper right', labels= ['(3, 33]', '(33,
→ 45]', '(45, 58]', '(58, 90]'])
```

<matplotlib.legend.Legend at 0x2736ed181c0>

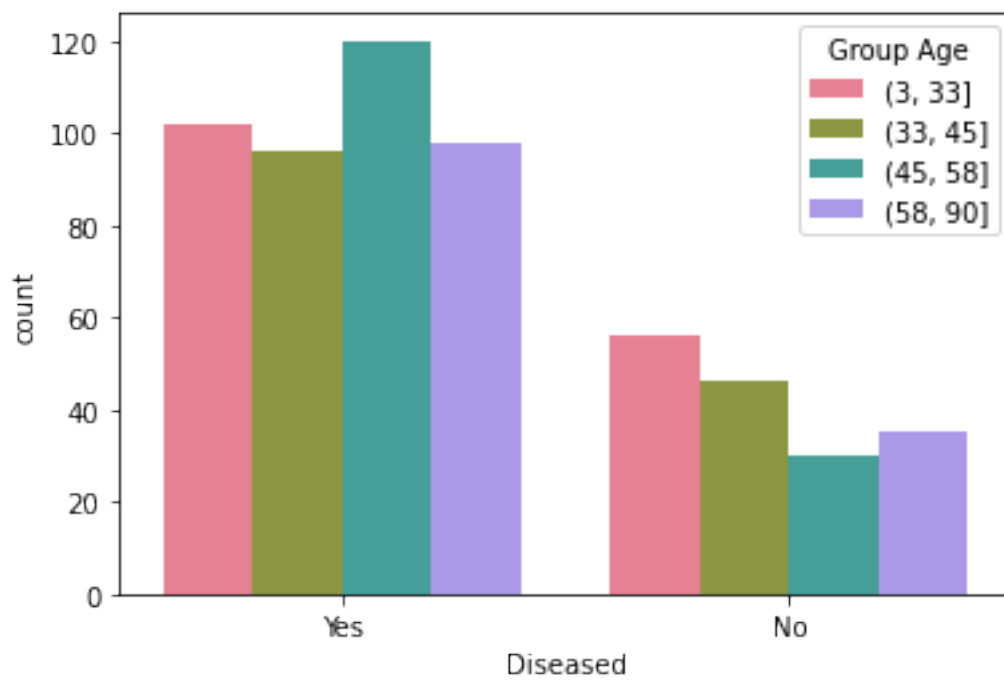


Figure 7: png

Frec_Relativas_Condicionadas_Age_in_Diseased

Age Group | Diseased

Yes

No

(3, 33]

0.245192

0.335329

(33, 45]

0.230769

0.275449

(45, 58]

0.288462

0.179641

(58, 90]

0.235577

0.209581

Se puede apreciar que dentro del grupo de los enfermos , el grupo de edad mas frecuente seria el de los individuos con una edad entre 45 y 58 años, seguido del de más de 58 años.

Por otro lado dentro del grupo de los no enfermos el grupo de edad claramente mayoritario es el de los mas jovenes (edad menor o igual a 33 años), seguido del siguiente grupo mas joven (edad entre 33 y 45).

4.3.5. Análisis de la relación entre los predictores cuantitativos y la respuesta

4.3.5.1. Resumen Estadístico Descriptivo Cuantitativo en función de Diseased Ahora vamos a hacer una serie de operaciones para obtener una tabla en la que se pueden comparar los valores de diferentes estadísticos descriptivos básicos para cada variable cuantitativa en función del valor de la respuesta (Diseased).

```
Data_Python_Quantitative_Diseased_yes = Data_Python.loc[
→ Data_Python['Diseased'] == 0 , (Data_Python.columns != 'Gender') &
→ (Data_Python.columns != 'Diseased') & (Data_Python.columns !=
→ 'proportion_Gender') & (Data_Python.columns != 'proportion_Diseased')
→ & (Data_Python.columns != 'Age_cat') ]
Data_Python_Quantitative_Diseased_no = Data_Python.loc[
→ Data_Python['Diseased'] == 1 , (Data_Python.columns != 'Gender') &
→ (Data_Python.columns != 'Diseased') & (Data_Python.columns !=
→ 'proportion_Gender') & (Data_Python.columns != 'proportion_Diseased')
→ & (Data_Python.columns != 'Age_cat') ]
```

```
std_yes = Data_Python_Quantitative_Diseased_yes.std()
std_no = Data_Python_Quantitative_Diseased_no.std()

mean_yes = Data_Python_Quantitative_Diseased_yes.mean()
mean_no = Data_Python_Quantitative_Diseased_no.mean()

Q25_yes = Data_Python_Quantitative_Diseased_yes.quantile(q=0.25)
Q25_no = Data_Python_Quantitative_Diseased_no.quantile(q=0.25)

Q50_yes = Data_Python_Quantitative_Diseased_yes.quantile(q=0.5)
Q50_no = Data_Python_Quantitative_Diseased_no.quantile(q=0.5)

Q75_yes = Data_Python_Quantitative_Diseased_yes.quantile(q=0.75)
Q75_no = Data_Python_Quantitative_Diseased_no.quantile(q=0.75)

min_yes = Data_Python_Quantitative_Diseased_yes.min()
min_no = Data_Python_Quantitative_Diseased_no.min()

max_yes = Data_Python_Quantitative_Diseased_yes.max()
max_no = Data_Python_Quantitative_Diseased_no.max()
```

```
df_yes = pd.DataFrame({'mean':mean_yes , 'min':min_yes , 'Q25':Q25_yes ,
→ 'median':Q50_yes , 'Q75':Q75_yes , 'max':max_yes , 'std':std_yes})
```

```
df_no = pd.DataFrame({'mean':mean_no , 'min':min_no , 'Q25':Q25_no ,
→ 'median':Q50_no , 'Q75':Q75_no , 'max':max_no , 'std':std_no})
```

```
Statistics_Quantitatives_Diseased = pd.DataFrame({

    'Age_yes':df_yes.iloc[0,:] ,
    'Age_no':df_no.iloc[0,:],
    'Total_Bilirubin_yes':df_yes.iloc[1,:] ,
    'Total_Bilirubin_no':df_no.iloc[1,:],
    'Direct_Bilirubin_yes':df_yes.iloc[2,:] ,
    'Direct_Bilirubin_no':df_no.iloc[2,:],
    'Alkaline_Phosphotase_yes':df_yes.iloc[3,:] ,
    'Alkaline_Phosphotase_no':df_no.iloc[3,:],
    'Alamine_Aminotransferase_yes':df_yes.iloc[4,:] ,
    'Alamine_Aminotransferase_no':df_no.iloc[4,:],
    'Aspartate_Aminotransferase_yes':df_yes.iloc[5,:] ,
    'Aspartate_Aminotransferase_no':df_no.iloc[5,:],
    'Total_Protiens_yes':df_yes.iloc[6,:] ,
    'Total_Protiens_no':df_no.iloc[6,:],
    'Albumin_yes':df_yes.iloc[7,:] ,
    'Albumin_no':df_no.iloc[7,:],
    'Albumin_and_Globulin_Ratio_yes':df_yes.iloc[8,:] ,
    'Albumin_and_Globulin_Ratio_no':df_no.iloc[8,:],

    })
```

Statistics_Quantitatives_Diseased

Age__yes

Age__no

Total__Bilirubin__yes

Total__Bilirubin__no

Direct__Bilirubin__yes

Direct__Bilirubin__no

Alkaline__Phosphotase__yes

Alkaline__Phosphotase__no

Alamine__Aminotransferase__yes

Alamine__Aminotransferase__no

Aspartate__Aminotransferase__yes

Aspartate__Aminotransferase__no

Total_Protiens_yes
 Total_Protiens_no
 Albumin_yes
 Albumin_no
 Albumin_and_Globulin_Ratio_yes
 Albumin_and_Globulin_Ratio_no
 mean
 46.153846
 41.239521
 4.164423
 1.142515
 1.923558
 0.396407
 319.007212
 219.754491
 99.605769
 33.652695
 137.699519
 40.688623
 6.459135
 6.543114
 3.060577
 3.344311
 0.914337
 1.028588
 min
 7.000000
 4.000000
 0.400000
 0.500000
 0.100000
 0.100000
 63.000000
 90.000000
 12.000000

10.000000
11.000000
10.000000
2.700000
3.700000
0.900000
1.400000
0.300000
0.370000
Q25
34.000000
28.000000
0.800000
0.700000
0.200000
0.200000
186.000000
161.500000
25.000000
20.000000
29.750000
21.000000
5.700000
5.900000
2.500000
2.900000
0.700000
0.900000
median
46.000000
40.000000
1.400000
0.800000
0.500000
0.200000

229.000000
186.000000
41.000000
27.000000
52.500000
29.000000
6.550000
6.600000
3.000000
3.400000
0.900000
1.000000
Q75
58.000000
55.000000
3.625000
1.100000
1.800000
0.350000
315.250000
213.000000
76.500000
37.500000
108.750000
43.500000
7.200000
7.300000
3.625000
4.000000
1.100000
1.200000
max
90.000000
85.000000
75.000000

7.300000
19.700000
3.600000
2110.000000
1580.000000
2000.000000
181.000000
4929.000000
285.000000
9.600000
9.200000
5.500000
5.000000
2.800000
1.900000
std
15.654412
16.999366
7.144831
1.004472
3.206901
0.519255
268.307911
140.986262
212.768472
25.060392
337.389980
36.411620
1.094659
1.063042
0.786595
0.783690
0.325374
0.285658

Esta tabla nos aporta información muy relevante, algunos ejemplos son los siguientes:

- La media de edad en el grupo de los enfermos es de 46 años mientras que en el de no enfermos es 41.24
- Hay un 25% de los enfermos que tienen un valor de Total_Bilirubin superior a 3.62, mientras que en el grupo de los no enfermos solo un 25% supera un valor de 1.1.

4.3.5.2 4.3.5.2. Diagrama de puntos de la respuesta (Diseased) en función de predictores cuantitativos En este caso vamos a generar unos diagramas de puntos de la respuesta en función de cada uno de los predictores cuantitativos. El diagrama de puntos que usaremos es un tipo especial que añade ruido horizontal a los puntos para poder así visualizar varios puntos que tengan un mismo valor real (valor sin ruido).

```
fig, axs = plt.subplots(3, 3, figsize=(15, 15))

p1 = sns.stripplot(data=Data_Python, x="Diseased", y="Age", jitter=0.3,
    ↪ size=4, color='red', ax=axs[0, 0])
p1.set_xticklabels(['Yes', 'No'])
p1.set_yticks( range(int(Data_Python['Age'].min()) ,
    ↪ int(Data_Python['Age'].max()) , 7) )

p2 = sns.stripplot(data=Data_Python, x="Diseased", y="Direct_Bilirubin",
    ↪ jitter=0.3, size=4, color='red', ax=axs[0, 1])
p2.set_xticklabels(['Yes', 'No'])

p3 = sns.stripplot(data=Data_Python, x="Diseased",
    ↪ y="Alkaline_Phosphotase", jitter=0.3, size=4, color='red', ax=axs[1,
    ↪ 0])
p3.set_xticklabels(['Yes', 'No'])

p4 = sns.stripplot(data=Data_Python, x="Diseased",
    ↪ y="Alamine_Aminotransferase", jitter=0.3, size=4, color='red',
    ↪ ax=axs[1, 1])
p4.set_xticklabels(['Yes', 'No'])

p5 = sns.stripplot(data=Data_Python, x="Diseased",
    ↪ y="Aspartate_Aminotransferase", jitter=0.3, size=4, color='red',
    ↪ ax=axs[0, 2])
p5.set_xticklabels(['Yes', 'No'])

p6 = sns.stripplot(data=Data_Python, x="Diseased", y="Total_Protiens",
    ↪ jitter=0.3, size=4, color='red', ax=axs[1, 2])
p6.set_xticklabels(['Yes', 'No'])

p7 = sns.stripplot(data=Data_Python, x="Diseased", y="Albumin",
    ↪ jitter=0.3, size=4, color='red', ax=axs[2, 2])
p7.set_xticklabels(['Yes', 'No'])

p8 = sns.stripplot(data=Data_Python, x="Diseased",
    ↪ y="Albumin_and_Globulin_Ratio", jitter=0.3, size=4, color='red',
    ↪ ax=axs[2, 1])
```



```

p8.set_xticklabels(['Yes', 'No'])

p9 = sns.stripplot(data=Data_Python, x="Diseased", y="Total_Bilirubin",
    ⇨ jitter=0.3, size=4, color='red', ax=axes[2, 0])
p9.set_xticklabels(['Yes', 'No'])

plt.show()

```

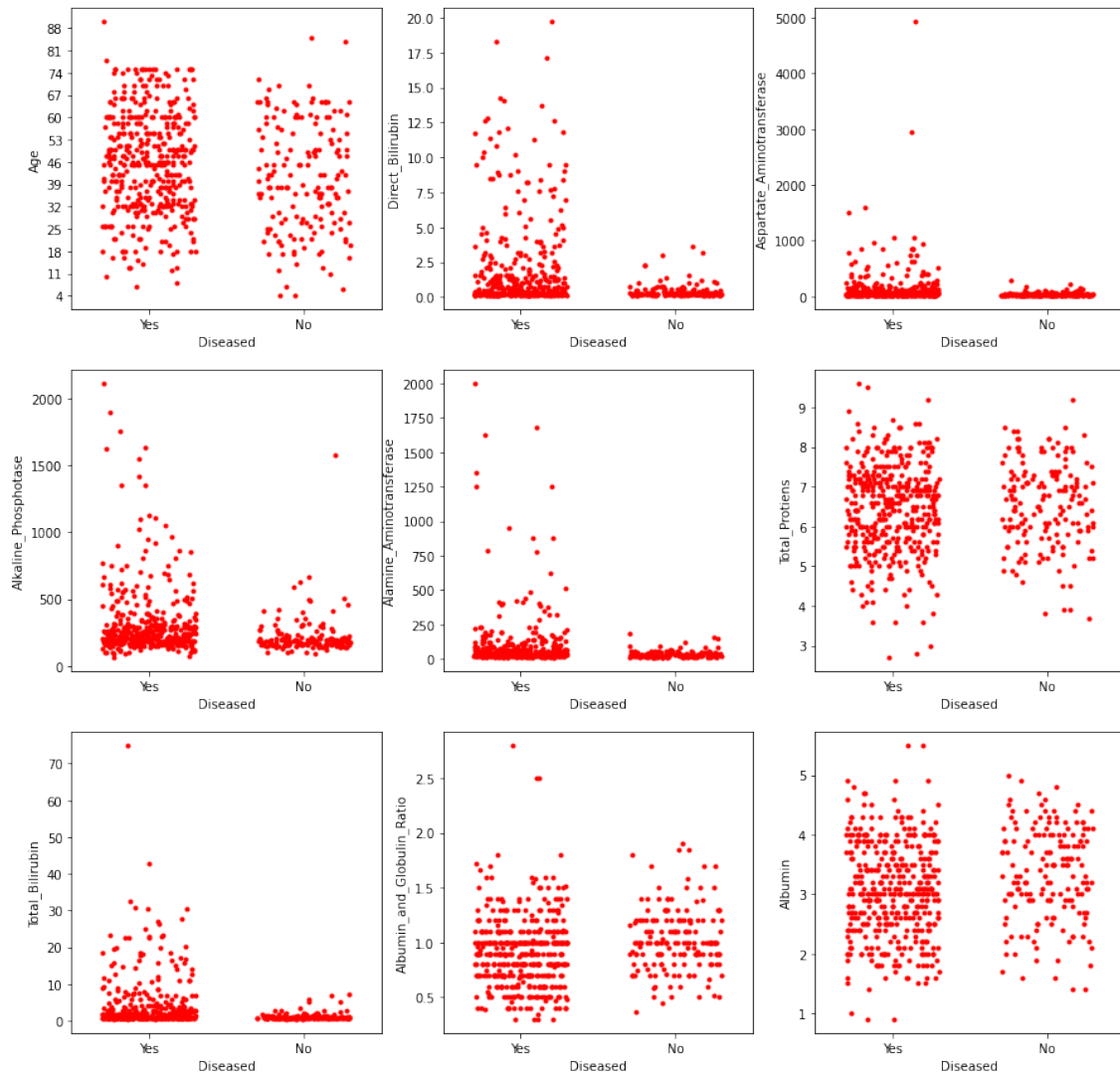


Figure 8: png

4.4 4.4. Árboles de clasificación en R

4.4.1 4.4.1. Algoritmo rpart con R

```
%%R
```

```
library(rpart)
library(rpart.plot)
library(caret)
library(tidyverse)
```

```
R[write to console]: Loading required package: lattice
```

```
R[write to console]:
Attaching package: 'caret'
```

```
R[write to console]: The following object is masked from 'package:purrr':
```

```
lift
```

Vamos a hacer la visualización del árbol con `rpart`

```
%%R
```

```
set.seed(0)
datos_entreno<-sample_frac(ilpd,0.75) # fraccionamos la muestra en
→ entrenamiento y test
datos_test<-setdiff(ilpd,datos_entreno)

arbol_0<-rpart(diseased~.,data = datos_entreno, method = "class", cp=0.01)
rpart.plot(arbol_0,
            extra = 104, # show fitted class, probs, percentages
            box.palette = "GnBu", # color scheme
            branch.lty = 3, # dotted branch lines
            shadow.col = "gray", # shadows under the node boxes
            nn = TRUE)
```

Ahora vamos a calcular las predicciones y la matriz de confusión:

```
%%R
```

```
prediccion1<-predict(arbol_0,newdata=datos_test,type="class")

matriz_confusion1<-confusionMatrix(prediccion1,datos_test[["diseased"]])
```

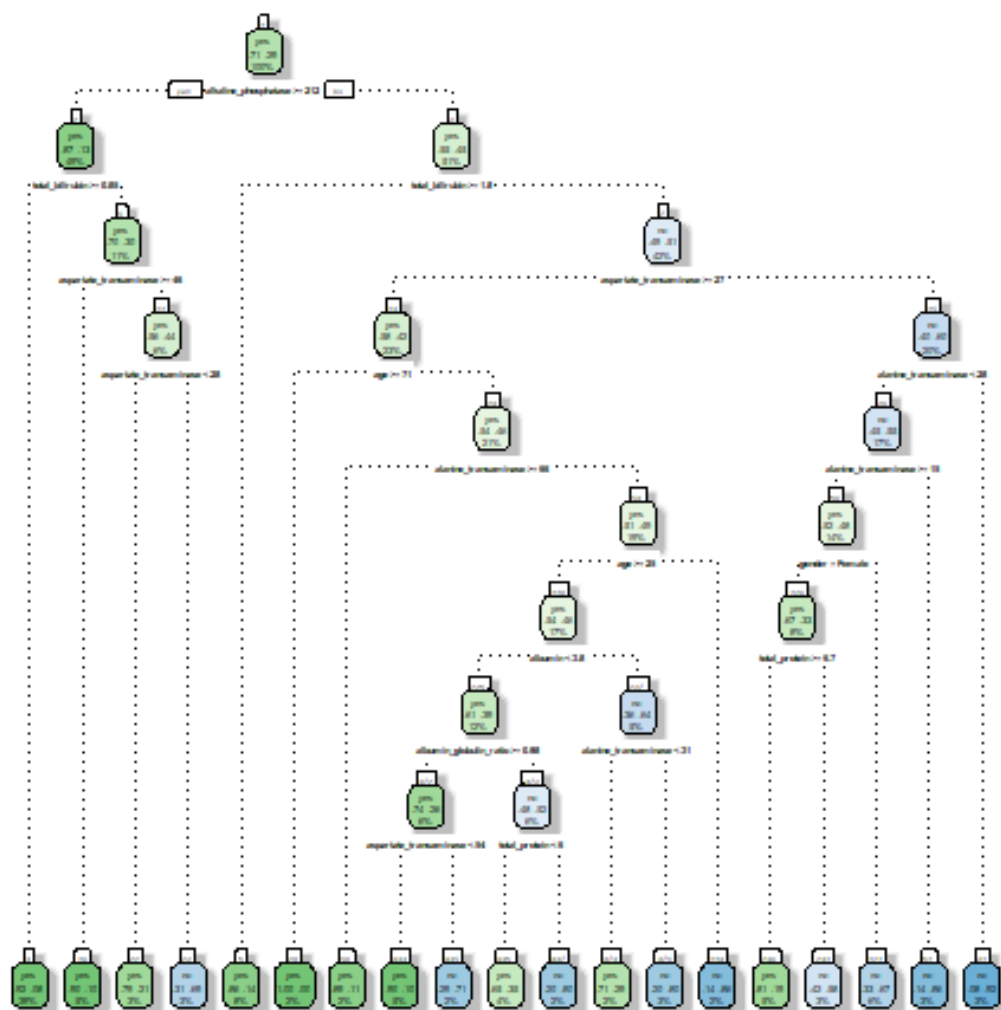


Figure 9: png

%%R

prediccion1

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
no  no yes yes yes yes  no yes  no yes yes yes yes  no  no yes yes yes yes yes
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
yes yes yes yes  no yes yes yes yes yes yes yes yes yes yes yes yes yes yes yes
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
no yes  no  no yes yes yes  no yes yes yes  no yes yes yes  no yes yes yes yes
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
yes yes yes  no  no yes yes  no yes yes yes  no yes yes  no  no yes yes yes yes
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
yes yes  no yes  no yes yes yes yes yes  no yes  no  no  no yes yes  no yes yes
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
yes yes yes yes yes  no  no yes yes yes  no  no  no yes  no yes yes yes  no yes
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
yes yes yes yes  no yes yes yes  no yes yes  no yes yes yes yes yes yes yes yes
141
yes
Levels: yes no
```

%%R

matriz_confusion1

Confusion Matrix and Statistics

```
      Reference
Prediction yes no
yes      80 25
no       22 14
```

```
Accuracy : 0.6667
95% CI : (0.5824, 0.7437)
No Information Rate : 0.7234
P-Value [Acc > NIR] : 0.9431
```

```
Kappa : 0.1468
```

```
Mcnemar's Test P-Value : 0.7705
```

```
Sensitivity : 0.7843
Specificity : 0.3590
Pos Pred Value : 0.7619
Neg Pred Value : 0.3889
Prevalence : 0.7234
```

```
Detection Rate : 0.5674
Detection Prevalence : 0.7447
Balanced Accuracy : 0.5716
```

```
'Positive' Class : yes
```

Debido a la alta complejidad de estos arboles le vamos a hacer un proceso de podado, para ello podemos hacer uso tanto del parámetro `cp` o directamente manipulando los hiperparametros de los modelos.

Arbol podado con un profundidad maxima de 4:

```
%%R

set.seed(0)
datos_entreno2<-sample_frac(ilpd,0.75) # Separamos los datos de
→ entrenamiento
datos_test2<-setdiff(ilpd, datos_entreno2) # Separamos los datos de test

arbol_1<-rpart(diseased~.,data=datos_entreno2, maxdepth=4, method =
→ "class") # Cambiamos la profundidad
rpart.plot(arbol_1,
           extra = 104,          # show fitted class, probs, percentages
           box.palette = "GnBu", # color scheme
           branch.lty = 3,       # dotted branch lines
           shadow.col = "gray",  # shadows under the node boxes
           nn = TRUE)
```

Con esto hemos conseguido un modelo mucho más simple que el anterior sin podado. El cual es más facil de interpretar.

En estos gráficos, cada uno de los rectángulos representa un nodo de nuestro árbol, con su regla de clasificación.

Cada nodo está coloreado de acuerdo a la categoría mayoritaria entre los datos que agrupa. Esta es la categoría que ha predicho el modelo para ese grupo.

Dentro del rectángulo de cada nodo se nos muestra qué proporción de casos pertenecen a cada categoría y la proporción del total de datos que han sido agrupados allí.

Estas proporciones nos dan una idea de la precisión de nuestro modelo al hacer predicciones.

Calculamos las predicciones y la matriz de confusión:

```
%%R

prediccion2 <-predict(arbol_1,newdata=datos_test2,type="class")
matriz_confusion2<-confusionMatrix(prediccion2,datos_test2[["diseased"]])
```

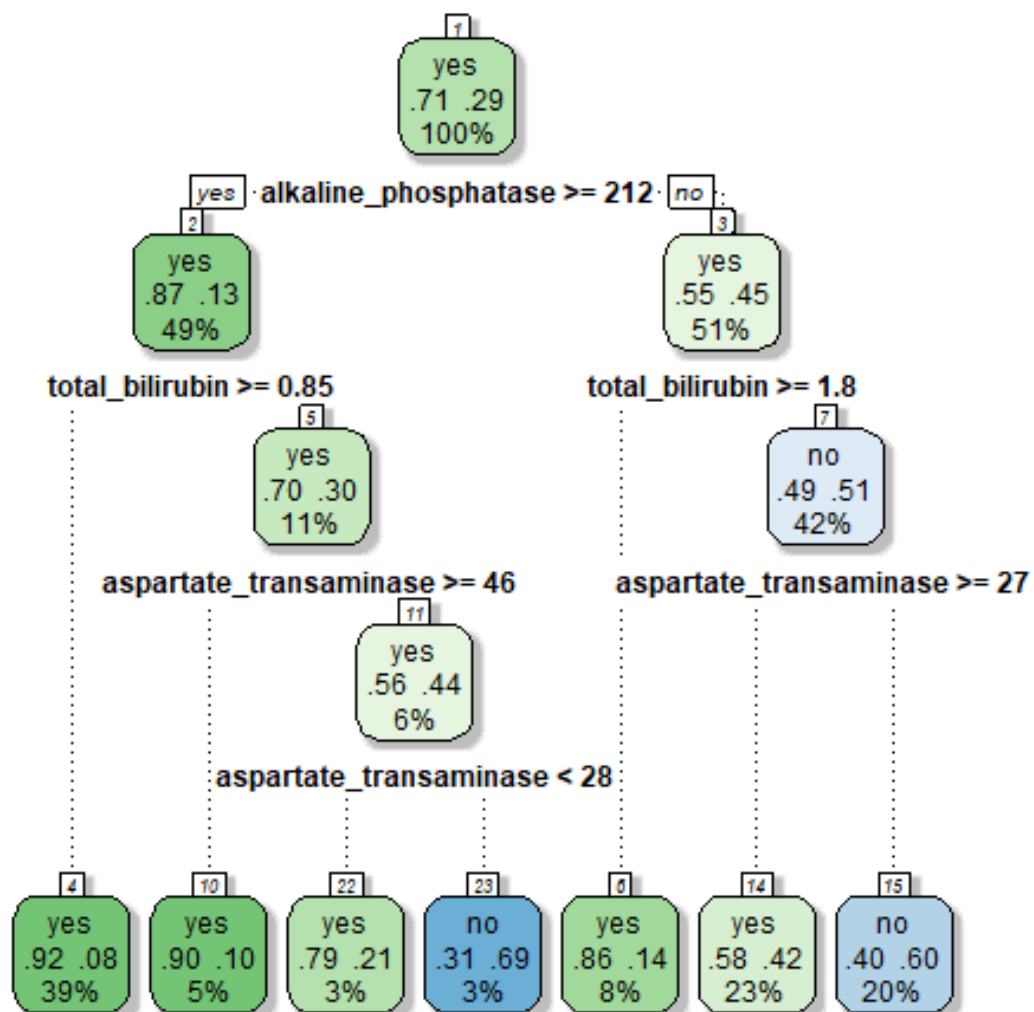


Figure 10: png

%%R

prediccion2

```
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
no  no yes yes yes yes  no yes  no yes yes yes yes  no  no yes yes yes yes yes
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
yes yes yes yes  no yes yes yes yes yes yes yes yes yes yes yes yes yes yes yes
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
no yes yes  no yes yes yes  no yes yes yes yes yes  no yes yes yes yes yes yes
61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
yes yes yes  no  no yes yes  no yes yes yes yes yes yes yes yes yes yes yes yes
81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
yes yes  no yes yes yes yes yes yes yes yes yes  no yes  no yes yes  no yes yes
101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
yes yes yes yes yes yes  no yes yes yes  no  no  no yes  no yes yes yes yes yes
121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
yes yes yes yes yes yes yes yes  no yes yes  no yes yes yes yes yes yes yes yes
141
yes
Levels: yes no
```

%%R

matriz_confusion2

Confusion Matrix and Statistics

```
      Reference
Prediction yes no
yes      87 29
no       15 10
```

```
Accuracy : 0.6879
95% CI : (0.6045, 0.7633)
No Information Rate : 0.7234
P-Value [Acc > NIR] : 0.84963
```

```
Kappa : 0.123
```

```
Mcnemar's Test P-Value : 0.05002
```

```
Sensitivity : 0.8529
Specificity : 0.2564
Pos Pred Value : 0.7500
Neg Pred Value : 0.4000
Prevalence : 0.7234
```

```
Detection Rate : 0.6170
Detection Prevalence : 0.8227
Balanced Accuracy : 0.5547
```

```
'Positive' Class : yes
```

Se puede ver como hemos mejorado un poco la precisión de la predicción simplemente cambiando la profundidad máxima. Cabe remarcar que el algoritmo de los CART utilizan el índice de Gini como criterio de división.

4.4.2 4.4.2. Algoritmo C5.0 con R

Cargamos el paquete específico del Arbol de clasificación C5.0

```
%%R

# install.packages("C50",dependencies=TRUE)

library(C50)
```

Realizamos la partición en datos de entrenaminento y test:

```
%%R

set.seed(0)
tamano_total<-nrow(ilpd)
tamano_entreno<-round(tamano_total*0.75)
datos_indices<-sample(1:tamano_total,size = tamano_entreno)
datos_entreno<-ilpd[datos_indices,]
datos_test<-ilpd[-datos_indices,]
```

Las siguientes proporciones deberían de ser relativamente similares para que los arboles den unos buenos resultados:

```
%%R

round(table(datos_entreno$diseased)/nrow(datos_entreno), 3)
```

```
yes    no
0.709 0.291
```

```
%%R

round(table(datos_test$diseased)/nrow(datos_test), 3)
```



```
      yes      no  
0.726 0.274
```

Ejecución del modelo de clasificación C5.0

```
%%R  
  
modeloC50 <- C5.0(diseased~.,data=datos_entreno, trials=1, rules=FALSE)
```

Información del modelo creado

```
%%R  
  
summary(modeloC50)
```

```
Call:  
C5.0.formula(formula = diseased ~ ., data = datos_entreno, trials = 1, rules  
= FALSE)
```

```
C5.0 [Release 2.07 GPL Edition]      Wed Oct 12 23:37:09 2022  
-----
```

Class specified by attribute `outcome'

Read 437 cases (11 attributes) from undefined.data

Decision tree:

```
direct_bilirubin > 0.9: yes (135/9)  
direct_bilirubin <= 0.9:  
  ...alanine_transaminase > 65:  
    ...albumin <= 3.9: yes (35/1)  
    :   albumin > 3.9:  
    :     ...aspartate_transaminase <= 99: no (4/1)  
    :       aspartate_transaminase > 99: yes (4)  
  alanine_transaminase <= 65:  
    ...alkaline_phosphatase > 211:  
      ...total_bilirubin <= 0.8:  
        :   ...gender = Female:  
        :     :   ...age <= 39: yes (2)  
        :     :     age > 39: no (5)  
        :     :   gender = Male:  
        :     :     ...age <= 13: no (3)  
        :     :       age > 13:  
        :     :         ...albumin_globulin_ratio <= 0.55: no (2)  
        :     :           albumin_globulin_ratio > 0.55: yes (25/4)  
        :     total_bilirubin > 0.8:
```

```

:   ...age > 37: yes (27/1)
:   age <= 37:
:   ...total_bilirubin > 1.6: no (2)
:   total_bilirubin <= 1.6:
:   ...alanine_transaminase <= 23: no (2)
:   alanine_transaminase > 23: yes (10/1)
alkaline_phosphatase <= 211:
:...direct_bilirubin <= 0.1:
:   ...gender = Male: yes (21/5)
:   gender = Female:
:   ...alkaline_phosphatase <= 168: no (4)
:   alkaline_phosphatase > 168: yes (8/2)
direct_bilirubin > 0.1:
:...total_bilirubin <= 0.7:
:   ...alanine_transaminase > 33:
:   :   ...aspartate_transaminase > 64: yes (3)
:   :   aspartate_transaminase <= 64:
:   :   :   ...aspartate_transaminase <= 41: yes (2)
:   :   :   aspartate_transaminase > 41: no (2)
:   :   alanine_transaminase <= 33:
:   :   ...albumin_globulin_ratio > 0.9: no (19)
:   :   albumin_globulin_ratio <= 0.9:
:   :   ...gender = Female:
:   :   :   ...alkaline_phosphatase <= 176: yes (3/1)
:   :   :   :   alkaline_phosphatase > 176: no (3)
:   :   :   gender = Male:
:   :   :   ...age <= 34: no (2)
:   :   :   age > 34: yes (5/1)
total_bilirubin > 0.7:
:...gender = Female:
:   ...alanine_transaminase > 29: no (7)
:   alanine_transaminase <= 29:
:   ...total_bilirubin > 0.9: no (7/1)
:   total_bilirubin <= 0.9:
:   ...albumin <= 4.3: yes (23/3)
:   albumin > 4.3: no (4/1)
gender = Male:
:...aspartate_transaminase <= 25:
:   ...albumin <= 3.9: no (18/1)
:   albumin > 3.9: yes (5/1)
aspartate_transaminase > 25:
:...aspartate_transaminase > 70: no (4)
:   aspartate_transaminase <= 70:
:   ...albumin_globulin_ratio <= 0.58: no (3)
:   albumin_globulin_ratio > 0.58: yes (38/11)

```

Evaluation on training data (437 cases):

```

      Decision Tree
-----
Size      Errors

    33    44(10.1%)    <<

(a)   (b)    <-classified as
----  ----
 306    4    (a): class yes
  40   87    (b): class no

```

Attribute usage:

```

100.00% direct_bilirubin
 69.11% alanine_transaminase
 59.27% alkaline_phosphatase
 51.72% total_bilirubin
 43.94% gender
 22.88% albumin_globulin_ratio
 21.28% albumin
 19.45% age
 18.99% aspartate_transaminase

```

Time: 0.0 secs

Podemos ver el gráfico del modelo:

```

%%R

plot(modeloC50)

```

Calculamos las predicciones del modelo:

```

%%R

(prediccion<-predict(modeloC50, newdata = datos_test,type="class"))

```

```

[1] no  yes yes yes yes yes no  yes yes yes yes no  yes no  yes yes yes no
[19] yes yes yes yes yes yes yes yes yes yes no  yes yes yes yes yes yes
[37] yes yes yes yes yes yes yes yes yes no  yes no  no  yes yes yes no  yes
[55] yes yes yes yes yes yes yes yes yes yes yes yes yes yes yes yes yes
[73] yes yes yes yes no  yes no  no  yes yes yes yes yes yes yes yes yes
[91] yes yes yes yes yes yes yes no  yes no  yes yes no  yes yes yes yes yes
[109] yes yes yes yes yes no  yes no  yes no  yes no  yes yes no  yes yes yes
[127] yes yes yes no  yes yes yes no  yes yes no  yes yes yes yes yes yes yes

```

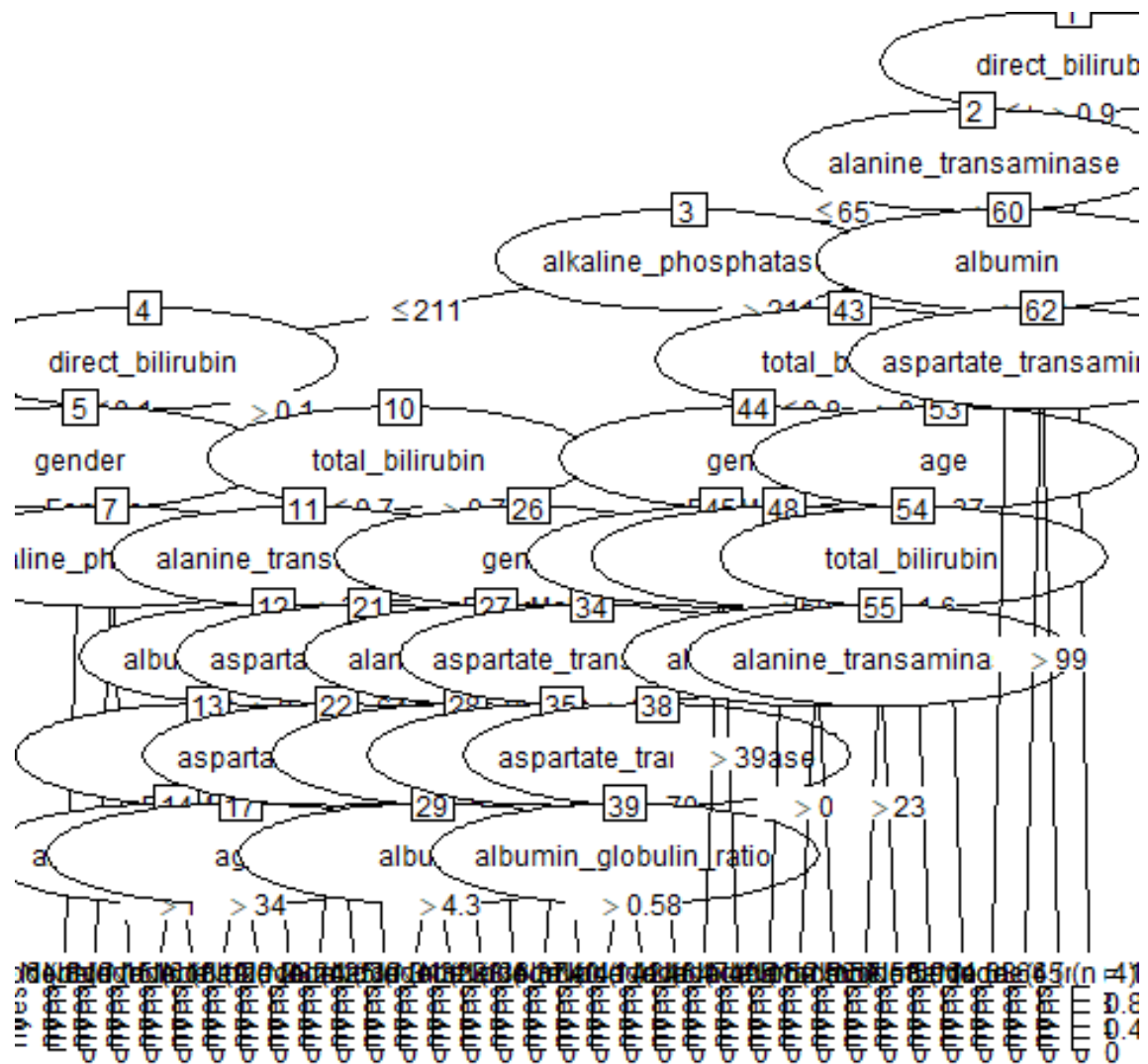


Figure 11: png

```
[145] yes yes
Levels: yes no
```

Calculamos la matriz de confusión:

```
%%R

(matriz_confusion<-table(predicho=prediccion, real=datos_test$diseased))

      real
predicho yes no
yes      92 30
no       14 10
```

Calculamos la tasa de acierto en la clasificación (TAC) obtenida parte del modelo, es decir el porcentaje de clasificaciones correctas.

```
%%R

100*sum(diag(matriz_confusion))/sum(matriz_confusion)
```

```
[1] 69.86301
```

TAC = 0.6986

Calculamos la tasa de error de clasificación (TEC = 1 - TAC) cometido por el modelo, que es el porcentaje de clasificaciones incorrectas

```
%%R

error_clas<-round(mean(prediccion != datos_test$diseased),3)
paste(
  "El error de clasificacion es
  → del:",100*error_clas,"%.",sum(prediccion==datos_test$diseased),"clasificaciones
  → correctas de un total de",length(prediccion)
)
```

```
[1] "El error de clasificacion es del: 30.1 %. 102 clasificaciones correctas de un total de 334"
```

TEC = 0.301

Ahora vamos a podar el árbol con la librería C5.0

Seleccionamos la submuestra del 75% de los datos

```
%%R
```

```
set.seed(0)
tamano_total<-nrow(ilpd)
tamano_entreno<-round(tamano_total*0.75)
datos_indices<-sample(1:tamano_total,size = tamano_entreno)
datos_entreno<-ilpd[datos_indices,]
datos_test<-ilpd[-datos_indices,]
```

Las siguientes proporciones deberían de ser relativamente similares para que los arboles den unos buenos resultados:

```
%%R
```

```
round(table(datos_entreno$diseased)/nrow(datos_entreno), 3)
```

```
   yes    no
0.709 0.291
```

```
%%R
```

```
round(table(datos_test$diseased)/nrow(datos_test), 3)
```

```
   yes    no
0.726 0.274
```

Ejecución del arbol de clasificación podado con la libreria C5.0

```
%%R
```

```
modeloC50<-C5.0(diseased~.,data=datos_entreno, trials=1, rules=FALSE, control
↪ = C5.0Control(minCases = 10, earlyStopping = TRUE))
```

Información del modelo creado

```
%%R
```

```
summary(modeloC50)
```

Call:

```
C5.0.formula(formula = diseased ~ ., data = datos_entreno, trials = 1, rules
= FALSE, control = C5.0Control(minCases = 10, earlyStopping = TRUE))
```

 Class specified by attribute `outcome'

Read 437 cases (11 attributes) from undefined.data

Decision tree:

```

direct_bilirubin > 0.9: yes (135/9)
direct_bilirubin <= 0.9:
: ...alanine_transaminase > 65: yes (43/4)
  alanine_transaminase <= 65:
: ...alkaline_phosphatase > 211: yes (78/20)
  alkaline_phosphatase <= 211:
: ...direct_bilirubin <= 0.1: yes (33/11)
  direct_bilirubin > 0.1:
: ...total_bilirubin <= 0.7: no (39/11)
  total_bilirubin > 0.7:
: ...gender = Female:
  : ...total_bilirubin <= 0.8: yes (20/5)
  :   total_bilirubin > 0.8: no (21/7)
  gender = Male:
  : ...aspartate_transaminase <= 25: no (23/5)
  :   aspartate_transaminase > 25: yes (45/18)
  
```

Evaluation on training data (437 cases):

```

      Decision Tree
      -----
Size      Errors

      9    90(20.6%)    <<

(a)      (b)      <-classified as
-----  -----
287      23      (a): class yes
67       60      (b): class no
  
```

Attribute usage:

```

100.00% direct_bilirubin
 69.11% alanine_transaminase
 59.27% alkaline_phosphatase
 33.87% total_bilirubin
 24.94% gender
  
```

15.56% aspartate_transaminase

Time: 0.0 secs

Gráfico del modelo:

%%R

```
plot(modeloC50)
```

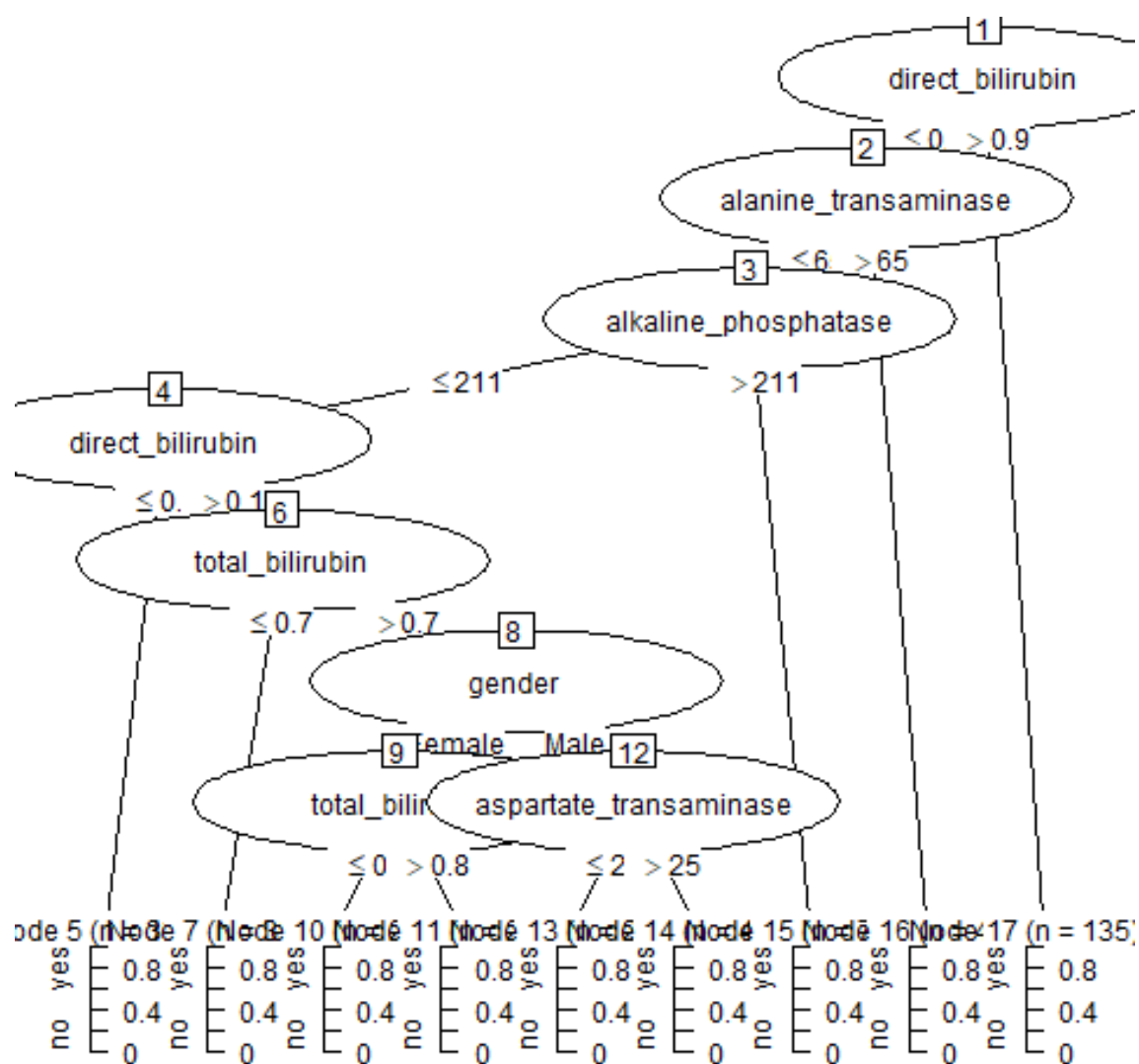


Figure 12: png

Calculamos las predicciones:


```
%%R
```

```
(prediccion<-predict(modeloC50, newdata = datos_test,type="class"))
```

```
[1] no no yes yes yes yes yes yes yes no yes yes yes yes yes yes yes yes
[19] yes yes yes yes yes yes yes yes yes yes no yes yes yes yes yes yes yes
[37] yes yes yes yes yes yes yes yes yes no yes no no yes yes yes yes yes
[55] yes yes yes yes yes yes yes yes yes no no yes no yes no no yes yes
[73] no yes yes yes yes yes no yes yes yes yes yes yes yes yes no yes no
[91] yes no yes yes yes yes yes no yes no yes yes no yes no yes yes yes
[109] yes yes yes yes yes yes yes no yes yes no no yes yes yes yes yes yes
[127] yes yes yes yes yes yes yes no yes yes no yes yes yes yes yes yes
[145] yes yes
Levels: yes no
```

Calculamos la matriz de confusión:

```
%%R
```

```
(matriz_confusion<-table(predicho=prediccion, real=datos_test$diseased))
```

```
      real
predicho yes no
yes      92 28
no       14 12
```

Porcentaje de clasificados correctamente:

```
%%R
```

```
100*sum(diag(matriz_confusion))/sum(matriz_confusion)
```

```
[1] 71.23288
```

TAC = 0.7123

Error de clasificación :

```
%%R
```

```
error_clas<-round(mean(prediccion != datos_test$diseased),3)
paste(
  "El error de clasificacion es
  → del:",100*error_clas,"%.",sum(prediccion==datos_test$diseased),"clasificaciones
  → correctas de un total de",length(prediccion)
)
```

```
[1] "El error de clasificacion es del: 28.8 %. 104 clasificaciones correctas de un total
```

TEC = 0.288

4.4.3 4.4.3. Algoritmo CART en R con mlr3

No es necesario preprocesar los datos para árboles (dummy y normalización).

```
%%R  
  
# install.packages('mlr3extralearners')  
  
library(mlr3)  
library(mlr3learners)
```

```
R[write to console]:  
Attaching package: 'mlr3'
```

```
R[write to console]: The following object is masked from 'package:skimr':
```

```
partition
```

Creamos la tarea de clasificación:

```
%%R  
  
ILPD_task <- as_task_classif(ilpd , target = "diseased")
```

Partimos el data-set en parte de test y parte de train:

```
%%R  
  
res_desc <- rsmp("holdout" , ratio=0.75)  
  
set.seed(0)  
  
res_desc$instantiate(ILPD_task)
```

Definimos el método de aprendizaje:

```
%%R  
  
tree_learner <- lrn("classif.rpart" , maxdepth=4)
```

Entrenamos y evaluamos el modelo:

```
%%R  
  
tree_resample<-resample(task = ILPD_task, learner =  
  ↪ tree_learner, resampling = res_desc, store_models = TRUE)
```

```
INFO [23:37:39.510] [mlr3] Applying learner 'classif.rpart' on task 'ilpd' (iter 1/1)
```

Calculamos las predicciones:

```
%%R

tree_test<-tree_resample$predictions()
tree_test[[1]]
```

```
<PredictionClassif> for 146 observations:
```

row_ids	truth	response
4	yes	no
9	no	no
11	yes	yes

575	yes	yes
577	yes	yes
583	no	yes

Calculamos la accuracy (que es la TAC):

```
%%R

tree_acc <- tree_resample$aggregate(msr("classif.acc"))

tree_acc
```

```
classif.acc
0.6917808
```

Visualizamos el modelo:

Primero obtendremos la expresion en texto del modelo, luego su gráfico.

```
%%R

tree_learner<-tree_resample$learners[[1]]
tree_learner$model
```

```
n= 437
```

```
node), split, n, loss, yval, (yprob)
* denotes terminal node
```

```
1) root 437 127 yes (0.70938215 0.29061785)
  2) alkaline_phosphatase>=211.5 216 28 yes (0.87037037 0.12962963)
    4) total_bilirubin>=0.85 169 14 yes (0.91715976 0.08284024) *
```

```

5) total_bilirubin < 0.85 47 14 yes (0.70212766 0.29787234)
10) aspartate_transaminase >= 45.5 20 2 yes (0.90000000 0.10000000) *
11) aspartate_transaminase < 45.5 27 12 yes (0.55555556 0.44444444)
22) aspartate_transaminase < 27.5 14 3 yes (0.78571429 0.21428571) *
23) aspartate_transaminase >= 27.5 13 4 no (0.30769231 0.69230769) *
3) alkaline_phosphatase < 211.5 221 99 yes (0.55203620 0.44796380)
6) total_bilirubin >= 1.75 36 5 yes (0.86111111 0.13888889) *
7) total_bilirubin < 1.75 185 91 no (0.49189189 0.50810811)
14) aspartate_transaminase >= 26.5 99 42 yes (0.57575758 0.42424242) *
15) aspartate_transaminase < 26.5 86 34 no (0.39534884 0.60465116) *

```

```
%%R
```

```
rpart.plot(tree_learner$model)
```

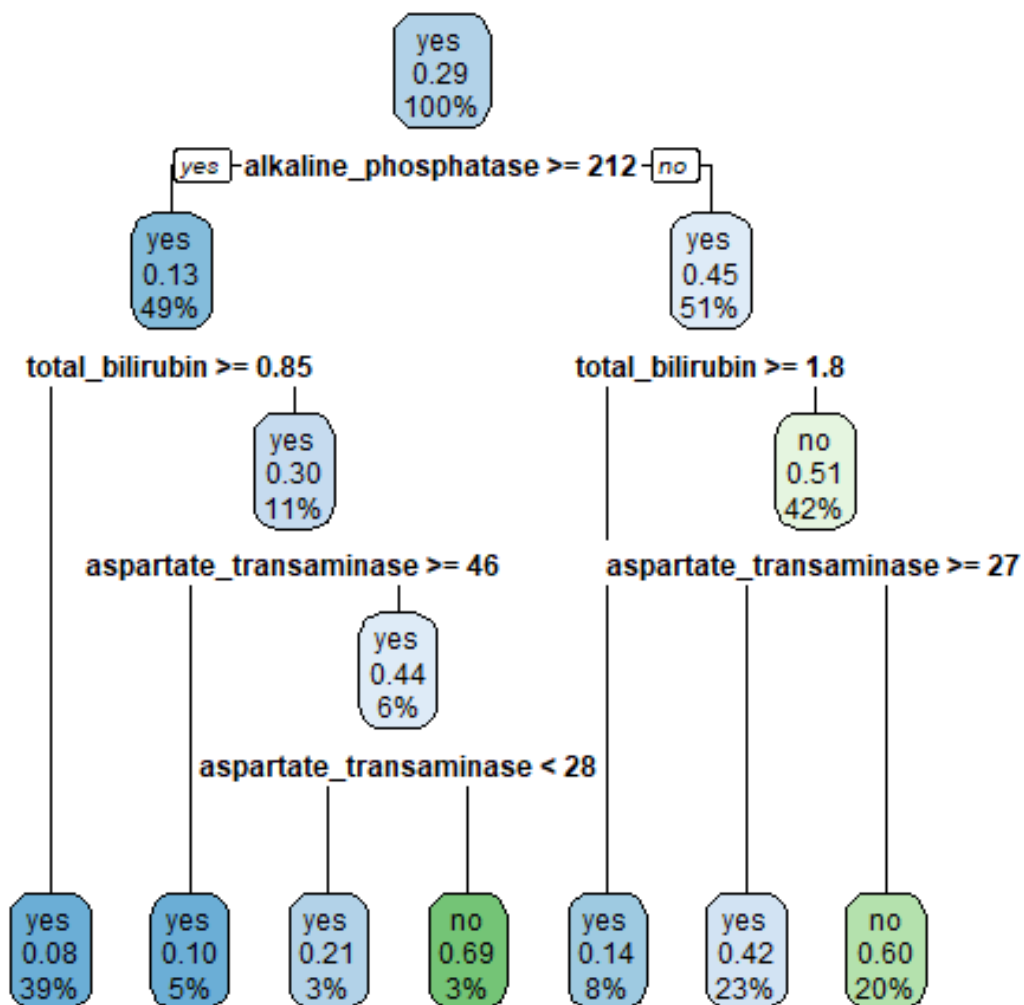


Figure 13: png

4.4.4 4.4.4. Algoritmo C5.0 en R con mlr3

```
%%R

library(mlr3)
library(mlr3learners)
library(mlr3extralearners)
```

Cambiamos el tipo de los datos a numerico porque si no el algoritmo no funciona.

```
%%R

ilpd$age<-as.numeric(ilpd$age)
ilpd$alkaline_phosphatase<-as.numeric(ilpd$alkaline_phosphatase)
ilpd$alanine_transaminase<-as.numeric(ilpd$alanine_transaminase)
ilpd$aspartate_transaminase<-as.numeric(ilpd$aspartate_transaminase)
```

Creamos la tarea de clasificación:

```
%%R

ILPD_task<-as_task_classif(ilpd,target = "diseased")
```

Definimos el método de evaluación:

```
%%R

res_desc<-rsmp("holdout",ratio=0.75)
set.seed(0)
res_desc$instantiate(ILPD_task)
```

Definimos el método de aprendizaje:

```
%%R

tree_learner<-lrn("classif.C50")
```

Entrenamos y evaluamos el modelo:

```
%%R

tree_resample<-resample(task = ILPD_task, learner = tree_learner,
  ↪ resampling = res_desc, store_models = TRUE)
```

```
INFO [23:37:50.846] [mlr3] Applying learner 'classif.C50' on task 'ilpd' (iter 1/1)
```

Obtenemos la predicciones del modelo:

```
%%R

tree_test<-tree_resample$predictions()
tree_test[[1]]
```

```
<PredictionClassif> for 146 observations:
  row_ids truth response
      4   yes       no
      9    no       yes
     11   yes       yes
---
    575   yes       yes
    577   yes       yes
    583    no       yes
```

Calculamos la accuracy (TAC) del modelo:

```
%%R

tree_acc<-tree_resample$aggregate(msr("classif.acc"))

tree_acc
```

```
classif.acc
0.6986301
```

Visualizamos el modelo :

```
%%R

tree_learner<-tree_resample$learners[[1]]
tree_learner$model
```

```
Call:
C50::C5.0.formula(formula = f, data = data, control = ctrl)
```

```
Classification Tree
Number of samples: 437
Number of predictors: 10
```

```
Tree size: 35
```

```
Non-standard options: attempt to group attributes
```

Como vemos no da plena información sobre la estructura del modelo.

Ahora hemos intentado graficar el modelo, pero nos sale un error.

```
%%R  
  
# plot(tree_learner$model)
```

NULL

4.4.5 Comparación entre R y mlr3

Algoritmo RPART con Rpart (R) Inicializamos la semilla y dividimos el data frame en datos de muestra de entrenamiento y muestra de test con funciones de dplyr. Después creamos el modelo con rpart y le metemos la variable respuesta, el data frame de entrenamiento, el método y el coeficiente de partición que nos poda el árbol. Posteriormente, sacamos las predicciones a partir del modelo, el data frame de test y el type. Por último, lo que hacemos es sacar la matriz de confusiones, que nos da una medida de cómo de bueno es nuestro modelo. Más concretamente, en este problema, nos interesa la accuracy.

Aunque no se pide, para poder podar los árboles en este caso, lo que se puede hacer es ajustar los hiperparámetros como pueden ser la profundidad máxima. Y en este tipo de árboles se utiliza GINI como criterio de división.

Algoritmo C5.0 con C5.0 (R) Lo primero que hacemos es crear la semilla y dividir el data frame en datos de entrenamiento y datos de test. Hecho esto, hacemos el modelo con C5.0 y le metemos la variable respuesta, el data frame de entrenamiento, si queremos que nos saque las reglas y el número de intentos (este es un argumento que nos es útil en un proceso de boosting). Para ver la información del modelo creado lo que hacemos es uso de la función summary y lo graficamos con plot. Por último, lo que hacemos es como en el caso anterior sacar las predicciones con predict y la matriz de confusiones, en este caso la hemos sacado con table, que no te saca como en el caso anterior otros estadísticos a parte de la accuracy como puede ser Kappa. En este caso, no es así y para sacar la accuracy, tenemos que calcularla a mano.

Para podar, este tipo de árboles lo que se puede hacer es usar control dentro de la función que crea el modelo, en nuestro caso hemos puesto un mínimo de casos y una parada rápida. La diferencia principal entre los algoritmos es que el C5.0 utiliza la entropía como criterio de división.

Algoritmo RPART con Mlr3 Lo primero que hacemos es crear una tarea de clasificación, donde tenemos los datos y el problema a resolver. Después le metemos el método de evaluación, en este caso holdout al 75% y dividimos los datos en test y entrenamiento. Posteriormente, definimos el método de aprendizaje con un learner donde se contiene el método a utilizar. Lo entrenamos y evaluamos con resample y los distintos objetos que hemos creado en pasos anteriores. Para sacar las predicciones, la accuracy y visualizar el modelo, lo que debemos hacer es llamar a los elementos del modelo pertinentes, y no como en R que debemos calcularlos a mano.

Algoritmo C5.0 con Mlr3 Lo primero que hacemos es cambiar las variables que son enteras a numéricas. Después, de la misma forma que para rpart creamos la tarea donde metemos

los datos y le indicamos la variable respuesta. Definimos el método de evaluación, que como en el anterior usamos un holdout al 75% y dividimos los datos en entrenamiento y test. Posteriormente, definimos el método de aprendizaje con un learner y en este caso es “classif.c50”. A partir de aquí, procedemos de la misma forma que en el caso anterior con las predicciones, accuracy y visualización del modelo.

A modo de resumen, con R es bastante intuitivo el proceso, pero tiene el problema de que cuando quieres sistematizar el análisis, ya que este tipo de árboles dependen mucho de como sea la partición en muestra entrenamiento y muestra test, es realmente complicado. También podemos ver como el ajuste de hiperparámetros o la poda de los árboles es bastante manual, por lo que esto no es demasiado eficiente.

En general, en mlr3 está todo el proceso mucho más optimizado y de forma más automática. Además el preproceso o el ajuste de hiperparámetros es mucho más fácil que con R. Podemos decir que mlr3 es una familia de paquetes que forman un ecosistema muy avanzado para algoritmos de machine learning, aun así es algo complicado acostumbrarnos a la sintaxis del paquete y a su filosofía. Después de haber trabajado un tiempo con este grupo de paquetes, está claro que es mucho mejor que hacerlo con R y algunas librerías sueltas.

A modo de resumen, en R puedes hacer problemas básicos, pero a la hora de hacer un proceso completo, lo idóneo es usar la librería mlr3 a pesar de que la curva de aprendizaje y aclimatación al paquete por su filosofía y sintaxis pueda parecer algo complicada al principio. Todo ello sin decir que con mlr3 todos los algoritmos se programan de una forma similar, ahorrando líneas de código con respecto a hacerlo con rpart y C5.0.

En ninguno de los casos, hace falta un preproceso de los datos.

4.4.6 4.4.6. Comparación de resultados

Rpart con R

Sin podar:

5 predicciones primeras: NO NO YES YES YES

Accuracy modelo: 0.6667

Primeros índices: 1 2 3 5 6

Podado:

5 predicciones primeras: NO NO YES YES YES

Accuracy modelo: 0.6879

Primeros índices: 1 2 3 5 6

C5.0 con R

Sin podar:

5 predicciones primeras: NO YES YES YES YES

Accuracy modelo: 0.6986

Primeros índices: 1 2 3 5 6

Podado:

5 predicciones primeras: NO NO YES YES YES

Accuracy modelo: 0.7123

Primeros índices: 1 2 3 5 6

Rpart con Mlr3

5 predicciones primeras: NO NO YES YES YES

Accuracy modelo: 0.6917

Primeros índices: 4 9 11 12 13

C5.0 con Mlr3

5 predicciones primeras: NO YES YES YES YES

Accuracy modelo: 0.6986

Primeros índices: 4 9 11 12 13

Se puede ver como dan cosas distintas con R y Mlr3 aunque la diferencia es mínima. También vemos diferencias entre algoritmos ya que parece que dan una mayor precisión los algoritmos c5.0 además de una mayor eficiencia computacional. En cuanto a los índices, podemos ver como Mlr3 coge distintos de los que toma R. También tenemos que remarcar que la semilla es la misma en todos los casos.

4.5 Arboles de clasificación en Python

4.5.1 Arboles de clasificación: teoría

Vamos a considerar el siguiente tipo de arboles (matematicos):

```
from IPython.display import Image
Image(filename='arbol.jpg', width = 1000, height = 400)
```

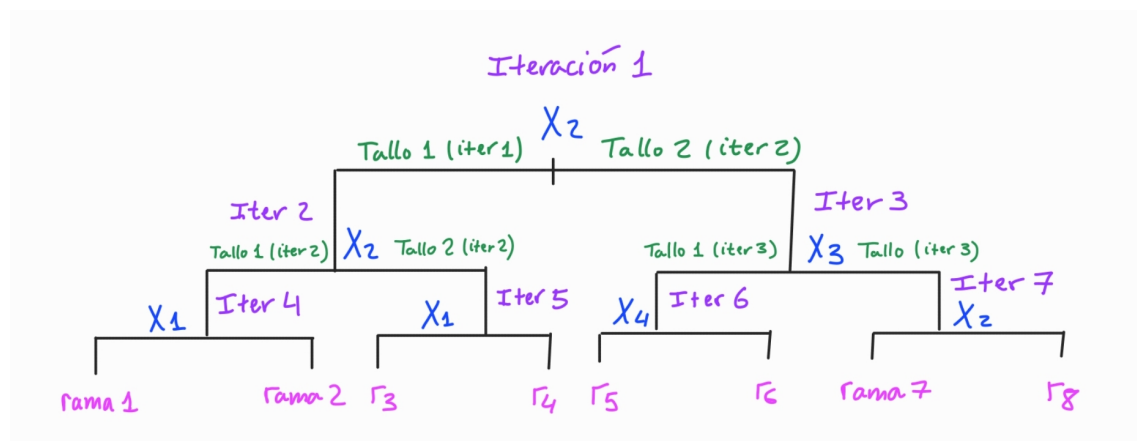


Figure 14: jpeg

Es importante tener en cuenta los elementos que estan reflejados, pues los usaremos posteriormente.

Los arboles estan compuestos de iteraciones, que a su vez cada una de ellas se dividen en dos tallos. La union de tallos de distintas iteraciones da lugar a las ramas del arbol.

4.5.1.1 Arboles de clasificación La idea de los algoritmos de arboles de clasificacion es segmentar las observaciones de los predictores X_1, \dots, X_p para predecir el valor de la respuesta Y en base a esa informacion segmentada. Es algo asi como predecir Y por grupos/segmentos.

4.5.2 Definicion formal de los arboles de clasificación:

Elementos Básicos

- Tenemos unos predictores X_1, \dots, X_p y una variable respuesta **categorica** Y
- Tenemos un arbol T de la forma del expuesto en la imagen con $m - 1$ iteraciones y m ramas.
- r_{ht} es la rama h del arbol con t iteraciones.
- Cada iteracion del arbol tiene asociado uno de los predictores X_1, \dots, X_n
- Cada iteracion del arbol tiene dos tallos (tallo 1 (izquierdo) y tallo 2 (derecho)).
- En cada tallo de una iteracion se define un intervalo.
- I_{lt} es el intervalo asociado al tallo l de la iteracion t
- Para simplificar el problema consideraremos $I_{1t} = (-\infty, s_t)$ y $I_{2t} = [s_t, \infty]$ donde s_t es llamado punto de corte de la iteracion t del arbol
- R_{ht} es la region (rectangulo n -dimensional) definida por la rama h de un arbol con t iteraciones

4.5.2.0.1 Criterio de prediccion de la variable respuesta Dada una nueva observacion $x_{new} = (x_{new,1}, x_{new,2}, \dots, x_{new,p})$ la idea es predecir y_{new} como sigue:

Sea $f_{r, R_{ht}}$ la frecuencia relativa de la clase/grupo r en la rama h de un arbol con t iteraciones.

Es decir, es la proporcion de individuos de la muestra de entrenamiento que caen en la rama h de un arbol con t iteraciones que pertenecen a la clase r (es decir, para los que $Y = r$) :

$$f_{r, R_{ht}} = \frac{\# \{i / x_i \in R_{ht} \text{ y } y_i = r\}}{\# \{i / x_i \in R_{ht}\}}$$

Donde: $r \in \text{Rango}(Y) = \{0, 1, \dots, c - 1\}$

Si $x_{new} \in R_{ht} \Rightarrow x_{new}$ es clasificado en la clase/grupo mayoritaria (mas frecuente) en la rama h (r_h)

Por tanto:

Si $r_{R_{ht}}^* = \arg \underset{r}{Max} (f_{r,R_{ht}})$, entonces :

$$Si \ x_{new} \in R_h \Rightarrow \hat{y}_{new} = r_{R_{ht}}^*$$

Observación:

Definida la region R_{ht} , es relativamente sencillo resolver el problema $\underset{r}{Max} (f_{r,R_{ht}})$ y asi obtener $r_{R_{ht}}^*$

4.5.2.0.2 Objetivo : Usando la tasa de error de clasificacion como métrica a optimizar Definimos el **error de entrenamiento de la rama h de un arbol de clasificacion con t iteraciones** como la tasa de error de clasificacion para las observaciones de entrenamiento que caen en la rama h de dicho arbol, es decir, como:

$$TEC(R_{ht}) = 1 - f_{r_{R_{ht}}^*, R_{ht}}$$

Observación:

$f_{r_{R_{ht}}^*, R_{ht}}$ es la proporcion de individuos de la muestra de entrenamiento que caen en la rama h de un arbol con t iteraciones que son de la clase/grupo $r_{R_{ht}}^*$ (el valor de la variable respuesta para ellos es $r_{R_{ht}}^*$)

Como el modelo clasifica a los que caen en esa rama como de la clase $r_{R_{ht}}^*$, es decir, como la predicion de la respuesta para todo individuo que pertenezca a esa rama es $r_{R_{ht}}^*$, por parte del modelo, entonces se tiene lo siguiente:

$f_{r_{R_{ht}}^*, R_{ht}}$ es la proporcion de individuos de la muestra de entrenamiento que caen en la rama h de un arbol con t iteraciones que son correctamente clasificados por el modelo (proporcion de individuos de la region R_{ht} a los que se les ha predicho bien la respuesta).

$TEC(R_{ht})$ es la proporcion de individuos de la muestra de entrenamiento que caen en la rama h de un arbol con t iteraciones (sus observaciones de los predictores pertenecen a R_{ht}) y que han sido clasificados erroneamente. Se les ha clasificado en la clase $r_{R_{ht}}^*$ y su clase era otra diferente, es decir, tenian un valor distinto a $r_{R_{ht}}^*$ para la variable respeusta, que es el valor que el modelo les predice para la respuesta.

Definimos el **error global de entrenamiento de un arbol de clasificación** como la suma de los errores de entrenamiento de las ramas del arbol de clasificación:

$$\sum_{h=1}^m TEC(R_h)$$

El **objetivo** es construir un arbol de regresion con m ramas tal que **minimice el error global de entrenamiento**.

Es decir, formalmente el objetivo es:

$$\underset{R_1, \dots, R_m}{Min} \sum_{h=1}^m TEC(R_h)$$

Pero para escoger las regiones R_1, \dots, R_m que definen las ramas del arbol hay que determinar dos elementos que definen a su vez a las regiones:

1. Qué predictores estan asociados a cada iteracion del arbol \Rightarrow Para cada iteracion i escoger X_j (\$ es decir, escoger j)
2. Qué intervalos estan asociados a cada uno de los dos tallos de cada interaccion \Rightarrow Para cada iteracion i escoger I_{1i} y I_{2i} (es decir, escoger el punto de corte s_i)

Por tanto el porblema a resolver se puede reformular como:

Para cada iteracion i escoger X_j (es decir j) y (I_{1i}, I_{2i}) (es decir s_i) tal que se acaben formando un arbol cuyas ramas definan unas regiones R_1, \dots, R_m que **minimicen** $\sum_{h=1}^m TEC(R_h)$

4.5.2.0.3 Objetivo : Usando el índice de Gini como métrica a optimizar
 Definimos el **error de entrenamiento de la rama h de un arbol de clasificacion con t iteraciones** como el índice de Gini de la respuesta en la rama h del arbol con t iteraciones (índice de gini de la respuesta en la region R_{ht}) , es decir, como:

$$G_{R_{ht}} = \sum_{r=0,1,\dots,c-1} f_{r,R_{ht}} \cdot (1 - f_{r,R_{ht}})$$

Donde: $Rango(Y) = \{0, 1, \dots, c - 1\}$

$G_{R_{ht}}$ toma valores pequeños cuando la frecuencia de una clase $r = 0, 1, \dots$ en la region R_{ht} es alta , y por tanto la del resto baja.

$G_{R_{ht}}$ toma valores altos cuando las frecuencias de las clases se reparten de manera “igualitaria” en la region R_{ht} . Y cuanto mas igualitaria es la reparticion de las classes, mas alto es $G_{R_{ht}}$. Hasta el punto que cuando la reparticion es totalmente igualitaria, esto es, cada clase tiene la misma frecuencia , si hay c clases, cada una tiene una frecuencia relativa de $1/c$ en la region, entonces en indice de Gini alcanza su maximo valor.

Ejemplo:

Para $c = 3$ ($Rango(Y) = \{0, 1, 2\}$)

Si tenemos: $f_{0,R_{ht}} = 0.40$, $f_{1,R_{ht}} = 0.30$ y $f_{2,R_{ht}} = 0.30 \Rightarrow G_{R_{ht}} = 0.66$

Si tenemos: $f_{0,R_{ht}} = 0.80$, $f_{1,R_{ht}} = 0.10$ y $f_{2,R_{ht}} = 0.10 \Rightarrow G_{R_{ht}} = 0.34$

Si tenemos: $f_{0,R_{ht}} = 0.9$, $f_{1,R_{ht}} = 0.05$ y $f_{2,R_{ht}} = 0.05 \Rightarrow G_{R_{ht}} = 0.185$

Teniendo esto en cuenta nos interesan que en cada rama (region R_{ht}) la frecuencia de la clase mayoritaria sea lo mayor posible, y eso equivale a que el indice de Gini sea lo menos posible dentro de cada rama, siguiendo la filosofia empleada con la TEC , donde nos interesaba que $f_{r_{R_{ht}}^*, R_{ht}}$ fuese lo mayor posible en cada rama.

Definimos el **error global de entrenamiento de un arbol de clasificación** como la suma de los errores de entrenamiento de las ramas del arbol de clasificación:

$$\sum_{h=1}^m G_{R_{ht}}$$

El **objetivo** es construir un arbol de regresion con m ramas tal que **minimice el error global de entrenamiento**.

Es decir, formalmente el objetivo es:

$$\underset{R_1, \dots, R_m}{Min} \sum_{h=1}^m G_{R_{ht}}$$

En el fondo minimizar el error de clasificacion de un arbol de clasificacion equivale a minimizar el indice de Gini en las ramas del arbol conjuntamente (a nivel global).

Pero para escoger las regiones R_1, \dots, R_m que definen las ramas del arbol hay que determinar dos elementos que definen a su vez a las regiones:

1. Qué predictores estan asociados a cada iteracion del arbol \Rightarrow Para cada iteracion i escoger X_j (\$ es decir, escoger j)
2. Qué intervalos estan asociados a cada uno de los dos tallos de cada interaccion \Rightarrow Para cada iteracion i escoger I_{1i} y I_{2i} (es decir, escoger el punto de corte s_i)

Por tanto el porblema a resolver se puede reformular como:

Para cada iteracion i escoger X_j (es decir j) y (I_{1i}, I_{2i}) (es decir s_i) tal que se acaben formando un arbol cuyas ramas definan unas regiones R_1, \dots, R_m que **minimicen** $\sum_{h=1}^m G_{R_{ht}}$

4.5.2.1 Algoritmo para la resolucion del problema \Rightarrow Algoritmo de particion binaria El siguiente algoritmo es una forma de resolver el problema planteado anteriormente. Consiste en ir generando el arbol de manera secuencial, iteracion a iteracion, minimizando en cada paso el error de clasificacion para las observaciones de train que caen en las ramas asociadas a la iteracion en cuestion que esta siendo optimizada.

El algoritmo se basa en la resolucion secuencial de problemas de minimizacion, uno por cada iteracion tenga el arbol que se acabará generando.

Importante:

Para esta exposicion teorica del algoritmo se va a usar como metrica principal la TEC , pero es facilmente extrapolable al caso en el que se usase el índice de Gini como metrica a optimizar.

4.5.2.1.1 Problema de la Iteración 1 Arbol con 1 iteración:

```
from IPython.display import Image
Image(filename='arbol_iter.jpg', width = 400, height = 200)
```

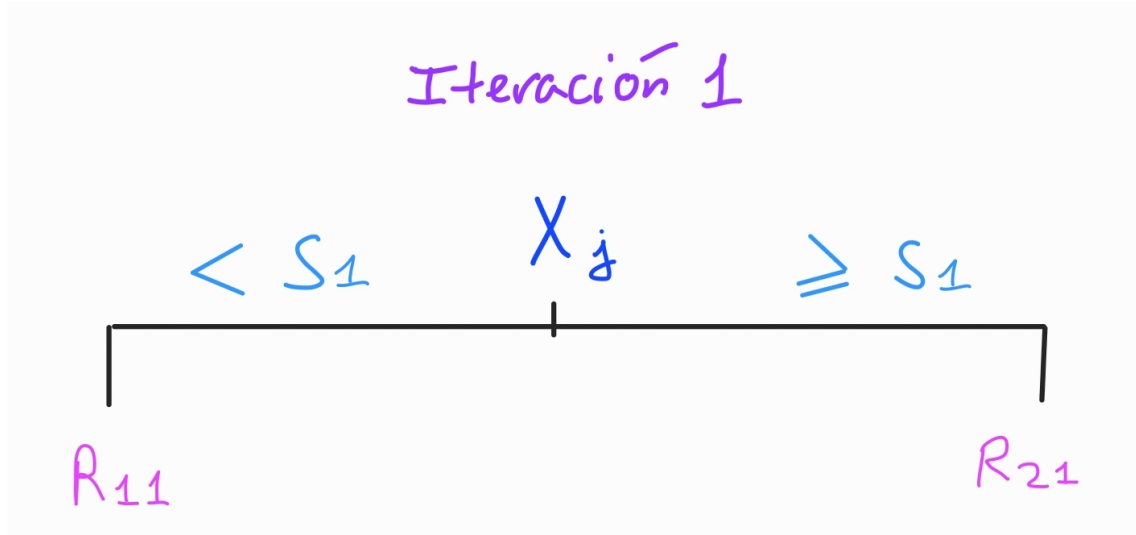


Figure 15: jpeg

La idea es, determinar las regiones R_{11} y R_{21} (es decir, j y s_1) del árbol con 1 iteración tal que minimizan el error de entrenamiento global de dicho árbol con 1 iteración.

hola

$$\begin{aligned}
& \underset{R_{11}, R_{21}}{Min} \ (G_1 = G_{R_{11}} + G_{R_{21}}) = \\
& = \underset{R_{11}, R_{21}}{Min} \left\{ \sum_{r=0,1,\dots,c-1} f_{r,R_{11}} \cdot (1 - f_{r,R_{11}}) + \sum_{r=0,1,\dots,c-1} f_{r,R_{21}} \cdot (1 - f_{r,R_{21}}) \right\} = \\
& = \underset{R_{11}, R_{21}}{Min} \left\{ \sum_{r=0,1,\dots,c-1} \frac{\# \{i / x_i \in R_{11} \text{ y } y_i = r\}}{\# \{i / x_i \in R_{11}\}} \cdot \left(1 - \frac{\# \{i / x_i \in R_{11} \text{ y } y_i = r\}}{\# \{i / x_i \in R_{11}\}}\right) + \right. \\
& \quad \left. \sum_{r=0,1,\dots,c-1} \frac{\# \{i / x_i \in R_{21} \text{ y } y_i = r\}}{\# \{i / x_i \in R_{21}\}} \cdot \left(1 - \frac{\# \{i / x_i \in R_{21} \text{ y } y_i = r\}}{\# \{i / x_i \in R_{21}\}}\right) \right\} = \\
& = \underset{j, s_1}{Min} \left\{ \sum_{r=0,1,\dots,c-1} \frac{\# \{i / x_{ij} < s_1 \text{ y } y_i = r\}}{\# \{i / x_{ij} < s_1\}} \cdot \left(1 - \frac{\# \{i / x_{ij} < s_1 \text{ y } y_i = r\}}{\# \{i / x_{ij} < s_1\}}\right) + \right. \\
& \quad \left. \sum_{r=0,1,\dots,c-1} \frac{\# \{i / x_{ij} \geq s_1 \text{ y } y_i = r\}}{\# \{i / x_{ij} \geq s_1\}} \cdot \left(1 - \frac{\# \{i / x_{ij} \geq s_1 \text{ y } y_i = r\}}{\# \{i / x_{ij} \geq s_1\}}\right) \right\}
\end{aligned}$$