Anexo_L

November 13, 2020

Anexo L

1 Análisis y Minería de Datos clínicos

Elaborado por: Ricardo Niño de Rivera Barrón

Ingeniería Biónica

Trabajo Terminal II .

En esta libreta interactiva en python 3.8 se analizan los datos clínicos de cada paciente involucrada en el estudio y propuestas de minería de datos para solventar la problemática de falta de algunos datos con el propósito de poder utilizar un modelo

Esta libreta se desarrolló en la plaforma Google Colab con el objetiivo de mejorar la velocidad de implementación.

[2]:
```
# Importando bibliotecas necesarias
import os
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import preprocessing
import matplotlib.image as mpimg
import sklearn
```

Instalando biblioteca para leer archivos de Google Drive

[3]:
```
!pip install PyDrive
```

```
Requirement already satisfied: PyDrive in /usr/local/lib/python3.6/dist-packages
(1.3.1)
Requirement already satisfied: google-api-python-client>=1.2 in
/usr/local/lib/python3.6/dist-packages (from PyDrive) (1.7.12)
Requirement already satisfied: oauth2client>=4.0.0 in
/usr/local/lib/python3.6/dist-packages (from PyDrive) (4.1.3)
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-
```

```
packages (from PyDrive) (3.13)
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in
/usr/local/lib/python3.6/dist-packages (from google-api-python-
client>=1.2->PyDrive) (3.0.1)
Requirement already satisfied: httplib2<1dev,>=0.17.0 in
/usr/local/lib/python3.6/dist-packages (from google-api-python-
client>=1.2->PyDrive) (0.17.4)
Requirement already satisfied: google-auth-httplib2>=0.0.3 in
/usr/local/lib/python3.6/dist-packages (from google-api-python-
client>=1.2->PyDrive) (0.0.4)
Requirement already satisfied: google-auth>=1.4.1 in
/usr/local/lib/python3.6/dist-packages (from google-api-python-
client>=1.2->PyDrive) (1.17.2)
Requirement already satisfied: six<2dev,>=1.6.1 in
/usr/local/lib/python3.6/dist-packages (from google-api-python-
client>=1.2->PyDrive) (1.15.0)
Requirement already satisfied: pyasn1-modules>=0.0.5 in
/usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->PyDrive)
(0.2.8)
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-
packages (from oauth2client>=4.0.0->PyDrive) (4.6)
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.6/dist-
packages (from oauth2client>=4.0.0->PyDrive) (0.4.8)
Requirement already satisfied: setuptools>=40.3.0 in
/usr/local/lib/python3.6/dist-packages (from google-auth>=1.4.1->google-api-
python-client>=1.2->PyDrive) (50.3.2)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/usr/local/lib/python3.6/dist-packages (from google-auth>=1.4.1->google-api-
python-client>=1.2->PyDrive) (4.1.1)
```

[4]:
```
# Importamos los métodos necesarios de la biblioteca instalada
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

[5]:
```
# Autenticamos el acceso a la cuenta Google Drive
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

Ahora instalamos una nueva biblioteca para poder descargar archivos de MEGA.

[6]:
```
# Instalanado mega.py
!pip install mega.py
```

```
Collecting mega.py
  Downloading https://files.pythonhosted.org/packages/a3/51/44a1085a091c27ade09e
```

```
122d5abdafb4b6400265081879a7c4e32973a175/mega.py-1.0.8-py2.py3-none-any.whl
Requirement already satisfied: pathlib==1.0.1 in /usr/local/lib/python3.6/dist-
packages (from mega.py) (1.0.1)
Requirement already satisfied: requests>=0.10 in /usr/local/lib/python3.6/dist-
packages (from mega.py) (2.23.0)
Collecting tenacity<6.0.0,>=5.1.5
  Downloading https://files.pythonhosted.org/packages/45/67/67bb1db087678bc5c6f2
0766cf18914dfe37b0b9d4e4c5bb87408460b75f/tenacity-5.1.5-py2.py3-none-any.whl
Collecting pycryptodome<4.0.0,>=3.9.6
  Downloading https://files.pythonhosted.org/packages/2b/6f/7e38d7c97fbbc3
987539c804282c33f56b6b07381bf2390deead696440c5/pycryptodome-3.9.9-cp36-cp36m-man
ylinux1_x86_64.whl (13.7MB)
     |%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%| 13.7MB 288kB/s
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.6/dist-packages (from requests>=0.10->mega.py) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.6/dist-packages (from requests>=0.10->mega.py)
(2020.6.20)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests>=0.10->mega.py) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.6/dist-packages (from requests>=0.10->mega.py) (3.0.4)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/dist-
packages (from tenacity<6.0.0,>=5.1.5->mega.py) (1.15.0)
Installing collected packages: tenacity, pycryptodome, mega.py
Successfully installed mega.py-1.0.8 pycryptodome-3.9.9 tenacity-5.1.5
```

[7]:
```
# De la biblioteca mega importamos el método Mega
from mega import Mega
```

[8]:
```
# Instanciando Mega en el objeto mega
mega = Mega()
```

Ahora necesitamos obtener un nuevo archivo de Google Drive con la información de acceso a la cuenta de MEGA donde tenemos los archivos de interés.

[9]:
```
# Accedemos al archivo con los datos de acceso a la cuenta en MEGA
downloaded = drive.CreateFile({'id':"1qY56J5ziiRklMPOIfS16vJrLsjZ7Vqso"})   #␣
 ↪instanciamos con el ID del documento al que se desea acceder en Google Drive
downloaded.GetContentFile('cuentaMEGA.csv')        # descargamos el archivo en␣
 ↪el entorno virtual de Google Colab
```

[10]:
```
# Leyendo el archivo con los datos de acceso
cuenta=pd.read_csv('cuentaMEGA.csv')
```

Inicamos sesión en la cuenta de MEGA

[11]:
```
email = cuenta.EMAIL.iloc[0]
password = cuenta.PASSWORD.iloc[0]

# Log in en la cuenta de MEGA con los datos de acceso
m = mega.login(email, password)
```

Descargamos los archivos con los datos clínicos.

[12]:
```
# Descargando Y_train.npy
file = m.find('/TT2_SegundaParte/train.csv')
m.download(file)
```

[12]:
```
PosixPath('train.csv')
```

[13]:
```
# Descargando Y_test.npy
file = m.find('/TT2_SegundaParte/test.csv')
m.download(file)
```

[13]:
```
PosixPath('test.csv')
```

Ahora se leen estos archivos con pandas

[14]:
```
train=pd.read_csv('train.csv')
train.head()
```

[14]:
```
   ID  ... Put some pomade, deodorant or products at breasts or armpits region?
0   1  ...                                                  NaN
1   2  ...                                                  0.0
2   4  ...                                                  NaN
3   5  ...                                                  NaN
4   6  ...                                                  NaN

[5 rows x 23 columns]
```

[15]:
```
test=pd.read_csv('test.csv')
test.head()
```

[15]:
```
   ID  ... Put some pomade, deodorant or products at breasts or armpits region?
0  12  ...                                                  0.0
1  19  ...                                                  0.0
2  20  ...                                                  0.0
3  22  ...                                                  0.0
4  32  ...                                                  0.0

[5 rows x 23 columns]
```

Realizamos una exploración de las columnas en train

[16]:
```
train.columns
```

[16]:
```
Index(['ID', 'Records', 'Name', 'Age', 'Exams', 'Diagnosis', 'Side',
       'Complaints', 'Mammography', 'Radiotherapy', 'Plastic surgery',
       'Prosthesis', 'Biopsy', 'Use of hormone replacement',
       'Is there signal of wart on breast?', 'Wart', 'Body temperature',
       'Race', 'Smoked', 'Drank coffe', 'Comsumed alcohol',
       'Physical exercise',
       'Put some pomade, deodorant or products at breasts or armpits region?'],
      dtype='object')
```

[17]:
```
len(train.columns)
```

[17]:
```
23
```

[18]:
```
len(test.columns)
```

[18]:
```
23
```

Como se observa existen 23 campos en nuestra tabla, sin embargo los campos ID, Records, Name, Side, Race y Exams no otorgan datos sobre el historial clínico o el cumplimiento de las condiciones previas al examen termográfico. Por tanto, los campos que otorgan información relevante aparentemente son 17, los cuales podrían disminuir aún más con un análisis detallado.

2 Analizando Datos Clínicos del conjunto de entrenamiento

El análisis que se propone en esta sección de la liberta interactiva no sólo es exploratorio, también determinará la utilización o no de los campos en las tablas para la construcción de un modelo reconocedor complementario a la image termográfica.

[19]:
```
# Examinando si alguna paciente no tiene ID o es NaN
train.ID.isnull().sum()
```

[19]:
```
0
```

Como podemos ver todas las pacientes en la tabla de entrenamiento tienen un ID.

Ahora se cuentan todos los valores en ID (todas las pacientes).

[20]:
```
# Contando a todas las pacientes del conjunto de entrenamiento
total_train=train.ID.count()
print(total_train)
```

```
169
```

En total tenemos 169 pacientes, con esta información procederemos a los análisis posteriores por campo en la tabla train.

2.0.1 Age

Se analiza la variable Age, de la tabla de pacientes.

Primero buscamos si falta algún valor o es null.

[21]:
```
train.ID.isnull().sum()
```

[21]:
```
0
```

[22]:
```
train.Age.count()
```

[22]:
```
169
```

Tenemos 100% de los datos completos en Age

2.0.2 Diagnosis

Este campo es la salida de interés del modelo que se realizará.

[23]:
```
# Calculando elementos faltantes
train.Diagnosis.isnull().sum()
```

[23]:
```
0
```

[24]:
```
# Elementos por clase
train.Diagnosis.value_counts()
```

[24]:
```
Healthy    140
Sick        29
Name: Diagnosis, dtype: int64
```

[25]:
```
train.Diagnosis.loc[train['Diagnosis']=='Healthy'].count()
```

[25]:
```
140
```

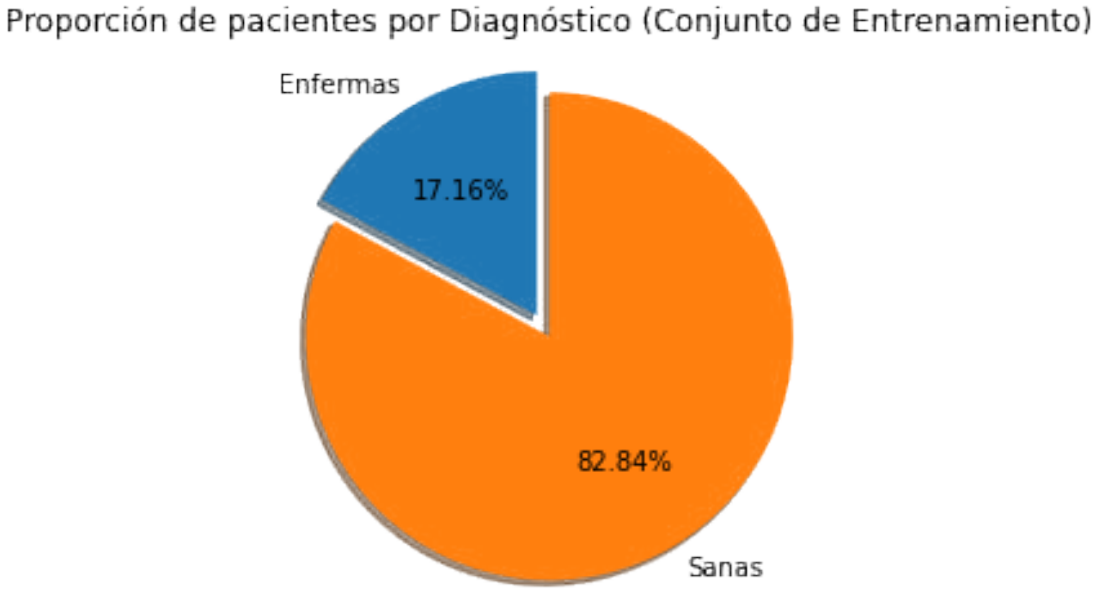
[26]:
```
tasa_positivas = (train.Diagnosis.loc[train['Diagnosis']=='Sick'].count()/
 ↪total_train)*100
tasa_negativas = (train.Diagnosis.loc[train['Diagnosis']=='Healthy'].count()/
 ↪total_train)*100
```

[27]:
```
# Sanas
print("Tasa de sanas %0.2f%%"%tasa_negativas)

# Enfermas
print("Tasa de enfermas %0.2f%%"%tasa_positivas)
```

```
Tasa de sanas 82.84%
Tasa de enfermas 17.16%
```

[28]:
```
# Ahora mostramos una gráfica de "pastel" de esto información
labels = 'Enfermas', 'Sanas'
sizes = [tasa_positivas, tasa_negativas]
explode = (0.1, 0)  # Resaltando enfermas o clase positiva
```

```
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de pacientes por Diagnóstico (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.3 Complaints

Esta variable se refiere a si la paciente sentía algún dolor en las regiones de interés (mamas).

Las etiquetas son:

- 1: Dolor
- 0: Sin dolor

[29]:
```
# Elementos NaN o null
train.Complaints.isnull().sum()
```

[29]: 27

[30]:
```
# Elementos con dolor o sin dolor
train.Complaints.value_counts()
```

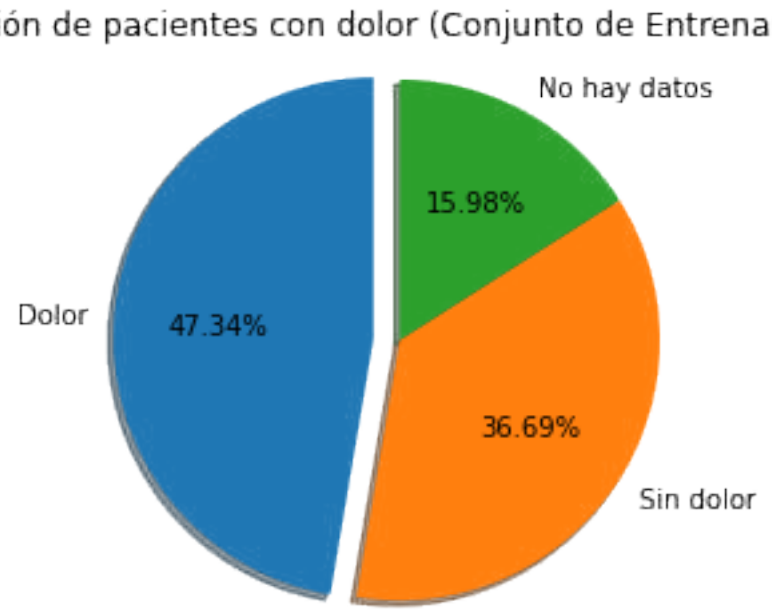
[30]:
```
1.0    80
0.0    62
Name: Complaints, dtype: int64
```

[31]:
```
tasa_dolor = (train.Complaints.loc[train['Complaints']==1.0].count()/
 ↪total_train)*100
tasa_NoDolor = (train.Complaints.loc[train['Complaints']==0.0].count()/
 ↪total_train)*100
tasa_no_data_dolor = (train.Complaints.isnull().sum()/total_train)*100
```

[32]:
```
# Dolor
print("Tasa de pacientes con dolor %0.2f%%"%tasa_dolor)

# Sin dolor
print("Tasa de pacientes sin dolor %0.2f%%"%tasa_NoDolor)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_no_data_dolor)
```

```
Tasa de pacientes con dolor 47.34%
Tasa de pacientes sin dolor 36.69%
Tasa de pacientes sin datos 15.98%
```

[33]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Dolor', 'Sin dolor', 'No hay datos'
sizes = [tasa_dolor, tasa_NoDolor, tasa_no_data_dolor]
explode = (0.1, 0, 0)  # Resaltando el porcentaje con dolor

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de pacientes con dolor (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.4 Mammography

Esta variable se refiere a si la paciente se realizó una mastografía o no. Esta información no es relevante para el reconocedor propuesto pero se mostraran las gráficas que describen a este campo de la tabla con el objetivo de describir de forma gráfica la información del paciente.

[34]:
```
# Elementos NaN o null
train.Mammography.isnull().sum()
```

[34]:
```
0
```

[35]:
```
# Elementos con mastografía y sin mastografía
train.Mammography.value_counts()
```

[35]:
```
1    146
0     23
Name: Mammography, dtype: int64
```

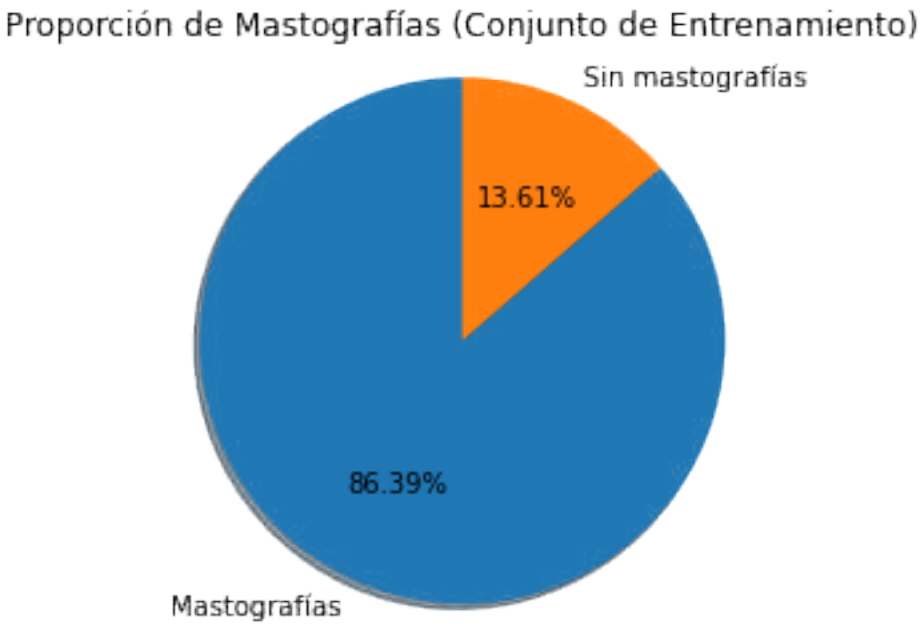
[36]:
```
tasa_masto = (train.Mammography.loc[train['Mammography']==1.0].count()/
 ↪total_train)*100
tasa_NoMasto = (train.Mammography.loc[train['Mammography']==0.0].count()/
 ↪total_train)*100
```

[37]:
```
# Con Mastografía
print("Tasa de pacientes con mastografía %0.2f%%"%tasa_masto)
```

```
# Sin Mastografía
print("Tasa de pacientes sin mastografía %0.2f%%"%tasa_NoMasto)
```

```
Tasa de pacientes con mastografía 86.39%
Tasa de pacientes sin mastografía 13.61%
```

[38]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Mastografías', 'Sin mastografías'
sizes = [tasa_masto, tasa_NoMasto]
explode = (0, 0)

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Mastografías (Conjunto de Entrenamiento)')

plt.show()
```



¿Por qué no todas las pacientes tienen registro de mastografía?

La normas para la detección no recomienda que las mujeres tengan que realizarse un mastografías cada año antes de los 40 años y después de los 60 años (incluida la norma mexicana).

1. Elemento de la lista
2. Elemento de la lista

Este estudio valida los positivos y negativos con base a los resultados de biopsia y un seguimiento de un año para descartar cáncer.

Con la biopsia se obtuvieron se confirman los casos positivos, mientras que con el seguimiento por un año, se confirman los casos negativos.

2.0.5 Radiotherapy

[39]:
```
# Elementos NaN o null
train.Radiotherapy.isnull().sum()
```

[39]:
```
3
```

[40]:
```
# Elementos con mastografía y sin mastografía
train.Radiotherapy.value_counts()
```

[40]:
```
0.0    147
1.0     19
Name: Radiotherapy, dtype: int64
```

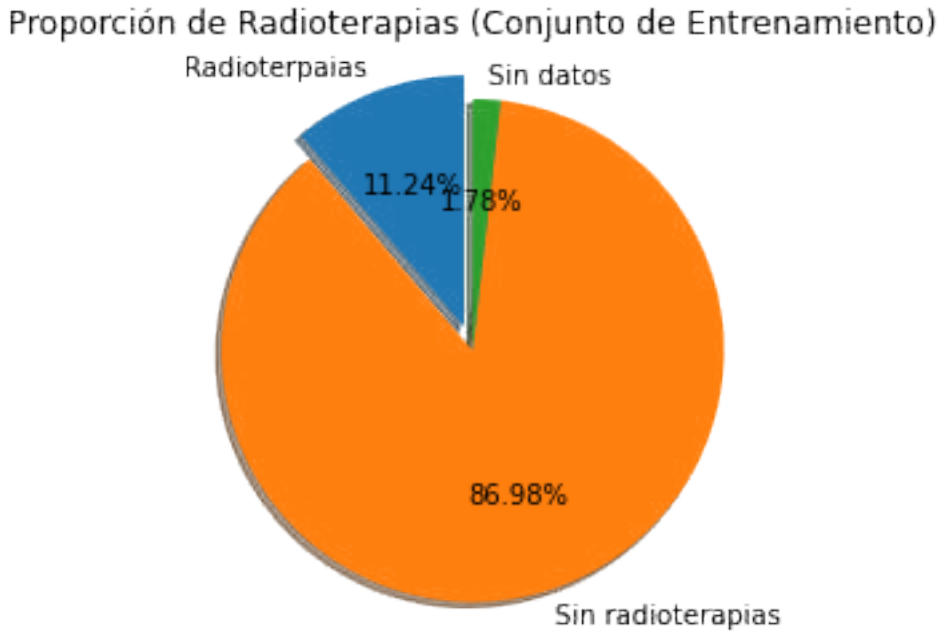
[41]:
```
tasa_radio = (train.Radiotherapy.loc[train['Radiotherapy']==1.0].count()/
 ↪total_train)*100
tasa_NoRadio = (train.Radiotherapy.loc[train['Radiotherapy']==0.0].count()/
 ↪total_train)*100
tasa_No_data_radio = (train.Radiotherapy.isnull().sum()/total_train)*100
```

[42]:
```
# Con Radioterapia
print("Tasa de pacientes con radioterpia %0.2f%%"%tasa_radio)

# Sin Radioterapia
print("Tasa de pacientes sin radioterapia %0.2f%%"%tasa_NoRadio)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_radio)
```

```
Tasa de pacientes con radioterpia 11.24%
Tasa de pacientes sin radioterapia 86.98%
Tasa de pacientes sin datos 1.78%
```

[43]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Radioterpaias', 'Sin radioterapias', 'Sin datos'
sizes = [tasa_radio, tasa_NoRadio, tasa_No_data_radio]
explode = (0.1, 0, 0)  # Resaltando el porcentaje con radioterapia

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo
```

```
ax1.title.set_text('Proporción de Radioterapias (Conjunto de Entrenamiento)')

plt.show()
```



2.0.6 Plastic surgery

[44]:
```
# Elementos NaN o null
train["Plastic surgery"].isnull().sum()
```

[44]:
```
2
```

[45]:
```
# Elementos con y sin cirugía plástica
train["Plastic surgery"].value_counts()
```

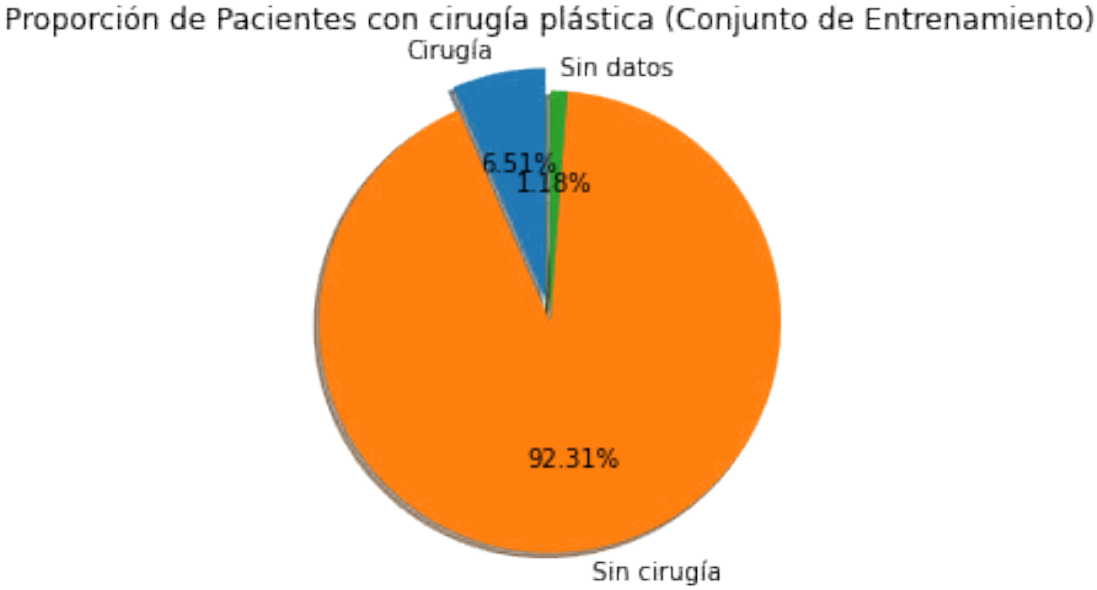
[45]:
```
0.0    156
1.0     11
Name: Plastic surgery, dtype: int64
```

[46]:
```
tasa_plastic = (train["Plastic surgery"].loc[train['Plastic surgery']==1.0].
 ↪count()/total_train)*100
tasa_NoPlastic = (train["Plastic surgery"].loc[train['Plastic surgery']==0.0].
 ↪count()/total_train)*100
tasa_No_data_plastic = (train["Plastic surgery"].isnull().sum()/total_train)*100
```

[47]:
```
# Con cirugía plástica
print("Tasa de pacientes con cirugía plástica %0.2f%%"%tasa_plastic)

# Sin cirugía plástica
print("Tasa de pacientes sin cirugía plástica %0.2f%%"%tasa_NoPlastic)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_plastic)
```

```
Tasa de pacientes con cirugía plástica 6.51%
Tasa de pacientes sin cirugía plástica 92.31%
Tasa de pacientes sin datos 1.18%
```

[48]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Cirugía', 'Sin cirugía', 'Sin datos'
sizes = [tasa_plastic, tasa_NoPlastic, tasa_No_data_plastic]
explode = (0.1, 0, 0)  # Resaltando el porcentaje con cirugía plástica

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes con cirugía plástica (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.7 Prosthesis

Esta variable se refiere a presencia o no de prótesis mamaria.

[49]:
```
# Elementos NaN o null
train.Prosthesis.isnull().sum()
```

[49]:
```
2
```

[50]:
```
# Elementos con y sin prótesis
train.Prosthesis.value_counts()
```

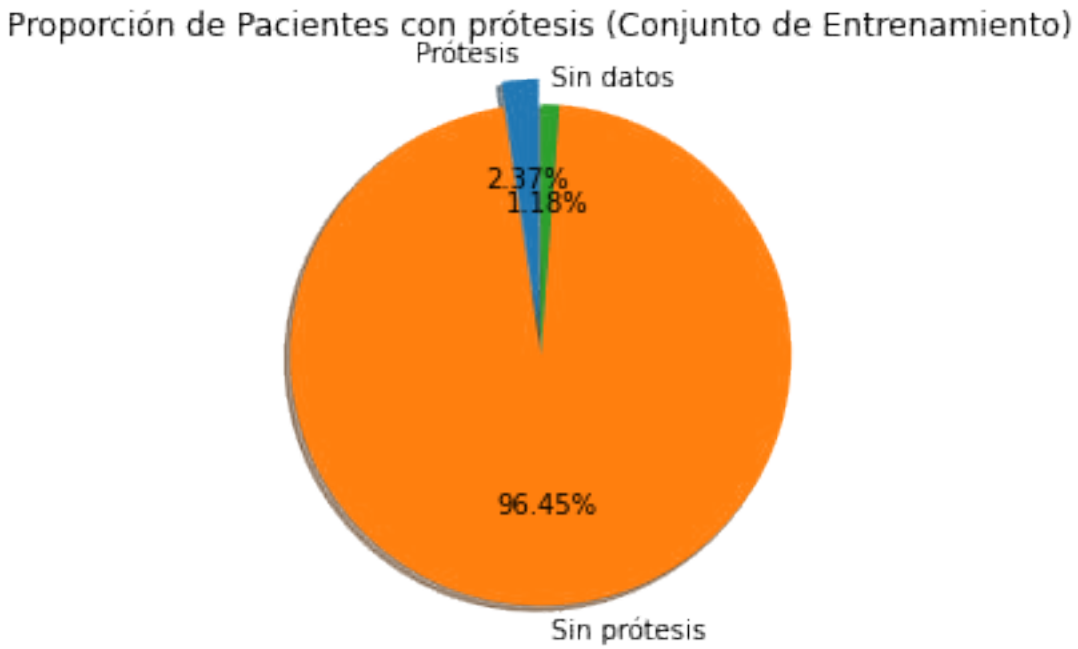
[50]:
```
0.0    163
1.0      4
Name: Prosthesis, dtype: int64
```

[51]:
```
tasa_prost = (train["Prosthesis"].loc[train['Prosthesis']==1.0].count()/
 ↪total_train)*100
tasa_NoProst = (train["Prosthesis"].loc[train['Prosthesis']==0.0].count()/
 ↪total_train)*100
tasa_No_data_prost = (train["Prosthesis"].isnull().sum()/total_train)*100
```

[52]:
```
# Con prótesis
print("Tasa de pacientes con protésis %0.2f%%"%tasa_prost)

# Sin prótesis
print("Tasa de pacientes sin protésis %0.2f%%"%tasa_NoProst)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_prost)
```

```
Tasa de pacientes con protésis 2.37%
Tasa de pacientes sin protésis 96.45%
Tasa de pacientes sin datos 1.18%
```

[53]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Prótesis', 'Sin prótesis', 'Sin datos'
sizes = [tasa_prost, tasa_NoProst, tasa_No_data_prost]
explode = (0.1, 0, 0)  # Resaltando el porcentaje con prótesis

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes con prótesis (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.8 Biopsy

Esta variable no se utilizará para el reconocimiento, es un indicador de validación de los ejemplos positivos en el estudio.

[54]:
```
# Elementos NaN o null
train.Biopsy.isnull().sum()
```

[54]:
```
6
```

[55]:
```
# Elementos con y sin biopsia
train.Biopsy.value_counts()
```

[55]:
```
0.0    119
1.0     44
Name: Biopsy, dtype: int64
```

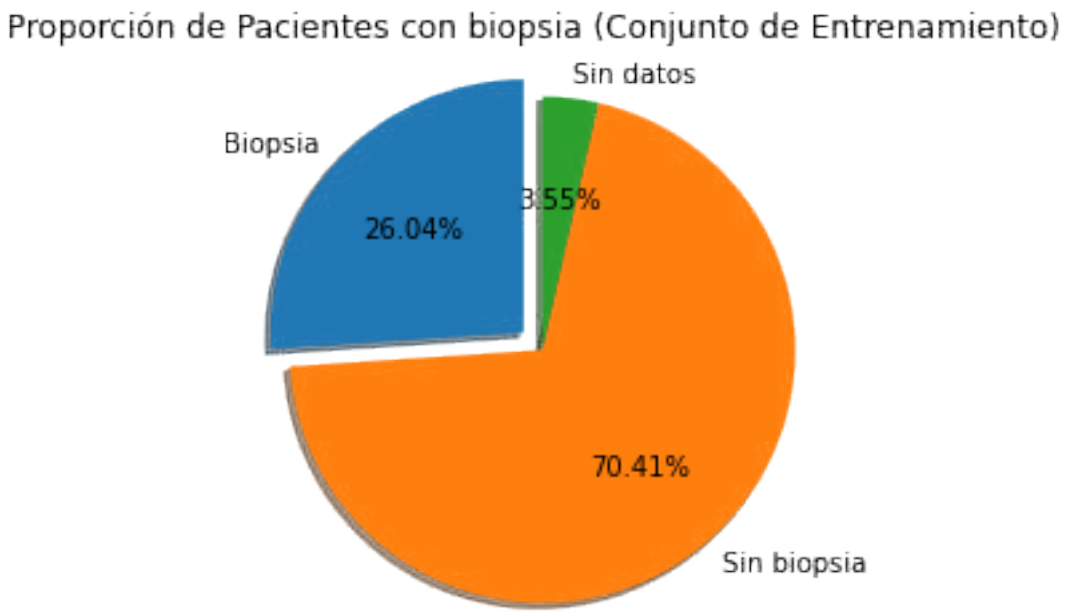
[56]:
```
tasa_bio = (train.Biopsy.loc[train['Biopsy']==1.0].count()/total_train)*100
tasa_NoBio = (train.Biopsy.loc[train['Biopsy']==0.0].count()/total_train)*100
tasa_No_data_bio = (train.Biopsy.isnull().sum()/total_train)*100
```

[57]:
```
# Con biopsia
print("Tasa de pacientes con biopsia %0.2f%%"%tasa_bio)

# Sin biopsia
print("Tasa de pacientes sin biopsia %0.2f%%"%tasa_NoBio)
```

```
# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_bio)
```

```
Tasa de pacientes con biopsia 26.04%
Tasa de pacientes sin biopsia 70.41%
Tasa de pacientes sin datos 3.55%
```

[58]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Biopsia', 'Sin biopsia', 'Sin datos'
sizes = [tasa_bio, tasa_NoBio, tasa_No_data_bio]
explode = (0.1, 0, 0)  # Resaltando el porcentaje con biopsia

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes con biopsia (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.9 Use of hormone replacement

Se refiere a si la paciente emplea algún tratamiento hormonal en general.

[59]:
```
# Elementos NaN o null
train['Use of hormone replacement'].isnull().sum()
```

[59]:
```
3
```

[60]:
```
# Elementos con y sin biopsia
train['Use of hormone replacement'].value_counts()
```

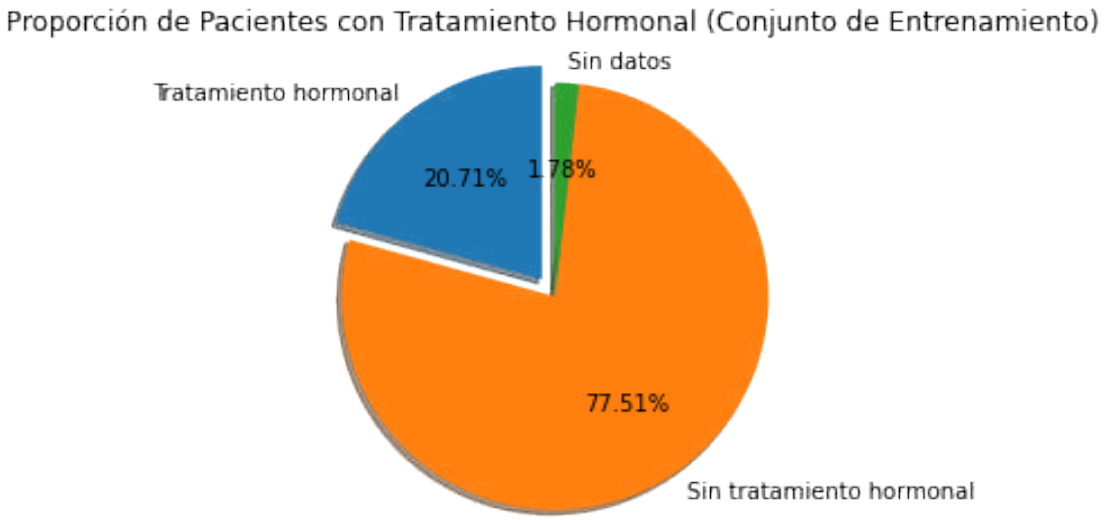
[60]:
```
0.0    131
1.0     35
Name: Use of hormone replacement, dtype: int64
```

[61]:
```
tasa_horm = (train['Use of hormone replacement'].loc[train['Use of hormone␣
 ↪replacement']==1.0].count()/total_train)*100
tasa_NoHorm = (train['Use of hormone replacement'].loc[train['Use of hormone␣
 ↪replacement']==0.0].count()/total_train)*100
tasa_No_data_horm = (train['Use of hormone replacement'].isnull().sum()/
 ↪total_train)*100
```

[62]:
```
# Con tratamiento hormonal
print("Tasa de pacientes con tratamiento hormonal %0.2f%%"%tasa_horm)

# Sin tratamiento hormonal
print("Tasa de pacientes sin biopsia %0.2f%%"%tasa_NoHorm)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_horm)
```

```
Tasa de pacientes con tratamiento hormonal 20.71%
Tasa de pacientes sin biopsia 77.51%
Tasa de pacientes sin datos 1.78%
```

[63]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Tratamiento hormonal', 'Sin tratamiento hormonal', 'Sin datos'
sizes = [tasa_horm, tasa_NoHorm, tasa_No_data_horm]
explode = (0.1, 0, 0)  # Resaltando el porcentaje con biopsia

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes con Tratamiento Hormonal (Conjunto␣
 ↪de Entrenamiento)')

plt.show()
```



2.0.10 Campos *Is there signal of wart on breast?* y *wart*

Estas variables se refieren a la presencia de alguna verruga, la primera no es específica de la ubicación, la segunda sí. Utilizaremos la primera.

[64]:
```
# Elementos NaN o null
train['Is there signal of wart on breast?'].isnull().sum()
```

[64]:
```
2
```

[65]:
```
# Elementos con y sin verrugas
train['Is there signal of wart on breast?'].value_counts()
```

[65]:
```
0.0    137
1.0     30
Name: Is there signal of wart on breast?, dtype: int64
```

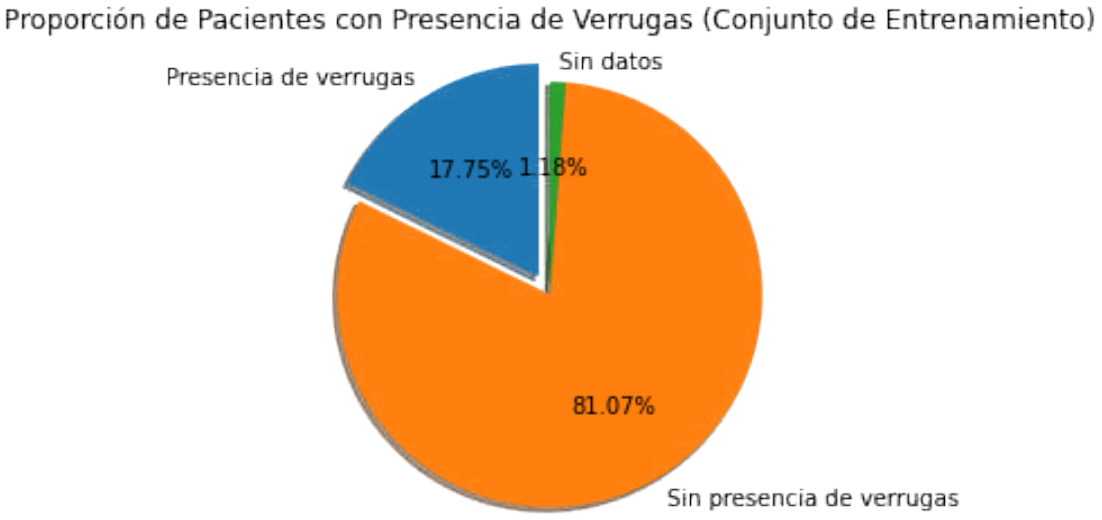
[66]:
```
tasa_wart = (train['Is there signal of wart on breast?'].loc[train['Is there␣
 ↪signal of wart on breast?']==1.0].count()/total_train)*100
tasa_NoWart = (train['Is there signal of wart on breast?'].loc[train['Is there␣
 ↪signal of wart on breast?']==0.0].count()/total_train)*100
tasa_No_data_wart = (train['Is there signal of wart on breast?'].isnull().sum()/
 ↪total_train)*100
```

[67]:
```
# Con presencia de verrugas
print("Tasa de pacientes con presencia de verrugas %0.2f%%"%tasa_wart)

# Sin presencia de verrugas
print("Tasa de pacientes sin presencia de verrugas %0.2f%%"%tasa_NoWart)
```

```
# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_wart)
```

```
Tasa de pacientes con presencia de verrugas 17.75%
Tasa de pacientes sin presencia de verrugas 81.07%
Tasa de pacientes sin datos 1.18%
```

[68]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Presencia de verrugas', 'Sin presencia de verrugas', 'Sin datos'
sizes = [tasa_wart, tasa_NoWart, tasa_No_data_wart]
explode = (0.1, 0, 0)  # Resaltando el porcentaje de pacientes con presencia de␣
 ↪verrugas

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes con Presencia de Verrugas (Conjunto␣
 ↪de Entrenamiento)')

plt.show()
```



2.0.11 Body temperature

Esta variable se refiere a la temperatura corporal obtenida con auxilio de un termómetro clínico.

[69]:
```
# Elementos NaN o null
train['Body temperature'].isnull().sum()
```

[69]:
```
5
```

[70]:
```
# Promedio de la temperatura
# Esta temperatura
train['Body temperature'].mean()
```

[70]:
```
35.46402439024391
```

[71]:
```
# Porcentaje de muejeres con valor de temperatura corporal
tasa_temp = ((total_train-train['Body temperature'].isnull().sum())/
 ↪total_train)*100
tasa_No_data_temp = (train['Body temperature'].isnull().sum()/total_train)*100
print("Porcentaje de pacientes con datos de temperatura corporal %0.
 ↪2f%%"%tasa_temp)
print("Porcentaje de pacientes sin datos de temperatura corporal %0.
 ↪2f%%"%tasa_No_data_temp)
```

```
Porcentaje de pacientes con datos de temperatura corporal 97.04%
Porcentaje de pacientes sin datos de temperatura corporal 2.96%
```

2.0.12 Smoked

A partir de esta variable se estará haiendo referencia a actividades que la paciente realizó 2 horas previas del estudio.

[72]:
```
# Elementos NaN o null
train.Smoked.isnull().sum()
```

[72]:
```
20
```

[73]:
```
# Elementos que furmaon y no fumaron
train.Smoked.value_counts()
```

[73]:
```
0.0    140
1.0      9
Name: Smoked, dtype: int64
```

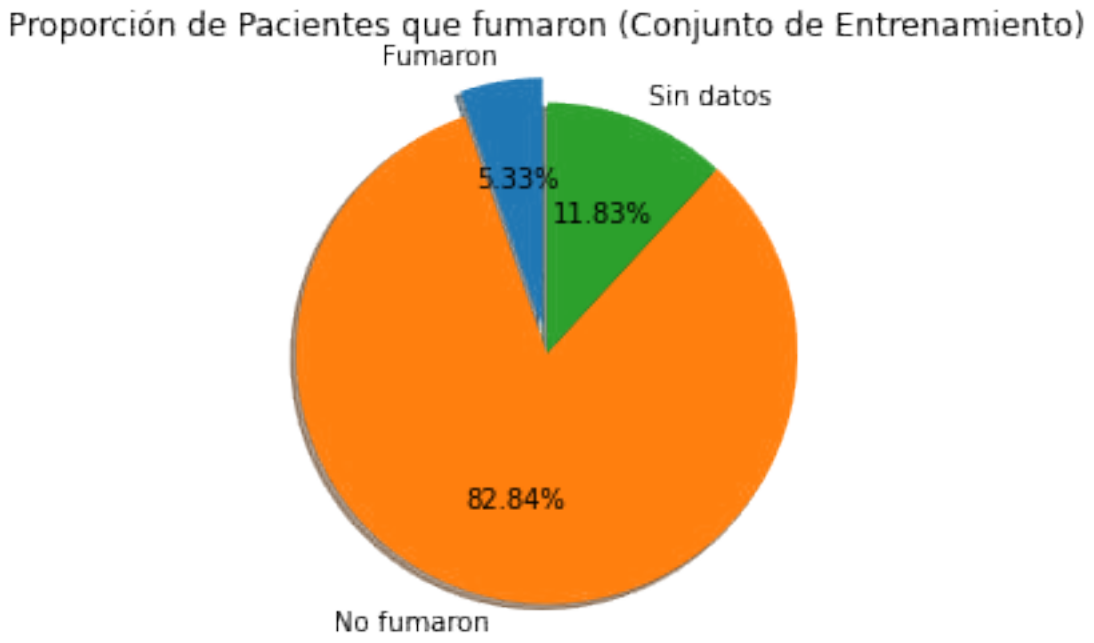
[74]:
```
tasa_smok = (train.Smoked.loc[train.Smoked==1.0].count()/total_train)*100
tasa_NoSmok = (train.Smoked.loc[train.Smoked==0.0].count()/total_train)*100
tasa_No_data_smok = (train.Smoked.isnull().sum()/total_train)*100
```

[75]:
```
# Fumaron
print("Tasa de pacientes que fumaron %0.2f%%"%tasa_smok)

# No fumaron
```

```
print("Tasa de pacientes que no fumaron %0.2f%%"%tasa_NoSmok)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_smok)
```

```
Tasa de pacientes que fumaron 5.33%
Tasa de pacientes que no fumaron 82.84%
Tasa de pacientes sin datos 11.83%
```

[76]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Fumaron', 'No fumaron', 'Sin datos'
sizes = [tasa_smok, tasa_NoSmok, tasa_No_data_smok]
explode = (0.1, 0, 0)  # Resaltando el porcentaje de pacientes que fumaron

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes que fumaron (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.13 Drank coffe

[77]:
```
# Elementos NaN o null
train['Drank coffe'].isnull().sum()
```

[77]:
```
20
```

[78]:
```
# Elementos que tomaron y no tomaron café
train['Drank coffe'].value_counts()
```

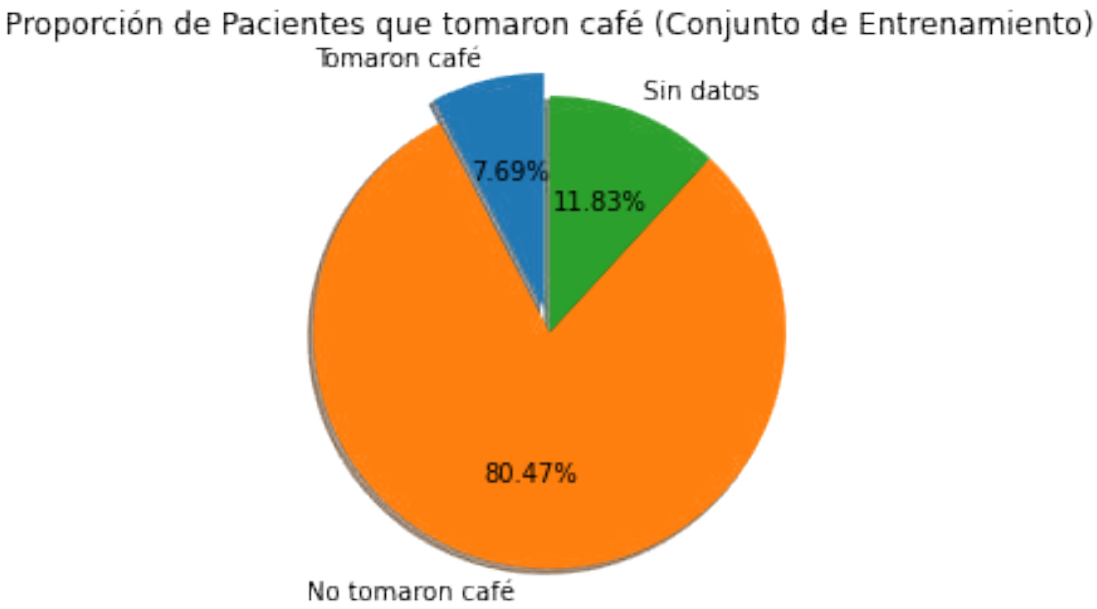
[78]:
```
0.0    136
1.0     13
Name: Drank coffe, dtype: int64
```

[79]:
```
tasa_coffee = (train['Drank coffe'].loc[train['Drank coffe']==1.0].count()/
 ↪total_train)*100
tasa_NoCoffee = (train['Drank coffe'].loc[train['Drank coffe']==0.0].count()/
 ↪total_train)*100
tasa_No_data_coffee = (train['Drank coffe'].isnull().sum()/total_train)*100
```

[80]:
```
# Tomaron café
print("Tasa de pacientes que tomaron café %0.2f%%"%tasa_coffee)

# No tomaron café
print("Tasa de pacientes que no tomaron café %0.2f%%"%tasa_NoCoffee)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_coffee)
```

```
Tasa de pacientes que tomaron café 7.69%
Tasa de pacientes que no tomaron café 80.47%
Tasa de pacientes sin datos 11.83%
```

[81]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Tomaron café', 'No tomaron café', 'Sin datos'
sizes = [tasa_coffee, tasa_NoCoffee, tasa_No_data_coffee]
explode = (0.1, 0, 0)  # Resaltando el porcentaje de pacientes que tomaron café

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes que tomaron café (Conjunto de␣
 ↪Entrenamiento)')

plt.show()
```



2.0.14 Comsumed alcohol

[82]:
```
# Elementos NaN o null
train['Comsumed alcohol'].isnull().sum()
```

[82]:
```
20
```

[83]:
```
# Elementos que tomaron y no tomaron café
train['Comsumed alcohol'].value_counts()
```

[83]:
```
0.0    149
Name: Comsumed alcohol, dtype: int64
```

[84]:
```
tasa_alc = (train['Comsumed alcohol'].loc[train['Comsumed alcohol']==1.0].
 ↪count()/total_train)*100
tasa_NoAlc = (train['Comsumed alcohol'].loc[train['Comsumed alcohol']==0.0].
 ↪count()/total_train)*100
tasa_No_data_alc = (train['Comsumed alcohol'].isnull().sum()/total_train)*100
```

[85]:
```
# Consumieron alcohol
print("Tasa de pacientes que consumieron alcohol %0.2f%%"%tasa_alc)

# No consumieron alcohol
print("Tasa de pacientes que no consumieron alcohol %0.2f%%"%tasa_NoAlc)

# Sin datos
```

```
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_alc)
```

```
Tasa de pacientes que consumieron alcohol 0.00%
Tasa de pacientes que no consumieron alcohol 88.17%
Tasa de pacientes sin datos 11.83%
```

[86]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Consumieron acohol', 'No consumieron alcohol', 'Sin datos'
sizes = [tasa_alc, tasa_NoAlc, tasa_No_data_alc]
explode = (0.1, 0, 0)  # Resaltando el porcentaje de pacientes que consumieron␣
↪alcohol

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes que consumieron alcohol (Conjunto de␣
↪Entrenamiento)')

plt.show()
```

2.0.15 Physical exercise

[87]:
```
# Elementos NaN o null
train['Physical exercise'].isnull().sum()
```

[87]: 20

[88]:
```
# Elementos que realizó ejercicio
train['Physical exercise'].value_counts()
```

[88]:
```
0.0    149
Name: Physical exercise, dtype: int64
```

[89]:
```
tasa_ex = (train['Physical exercise'].loc[train['Physical exercise']==1.0].
 ↪count()/total_train)*100
tasa_NoEx = (train['Physical exercise'].loc[train['Physical exercise']==0.0].
 ↪count()/total_train)*100
tasa_No_data_ex = (train['Physical exercise'].isnull().sum()/total_train)*100
```

[90]:
```
# Realizaron ejercicio
print("Tasa de pacientes que realizaron ejercicio %0.2f%%"%tasa_ex)

# No realizaron ejercicio
print("Tasa de pacientes que no realizaron ejercicio %0.2f%%"%tasa_NoEx)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_ex)
```

```
Tasa de pacientes que realizaron ejercicio 0.00%
Tasa de pacientes que no realizaron ejercicio 88.17%
Tasa de pacientes sin datos 11.83%
```

[91]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Realizaron ejercicio', 'No realizaron ejercicio', 'Sin datos'
sizes = [tasa_ex, tasa_NoEx, tasa_No_data_ex]
explode = (0.1, 0, 0)  # Resaltando el porcentaje de pacientes que realizaron␣
 ↪ejercicio

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('Proporción de Pacientes que realizaron ejercicio (Conjunto␣
 ↪de Entrenamiento)')

plt.show()
```

2.0.16 Put some pomade, deodorant or products at breasts or armpits region?

[92]:
```
# Elementos NaN o null
train['Put some pomade, deodorant or products at breasts or armpits region?'].
 ↪isnull().sum()
```

[92]:
```
20
```

[93]:
```
# Elementos que colocaron alguna pomada, desodarante u otro producto
train['Put some pomade, deodorant or products at breasts or armpits region?'].
 ↪value_counts()
```

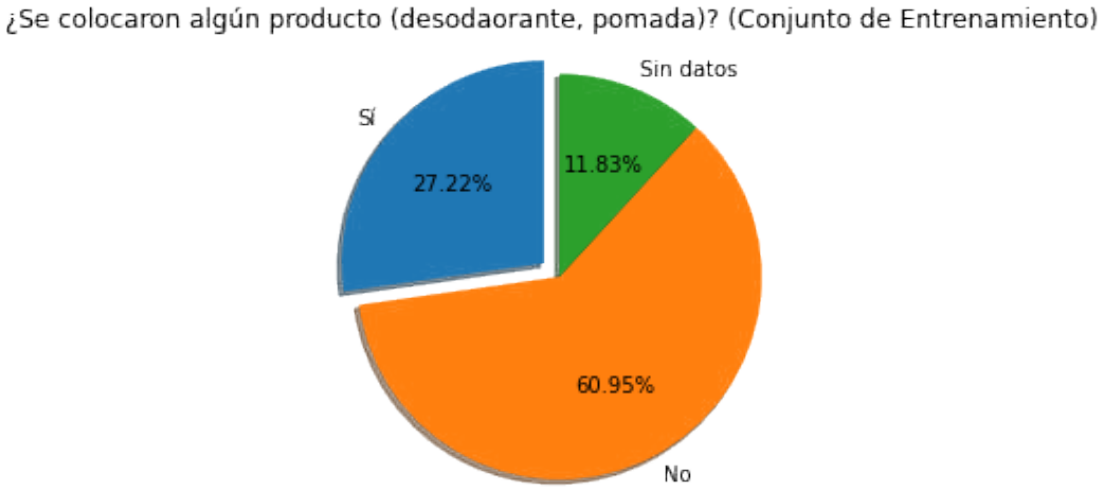
[93]:
```
0.0    103
1.0     46
Name: Put some pomade, deodorant or products at breasts or armpits region?,
dtype: int64
```

[94]:
```
tasa_deo = (train['Put some pomade, deodorant or products at breasts or armpits␣
 ↪region?'].loc[train['Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?']==1.0].count()/total_train)*100
tasa_NoDeo = (train['Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?'].loc[train['Put some pomade, deodorant or products at breasts␣
 ↪or armpits region?']==0.0].count()/total_train)*100
tasa_No_data_deo = (train['Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?'].isnull().sum()/total_train)*100
```

[95]:
```
# Se colocaron algún producto
print("Tasa de pacientes que se colocaron algún producto %0.2f%%"%tasa_deo)

# No se colocaron algún producto
print("Tasa de pacientes que no se colocaron algún producto %0.2f%%"%tasa_NoDeo)

# Sin datos
print("Tasa de pacientes sin datos %0.2f%%"%tasa_No_data_deo)
```

```
Tasa de pacientes que se colocaron algún producto 27.22%
Tasa de pacientes que no se colocaron algún producto 60.95%
Tasa de pacientes sin datos 11.83%
```

[96]:
```
# Ahora mostramos una gráfica de "pastel" de esta información
labels = 'Sí', 'No', 'Sin datos'
sizes = [tasa_deo, tasa_NoDeo, tasa_No_data_deo]
explode = (0.1, 0, 0)  # Resaltando el porcentaje de pacientes que se colocaron␣
 ↪algún producto

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%0.2f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Esta declaración asegura que se dibuje un círculo

ax1.title.set_text('¿Se colocaron algún producto (desodaorante, pomada)?␣
 ↪(Conjunto de Entrenamiento)')

plt.show()
```



2.0.17 Comentarios

Como se observa en las gráficas anteriores, la mayoría de los campos en estudio tiene datos faltantes.

Los campos completos son: 'Age','Diagnosis'

Los campos Alcohol y Ejercicio no se considerarán para el modelo propuesto y debido a que no aportarían suficiente información, ya que ninguna paciente afirmó haber consumido alcohol o haber hecho ejercicio.

Los demás campos contienen información relevante y la falta de datos en su mayoría es menor al 3% del total, salvo en cuatro (4) casos, donde supera el 10% pero son menores al 20% del total.

2.0.18 Propuestas de Minería de Datos

Con el objetivo de utilizar todos los pacientes para elaborar un modelo que tenga como entradas los datos clínicos, se propone una estrategia de "rellenado" de datos faltantes. En otras palabras, una estrategia de minería de datos para solventar la falta de algunos datos.

En este caso es posible implementar diferentes estrategias, como sustituir el dato faltante por una respuesta que por sentido común podríamos razonablemente inferir. Por ejemplo, los datos faltantes del campo "Complaints" quizá se deban a que al momento de la adquisición del termograma no existió una queja de la paciente, y le pareció irrelevante a la enfermera recavar dicha información. Por tanto tendría sentido "rellenar" estos datos con 0 (no hubo dolor). Esto se podría generalizar a los demás campos. Las desventaja de realizar esto es que al realizar esto estaríamos quizá incluyendo pacientes donde este supuesto no se cumple y alterar de forma "significativa" la capacidad de discriminación del clasificador artificial o modelo propuesto.

Otra estrategia es realizar un promedio de los datos y rellenar con este valor los datos faltantes en el campo de interés. Aunque realizar esto tendría sentido en variables continuas como la temperatura corporal, quizá no tendría "tanto" sentido en variables categóricas (las cuales son todas las demás). Sin embargo, las demás variables son categóricas binarias, es decir, califican si existe o no existe algo. Si se observa desde el punto de vista de la probabilidad, la media de estos valores indicarían la probabilidad de respuesta afirmativa a un evento y sería una forma de poder lidiar con la incertidumbre enlos datos faltantes para estas variables.

La última alternativa es la construcción de un modelo que aproxime la respuesta ideal con base a las demás características de los pacientes conocidos.

Por motivos de practicidad y tiempo de desarrollo se elegirá el promedio de las variables y se dejarán las otras opciones como propuesta de mejora para trabajos posteriores.

[97]:
```
# Realizamos una copia del DataFrame train
train2 = train.copy()
```

[98]:
```
train2.head()
```

[98]:
```
   ID  ... Put some pomade, deodorant or products at breasts or armpits region?
0   1  ...                                                  NaN
1   2  ...                                                  0.0
2   4  ...                                                  NaN
```

```
3   5  ...                                                  NaN
4   6  ...                                                  NaN

[5 rows x 23 columns]
```

[99]:
```
columnas = ['ID', 'Age', 'Diagnosis',
       'Complaints', 'Radiotherapy', 'Plastic surgery',
       'Prosthesis', 'Use of hormone replacement',
       'Is there signal of wart on breast?', 'Body temperature',
       'Smoked', 'Drank coffe',
       'Put some pomade, deodorant or products at breasts or armpits region?']
```

[100]:
```
len(columnas)
```

[100]:
```
13
```

Resultan 13 columnas para el análisis final, una es la salida del modelo (Diagnosis) y ID sólo es para fines de identificación.

[101]:
```
# Entonces almacenamos una nueva tabla con las columnas anteriores
train2 = train2[columnas]
train2.head()
```

[101]:
```
   ID  ...  Put some pomade, deodorant or products at breasts or armpits region?
0   1  ...                                                  NaN
1   2  ...                                                  0.0
2   4  ...                                                  NaN
3   5  ...                                                  NaN
4   6  ...                                                  NaN

[5 rows x 13 columns]
```

2.0.19 mean_age

[102]:
```
# Imprimiendo los valores faltantes en Age
print("Valores faltantes en Age: %d"%train2.Age.isnull().sum())

# Obteniendo media de Age
mean_age = train2.Age.mean()

print(mean_age)
train2['Age'] = train2['Age'].fillna(mean_age)

print("Valores faltantes en Age: %d"%train2.Age.isnull().sum())
```

```
Valores faltantes en Age: 0
61.455621301775146
Valores faltantes en Age: 0
```

[103]:
```
np.save('mean_age.npy', mean_age)
```

2.0.20 mean_complaints

[104]:
```
# Imprimiendo los valores faltantes en Complaints
print("Valores faltantes en Complaints: %d"%train2.Complaints.isnull().sum())

# Obteniendo media de Complaints
mean_complaints = train2.Complaints.mean()

print(mean_complaints)
train2['Complaints'] = train2['Complaints'].fillna(mean_complaints)

print("Valores faltantes en Complaints: %d"%train2.Complaints.isnull().sum())
```

```
Valores faltantes en Complaints: 27
0.5633802816901409
Valores faltantes en Complaints: 0
```

[105]:
```
np.save('mean_complaints.npy', mean_complaints)
```

2.0.21 mean_radio

[106]:
```
# Imprimiendo los valores faltantes en Radiotherapy
print("Valores faltantes en Radiotherapy: %d"%train2.Radiotherapy.isnull().sum())

# Obteniendo media de Radiotherapy
mean_radio = train2.Radiotherapy.mean()

print(mean_radio)
train2['Radiotherapy'] = train2['Radiotherapy'].fillna(mean_radio)

print("Valores faltantes en Radiotherapy: %d"%train2.Radiotherapy.isnull().sum())
```

```
Valores faltantes en Radiotherapy: 3
0.1144578313253012
Valores faltantes en Radiotherapy: 0
```

[107]:
```
np.save('mean_radio.npy', mean_radio)
```

2.0.22 mean_plastic

[108]:
```
# Imprimiendo los valores faltantes en Plastic surgery
print("Valores faltantes en Plastic surgery: %d"%train2['Plastic surgery'].
 ↪isnull().sum())

# Obteniendo media de Plastic surgery
mean_plastic = train2['Plastic surgery'].mean()
```

```
print(mean_plastic)
train2['Plastic surgery'] = train2['Plastic surgery'].fillna(mean_plastic)

print("Valores faltantes en Plastic surgery: %d"%train2['Plastic surgery'].
 ↪isnull().sum())
```

```
Valores faltantes en Plastic surgery: 2
0.0658682634730539
Valores faltantes en Plastic surgery: 0
```

[109]:
```
np.save('mean_plastic.npy', mean_plastic)
```

2.0.23 mean_prost

[110]:
```
# Imprimiendo los valores faltantes en Prosthesis
print("Valores faltantes en Prosthesis: %d"%train2['Prosthesis'].isnull().sum())

# Obteniendo media de Prosthesis
mean_prost = train2['Prosthesis'].mean()

print(mean_prost)
train2['Prosthesis'] = train2['Prosthesis'].fillna(mean_prost)

print("Valores faltantes en Prosthesis: %d"%train2['Prosthesis'].isnull().sum())
```

```
Valores faltantes en Prosthesis: 2
0.023952095808383235
Valores faltantes en Prosthesis: 0
```

[111]:
```
np.save('mean_prost.npy', mean_prost)
```

2.0.24 mean_horm

[112]:
```
# Imprimiendo los valores faltantes en Use of hormone replacement
print("Valores faltantes en Use of hormone replacement: %d"%train2['Use of␣
 ↪hormone replacement'].isnull().sum())

# Obteniendo media de Use of hormone replacement
mean_horm = train2['Use of hormone replacement'].mean()

print(mean_horm)
train2['Use of hormone replacement'] = train2['Use of hormone replacement'].
 ↪fillna(mean_horm)

print("Valores faltantes en Use of hormone replacement: %d"%train2['Use of␣
 ↪hormone replacement'].isnull().sum())
```

```
Valores faltantes en Use of hormone replacement: 3
0.21084337349397592
Valores faltantes en Use of hormone replacement: 0
```

[113]:
```
np.save('mean_horm.npy', mean_horm)
```

2.0.25 mean_wart

[114]:
```
# Imprimiendo los valores faltantes en Is there signal of wart on breast?
print("Valores faltantes en Is there signal of wart on breast?: %d"%train2['Is␣
 ↪there signal of wart on breast?'].isnull().sum())

# Obteniendo media de Is there signal of wart on breast?
mean_wart = train2['Is there signal of wart on breast?'].mean()

print(mean_wart)
train2['Is there signal of wart on breast?'] = train2['Is there signal of wart␣
 ↪on breast?'].fillna(mean_wart)

print("Valores faltantes en Is there signal of wart on breast?: %d"%train2['Is␣
 ↪there signal of wart on breast?'].isnull().sum())
```

```
Valores faltantes en Is there signal of wart on breast?: 2
0.17964071856287425
Valores faltantes en Is there signal of wart on breast?: 0
```

[115]:
```
np.save('mean_wart.npy', mean_wart)
```

2.0.26 mean_temp

[116]:
```
# Imprimiendo los valores faltantes en Body temperature
print("Valores faltantes en Body temperature: %d"%train2['Body temperature'].
 ↪isnull().sum())

# Obteniendo media de Body temperature
mean_temp = train2['Body temperature'].mean()

print(mean_temp)
train2['Body temperature'] = train2['Body temperature'].fillna(mean_temp)

print("Valores faltantes en Body temperature: %d"%train2['Body temperature'].
 ↪isnull().sum())
```

```
Valores faltantes en Body temperature: 5
35.46402439024391
Valores faltantes en Body temperature: 0
```

[117]:
```
np.save('mean_temp.npy', mean_temp)
```

2.0.27 mean_smoke

[118]:
```
# Imprimiendo los valores faltantes en Smoked
print("Valores faltantes en Smoked: %d"%train2['Smoked'].isnull().sum())

# Obteniendo media de Smoked
mean_smoke = train2['Smoked'].mean()

print(mean_smoke)
train2['Smoked'] = train2['Smoked'].fillna(mean_smoke)

print("Valores faltantes en Smoked: %d"%train2['Smoked'].isnull().sum())
```

```
Valores faltantes en Smoked: 20
0.06040268456375839
Valores faltantes en Smoked: 0
```

[119]:
```
np.save('mean_smoke.npy', mean_smoke)
```

2.0.28 mean_coffee

[120]:
```
# Imprimiendo los valores faltantes en Drank coffe
print("Valores faltantes en Drank coffe: %d"%train2['Drank coffe'].isnull().
 ↪sum())

# Obteniendo media de Drank coffe
mean_coffee = train2['Drank coffe'].mean()

print(mean_coffee)
train2['Drank coffe'] = train2['Drank coffe'].fillna(mean_coffee)

print("Valores faltantes en Drank coffe: %d"%train2['Drank coffe'].isnull().
 ↪sum())
```

```
Valores faltantes en Drank coffe: 20
0.087248322147651
Valores faltantes en Drank coffe: 0
```

[121]:
```
np.save('mean_coffee.npy', mean_coffee)
```

2.0.29 mean_deo

[122]:
```
# Imprimiendo los valores faltantes en Put some pomade, deodorant or products at␣
 ↪breasts or armpits region?
print("Valores faltantes en Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?: %d"%train2['Put some pomade, deodorant or products at breasts␣
 ↪or armpits region?'].isnull().sum())
```

```
# Obteniendo media de Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?
mean_deo = train2['Put some pomade, deodorant or products at breasts or armpits␣
 ↪region?'].mean()

print(mean_deo)
train2['Put some pomade, deodorant or products at breasts or armpits region?'] =␣
 ↪train2['Put some pomade, deodorant or products at breasts or armpits region?'].
 ↪fillna(mean_deo)

print("Valores faltantes en Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?: %d"%train2['Put some pomade, deodorant or products at breasts␣
 ↪or armpits region?'].isnull().sum())
```

```
Valores faltantes en Put some pomade, deodorant or products at breasts or
armpits region?: 20
0.3087248322147651
Valores faltantes en Put some pomade, deodorant or products at breasts or
armpits region?: 0
```

[123]:
```
np.save('mean_deo.npy', mean_deo)
```

2.1 Modificando valores de Diagnosis

[124]:
```
train2['Diagnosis'].replace(to_replace=['Healthy','Sick'],␣
 ↪value=[0,1],inplace=True)
```

[125]:
```
train2.head()
```

[125]:

	ID	...	Put some pomade, deodorant or products at breasts or armpits region?
0	1	...	0.308725
1	2	...	0.000000
2	4	...	0.308725
3	5	...	0.308725
4	6	...	0.308725

[5 rows x 13 columns]

2.2 Preparando los datos de validación

Ahora se preparan los datos de validación.

Los datos faltantes en este conjunto se calculan con los promedio obtenidos previamente, es decir, en el conjunto de entrenamiento.

[126]:
```
# Creamos una copia del DataFrame test
test2 = test.copy()
```

[127]:
```
# Almacenamos únicamente los valores de interés
test2 = test2[columnas]
test2.head()
```

[127]:
```
   ID  ...  Put some pomade, deodorant or products at breasts or armpits region?
0  12  ...                                                0.0
1  19  ...                                                0.0
2  20  ...                                                0.0
3  22  ...                                                0.0
4  32  ...                                                0.0

[5 rows x 13 columns]
```

[128]:
```
# Modificando salidas de Diagnosis en test
test2['Diagnosis'].replace(to_replace=['Healthy','Sick'],␣
 ↪value=[0,1],inplace=True)
```

2.2.1 Age

[129]:
```
# Imprimiendo los valores faltantes en Age
print("Valores faltantes en Age: %d"%test2.Age.isnull().sum())

test2['Age'] = test2['Age'].fillna(mean_age)

print("Valores faltantes en Age: %d"%test2.Age.isnull().sum())
```

```
Valores faltantes en Age: 0
Valores faltantes en Age: 0
```

2.2.2 Complaints

[130]:
```
# Imprimiendo los valores faltantes en Complaints
print("Valores faltantes en Complaints: %d"%test2.Complaints.isnull().sum())

test2['Complaints'] = test2['Complaints'].fillna(mean_complaints)

print("Valores faltantes en Complaints: %d"%test2.Complaints.isnull().sum())
```

```
Valores faltantes en Complaints: 3
Valores faltantes en Complaints: 0
```

2.2.3 Radiotherapy

[131]:
```
# Imprimiendo los valores faltantes en Radiotherapy
print("Valores faltantes en Radiotherapy: %d"%test2.Radiotherapy.isnull().sum())

test2['Radiotherapy'] = test2['Radiotherapy'].fillna(mean_radio)
```

```
print("Valores faltantes en Radiotherapy: %d"%test2.Radiotherapy.isnull().sum())
```

```
Valores faltantes en Radiotherapy: 0
Valores faltantes en Radiotherapy: 0
```

2.2.4 Plastic surgery

[132]:
```
# Imprimiendo los valores faltantes en Plastic surgery
print("Valores faltantes en Plastic surgery: %d"%test2['Plastic surgery'].
 ↪isnull().sum())

test2['Plastic surgery'] = test2['Plastic surgery'].fillna(mean_plastic)

print("Valores faltantes en Plastic surgery: %d"%test2['Plastic surgery'].
 ↪isnull().sum())
```

```
Valores faltantes en Plastic surgery: 0
Valores faltantes en Plastic surgery: 0
```

2.2.5 Prosthesis

[133]:
```
# Imprimiendo los valores faltantes en Prosthesis
print("Valores faltantes en Prosthesis: %d"%test2['Prosthesis'].isnull().sum())

test2['Prosthesis'] = test2['Prosthesis'].fillna(mean_prost)

print("Valores faltantes en Prosthesis: %d"%test2['Prosthesis'].isnull().sum())
```

```
Valores faltantes en Prosthesis: 0
Valores faltantes en Prosthesis: 0
```

2.2.6 Use of hormone replacement

[134]:
```
# Imprimiendo los valores faltantes en Use of hormone replacement
print("Valores faltantes en Use of hormone replacement: %d"%test2['Use of␣
 ↪hormone replacement'].isnull().sum())

test2['Use of hormone replacement'] = test2['Use of hormone replacement'].
 ↪fillna(mean_horm)

print("Valores faltantes en Use of hormone replacement: %d"%test2['Use of␣
 ↪hormone replacement'].isnull().sum())
```

```
Valores faltantes en Use of hormone replacement: 3
Valores faltantes en Use of hormone replacement: 0
```

2.2.7 Is there signal of wart on breast?

[135]:
```
# Imprimiendo los valores faltantes en Is there signal of wart on breast?
print("Valores faltantes en Is there signal of wart on breast?: %d"%test2['Is␣
 ↪there signal of wart on breast?'].isnull().sum())

test2['Is there signal of wart on breast?'] = test2['Is there signal of wart on␣
 ↪breast?'].fillna(mean_wart)

print("Valores faltantes en Is there signal of wart on breast?: %d"%test2['Is␣
 ↪there signal of wart on breast?'].isnull().sum())
```

```
Valores faltantes en Is there signal of wart on breast?: 0
Valores faltantes en Is there signal of wart on breast?: 0
```

2.2.8 Body temperature

[136]:
```
# Imprimiendo los valores faltantes en Body temperature
print("Valores faltantes en Body temperature: %d"%test2['Body temperature'].
 ↪isnull().sum())

test2['Body temperature'] = test2['Body temperature'].fillna(mean_temp)

print("Valores faltantes en Body temperature: %d"%test2['Body temperature'].
 ↪isnull().sum())
```

```
Valores faltantes en Body temperature: 0
Valores faltantes en Body temperature: 0
```

2.2.9 Smoked

[137]:
```
# Imprimiendo los valores faltantes en Smoked
print("Valores faltantes en Smoked: %d"%test2['Smoked'].isnull().sum())

test2['Smoked'] = test2['Smoked'].fillna(mean_smoke)

print("Valores faltantes en Smoked: %d"%test2['Smoked'].isnull().sum())
```

```
Valores faltantes en Smoked: 0
Valores faltantes en Smoked: 0
```

2.2.10 Drank coffe

[138]:
```
# Imprimiendo los valores faltantes en Drank coffe
print("Valores faltantes en Drank coffe: %d"%test2['Drank coffe'].isnull().sum())

test2['Drank coffe'] = test2['Drank coffe'].fillna(mean_coffee)
```

```
print("Valores faltantes en Drank coffe: %d"%test2['Drank coffe'].isnull().sum())
```

```
Valores faltantes en Drank coffe: 0
Valores faltantes en Drank coffe: 0
```

2.2.11 Put some pomade, deodorant or products at breasts or armpits region?

[139]:
```
# Imprimiendo los valores faltantes en Put some pomade, deodorant or products at␣
 ↪breasts or armpits region?
print("Valores faltantes en Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?: %d"%test2['Put some pomade, deodorant or products at breasts␣
 ↪or armpits region?'].isnull().sum())

test2['Put some pomade, deodorant or products at breasts or armpits region?'] =␣
 ↪test2['Put some pomade, deodorant or products at breasts or armpits region?'].
 ↪fillna(mean_deo)

print("Valores faltantes en Put some pomade, deodorant or products at breasts or␣
 ↪armpits region?: %d"%test2['Put some pomade, deodorant or products at breasts␣
 ↪or armpits region?'].isnull().sum())
```

```
Valores faltantes en Put some pomade, deodorant or products at breasts or
armpits region?: 0
Valores faltantes en Put some pomade, deodorant or products at breasts or
armpits region?: 0
```

2.3 Estándarizando las características

[140]:
```
# Esta lista es para la construcción de la X o matriz con características de␣
 ↪entrada al modeloa a ajustar
Features = ['Age',
       'Complaints', 'Radiotherapy', 'Plastic surgery',
       'Prosthesis', 'Use of hormone replacement',
       'Is there signal of wart on breast?', 'Body temperature',
       'Smoked', 'Drank coffe',
       'Put some pomade, deodorant or products at breasts or armpits region?']
```

[141]:
```
X_train = train2[Features].to_numpy()
X_train.shape
```

[141]: (169, 11)

Estandarizamos X_train y X_test Para esto nos apoyamos de un de scikit-Learn llamado Standard-scaler.

El objetivo de realizar este paso es eficientizar la optimización del modelo evitando disparidad en el intervalo de las dimensiones de las características lo cual podría implicar en un aprendizaje "lento".

Primero obtenemos los valores de necesarios para la estandarización de X_train y los guardamos ya que estos serán necesarios para la construcción del prototipo final.

La estandarización es el proceso mediante el cual una variable aleatoria (característica) se aproxima a la distribución normal con media igual a cero (centrada en el origen).

z = (x - mhu)/sigma

Donde *z* es el valor resultante de la estandarización, *x* es la variable aleatoria, *mhu* es la media (muestral) de esa variable aleatoria y *sigma* (muestral) su desviación estandar.

Se hace hincapié en que los descriptores *mhu* y *sigma* son muestrales y no poblacionales, ya que podría causar confusión debido a que en la literatura se utilizan como descriptores poblacionales.

[142]:
```
desc_standar=preprocessing.StandardScaler().fit(X_train)
```

[143]:
```
mean_X_data_clinic = desc_standar.mean_
print(mean_X_data_clinic)
```

```
[6.14556213e+01 5.63380282e-01 1.14457831e-01 6.58682635e-02
 2.39520958e-02 2.10843373e-01 1.79640719e-01 3.54640244e+01
 6.04026846e-02 8.72483221e-02 3.08724832e-01]
```

[144]:
```
std_X_data_clinic = np.sqrt(desc_standar.var_)
print(std_X_data_clinic)
```

```
[13.23924531  0.454625    0.31552812  0.24657955  0.15199252  0.40427071
  0.38160963  0.76149085  0.22369114  0.26497484  0.43377153]
```

[145]:
```
# Gurdando los descriptores de interés
np.save("mean_X_data_clinic.npy", mean_X_data_clinic)
np.save("std_X_data_clinic.npy", std_X_data_clinic)
```

[146]:
```
# Matriz estandarizada para el entrenamiento
X_train2 = preprocessing.StandardScaler().fit(X_train).transform(X_train)
```

[147]:
```
# Matriz de validación sin estandarizar
X_test = test2[Features].to_numpy()
X_test.shape
```

[147]:
```
(43, 11)
```

[148]:
```
test.ID.count()
```

[148]:
```
43
```

[149]:
```
# Positivos y negatovos para validación
test.Diagnosis.value_counts()
```

[149]:
```
Healthy    32
Sick       11
```

```
Name: Diagnosis, dtype: int64
```

En el conjunto de validación 74.41% de los pacientes son negativos o sanos y 25.58% son positivos o enfermos.

[150]:
```
# Estandarizando la matriz de validación uutilizanndo los descriptores de la matriz de validación.
X_test2 = (X_test-mean_X_data_clinic)/mean_X_data_clinic
```

[151]:
```
X_test2[:,0]==((X_test[:,0]-mean_X_data_clinic[0])/mean_X_data_clinic[0])
```

[151]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True])
```

NOTA IMPORTANTE

Es posible demostrar que dada una muestra de n elementos media es igual a la media de un conjunto definido con los mismos n elementos más k elementos (k medias del conjunto de n elementos). Por tanto es posible ocupar la media calculada con NaN o null o sin NaN (osea los sustituidos por la media).

3 Entrenando los modelos

Ya que hemos obtenido las matrices estandarizadas ahora las utilizamos para entrenar diferentes modelos

Propondremos cuatro (4) modelos:

1. Tree Decision
2. Random Forest Tree
3. Support Vector Machine
4. Logistic Regression
5. Red Neuronal con 3 capas escondidas

Las métricas que se emplearán para evaluar el desmpeño de los modelos serán:

1. Accuracy (Exactitud)
2. Precision (Precisión)
3. Recall (Sensibilidad)
4. F1-Score
5. AUC*

*Sólo aplica para los modelos 4 y 5. Los modelos 1, 2 y 3 no pueden ser evaluados con esta métrica debido a la definición misma de la métrica AUC (Área Bajo la Curva ROC).

La elección del mejor modelo se hará con base a su rendimiento en la métrica F1-Score.

3.0.1 Tree Decision

Este modelo no lineal se construye a partir de las caracteríscas. En un plano 2D con dos características, el modelo serían una serie de líneas HORIZONTALES y VERTICALES las cuales definen particiones donde se agrupan los puntos que representan a una u otra clase. El objetivo es encontrar la cantidad óptima de particiones que nos permitan discrimanar "lo mejor posible" las clases de salida.

El algoritmo para alcanzar el mejor modelo es:

1. Escoger un atributo de la base de datos que sea el "más prometedor" para separar a las clases de salida.
2. Calcular el peso que este atributo seleccionado tiene al dividir los datos.
3. Dividir los datos con base la mejor valor (umbral) del atributo.
4. Ir nuevamente al paso uno y repetir para el resto de los atributos.

[152]:
```
# Importando métricas de scikit learn}
from sklearn.metrics import f1_score, precision_score, recall_score,␣
 ↪confusion_matrix, roc_curve, auc, accuracy_score
```

[153]:
```
# Importando algunps método y bibliotecas necesarios para el modelo
from sklearn.externals.six import StringIO
import pydotplus
import matplotlib.image as mpimg
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31:
FutureWarning: The module is deprecated in version 0.21 and will be removed in
version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
```

Se define el arbol de decision con auxilio de la biblioteca ScikitLearn.

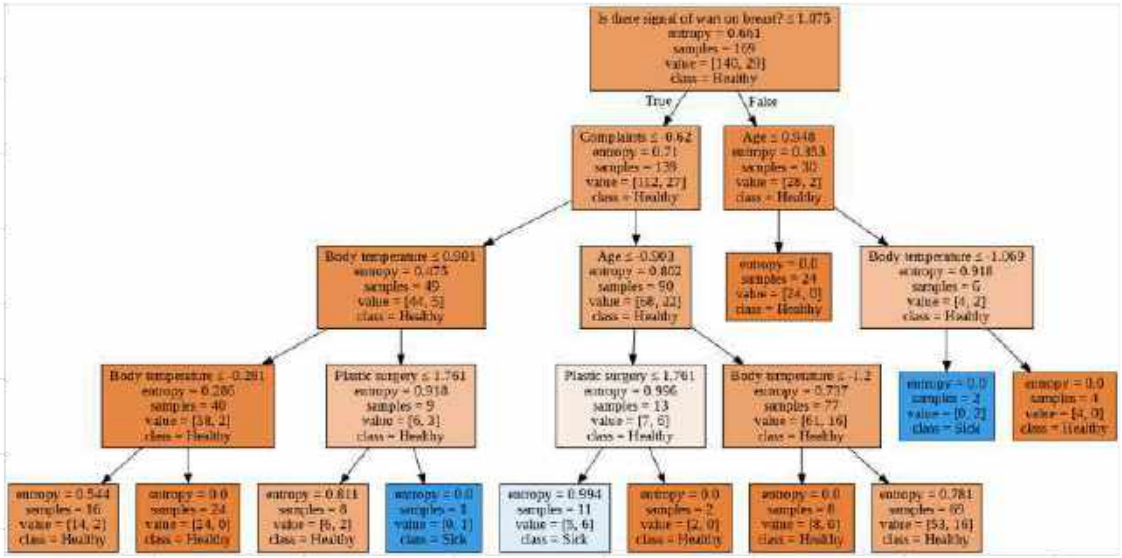
Mientras mayor sea la profundidad del árbol menor tenderá a ser la entropía ya que el modelo al ser más complejo podrá (teóricamente) disminuir la incertidumbre hacia una respuesta, dando más peso en los nodos más profundos a una clase en lugar de otra aumentando la capacidad de clacificación del modelo.

[154]:
```
# En esta lista se almacenan los diferentes modelos
tree_models = []
f1_tree_list = []

# Realizamos un ciclo para evaluar el rendimiento de diferentes árboles
# Estos árboles se modifican el parámetro max_depth con los 4, 5, 6, 7, 8, 9 y 10

for i in range(4, 16):

  # Definimos el modelo
```

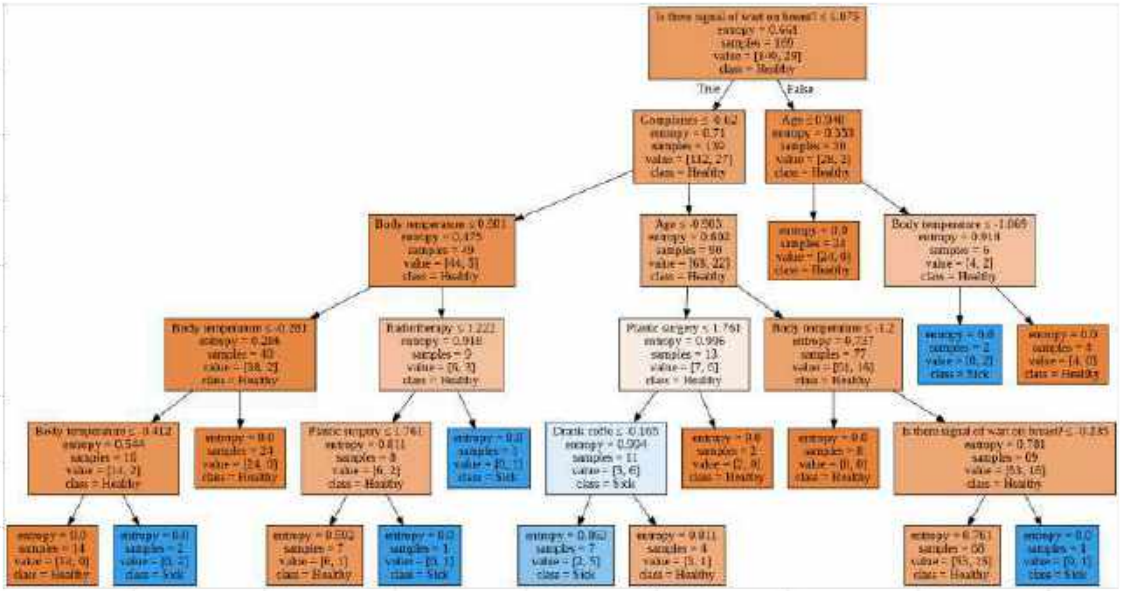
```
 model = DecisionTreeClassifier(criterion="entropy", max_depth = i)

 # Ajustamos el modelo
 model.fit(X_train2,train2['Diagnosis'].to_numpy())

 print("Decision Tree max_depth = %d"%i)
 dot_data = StringIO()
 filename = "clinic_data_tree"+str(i)+".png"
 featureNames = Features
 targetNames = train["Diagnosis"].unique().tolist()
 out=tree.export_graphviz(model,feature_names=featureNames, out_file=dot_data,␣
↪class_names= np.unique(train['Diagnosis']), filled=True, ␣
↪special_characters=True,rotate=False)
 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
 graph.write_png(filename)
 img = mpimg.imread(filename)
 plt.figure(figsize=(100, 200))
 plt.imshow(img,interpolation='nearest')
 plt.show()

 # Calculando las predicciones Y_train_pred y Y_test_pred
 Y_train_pred = model.predict(X_train2)
 Y_test_pred = model.predict(X_test2)

 # Desplegando métrica del rendimiento en el entrenamiento
 print()
 print("MÉTRICAS EN ENTRENAMIENTO")
 print("Accuracy: %0.2f%%"%(100*accuracy_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
 print("Precision: %0.2f%%"%(100*precision_score(train2['Diagnosis'].
↪to_numpy(), Y_train_pred)))
 print("Recall: %0.2f%%"%(100*recall_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
 print("F1-Score: %0.2f%%"%(100*f1_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
 print()
 print("MÉTRICAS EN VALIDACIÓN")
 print("Accuracy: %0.2f%%"%(100*accuracy_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print("Precision: %0.2f%%"%(100*precision_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print("Recall: %0.2f%%"%(100*recall_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print("F1-Score: %0.2f%%"%(100*f1_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print()
```

```
  print()

  tree_models.append(model)
  f1_tree_list.append(f1_score(test2['Diagnosis'].to_numpy(), Y_test_pred))
```

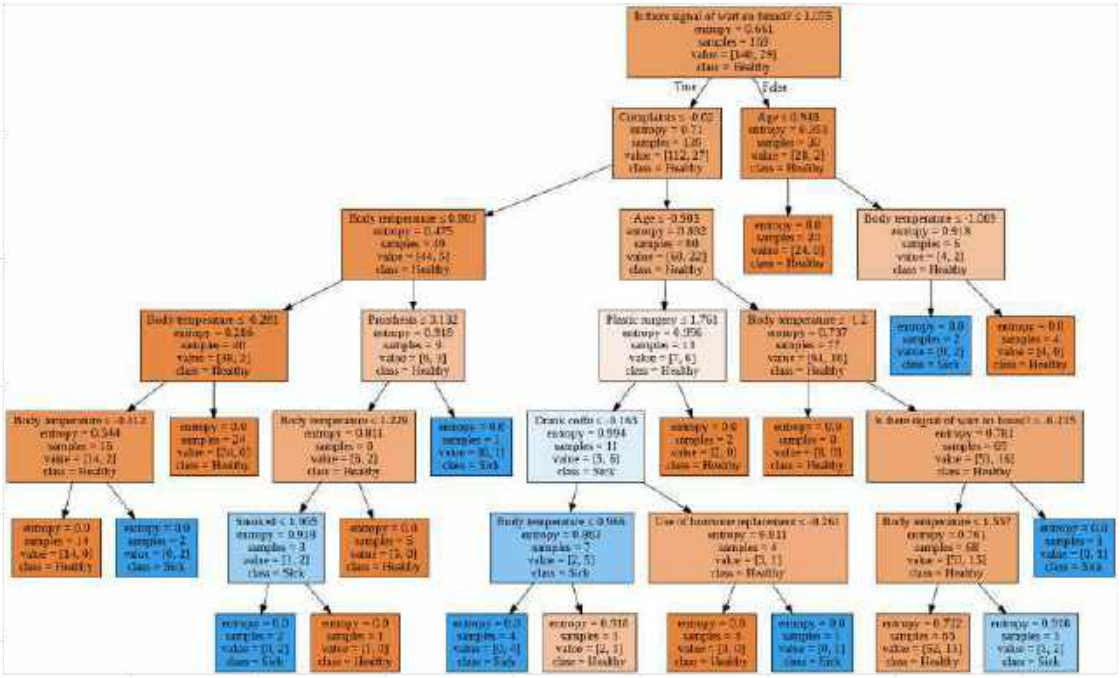
```
Decision Tree max_depth = 4
```



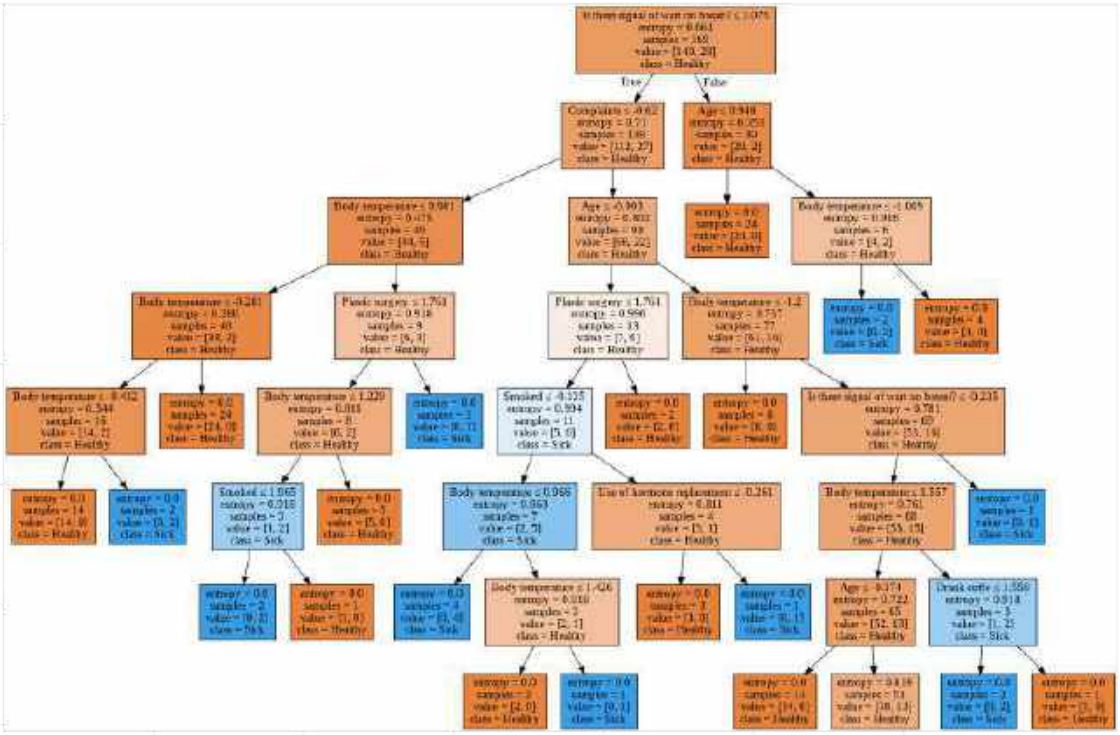

```
MÉTRICAS EN ENTRENAMIENTO
Accuracy: 85.21%
Precision: 64.29%
Recall: 31.03%
F1-Score: 41.86%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


Decision Tree max_depth = 5

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



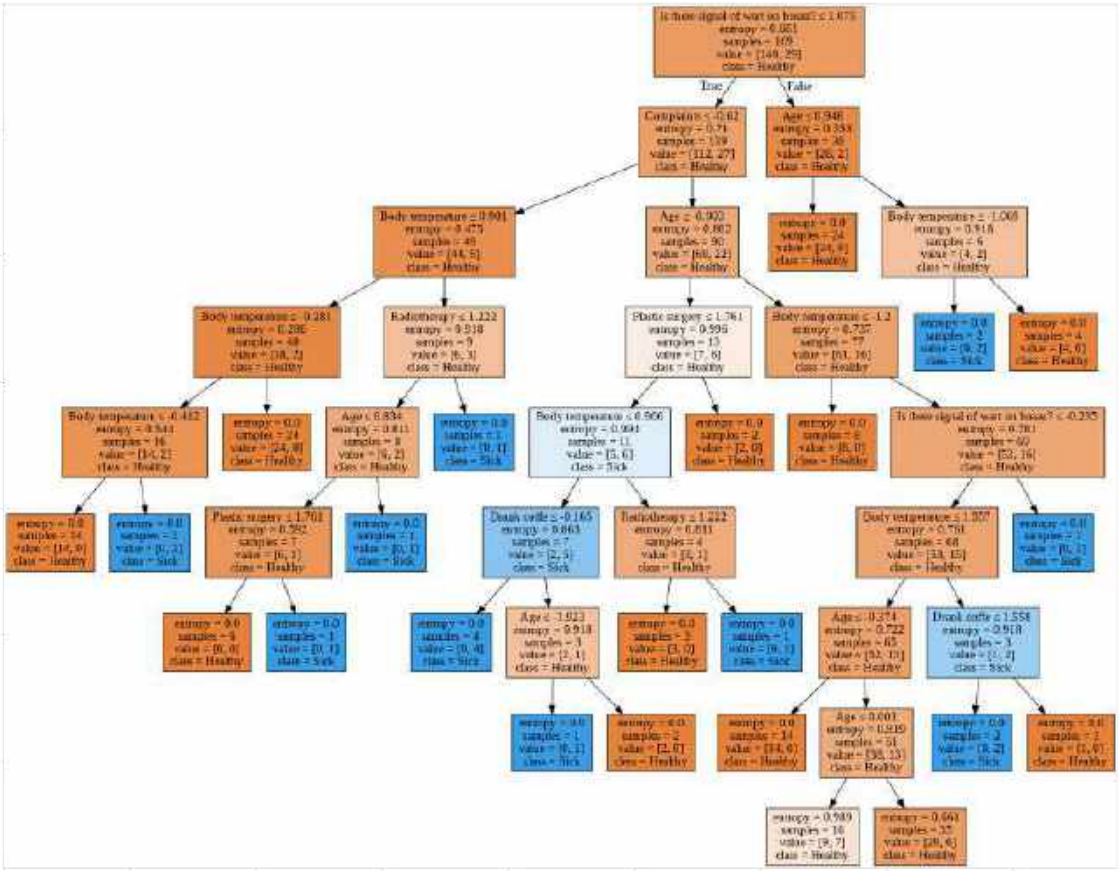
```
MÉTRICAS EN ENTRENAMIENTO
Accuracy: 88.76%
Precision: 85.71%
Recall: 41.38%
F1-Score: 55.81%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


Decision Tree max_depth = 6

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



```
MÉTRICAS EN ENTRENAMIENTO
Accuracy: 91.12%
Precision: 93.75%
Recall: 51.72%
F1-Score: 66.67%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


Decision Tree max_depth = 7

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



```
MÉTRICAS EN ENTRENAMIENTO
Accuracy: 92.31%
Precision: 100.00%
Recall: 55.17%
F1-Score: 71.11%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


Decision Tree max_depth = 8

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

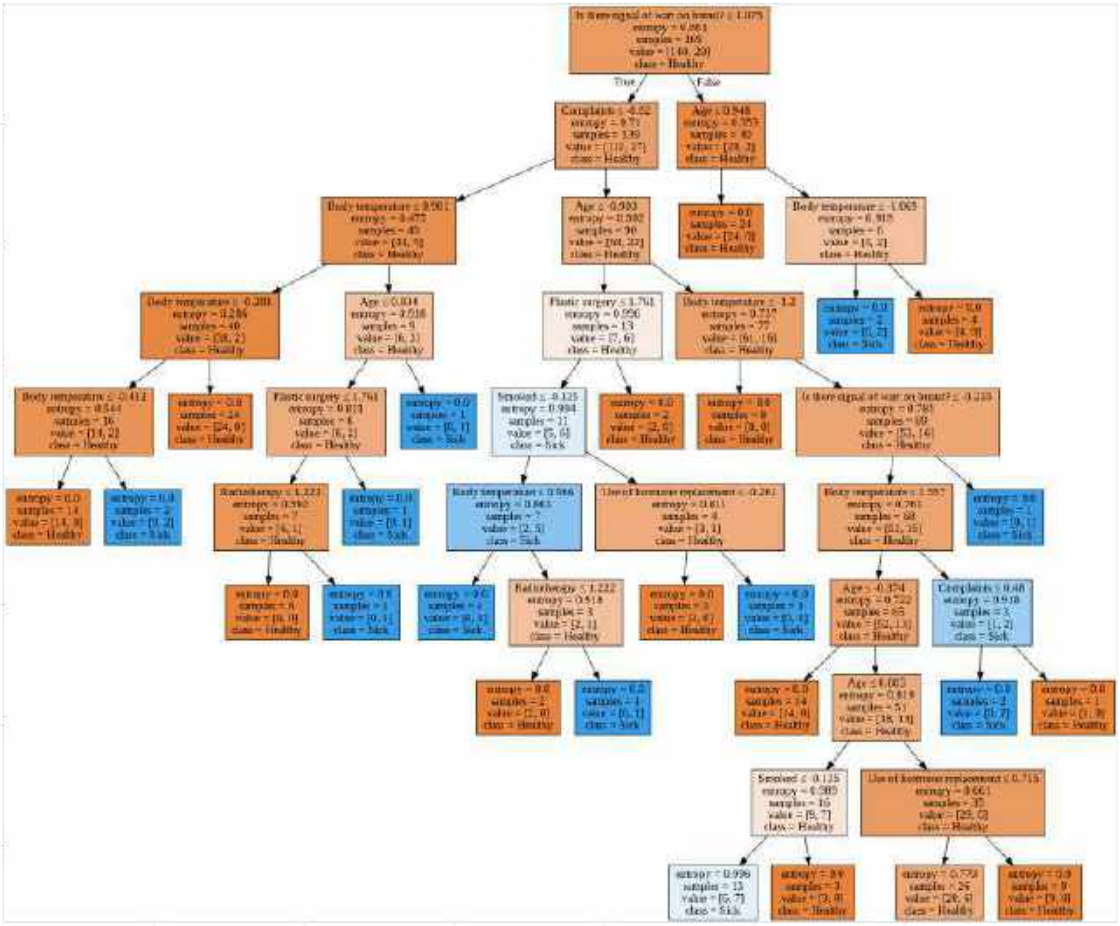


MÉTRICAS EN ENTRENAMIENTO
Accuracy: 92.31%
Precision: 100.00%
Recall: 55.17%
F1-Score: 71.11%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

Decision Tree max_depth = 9

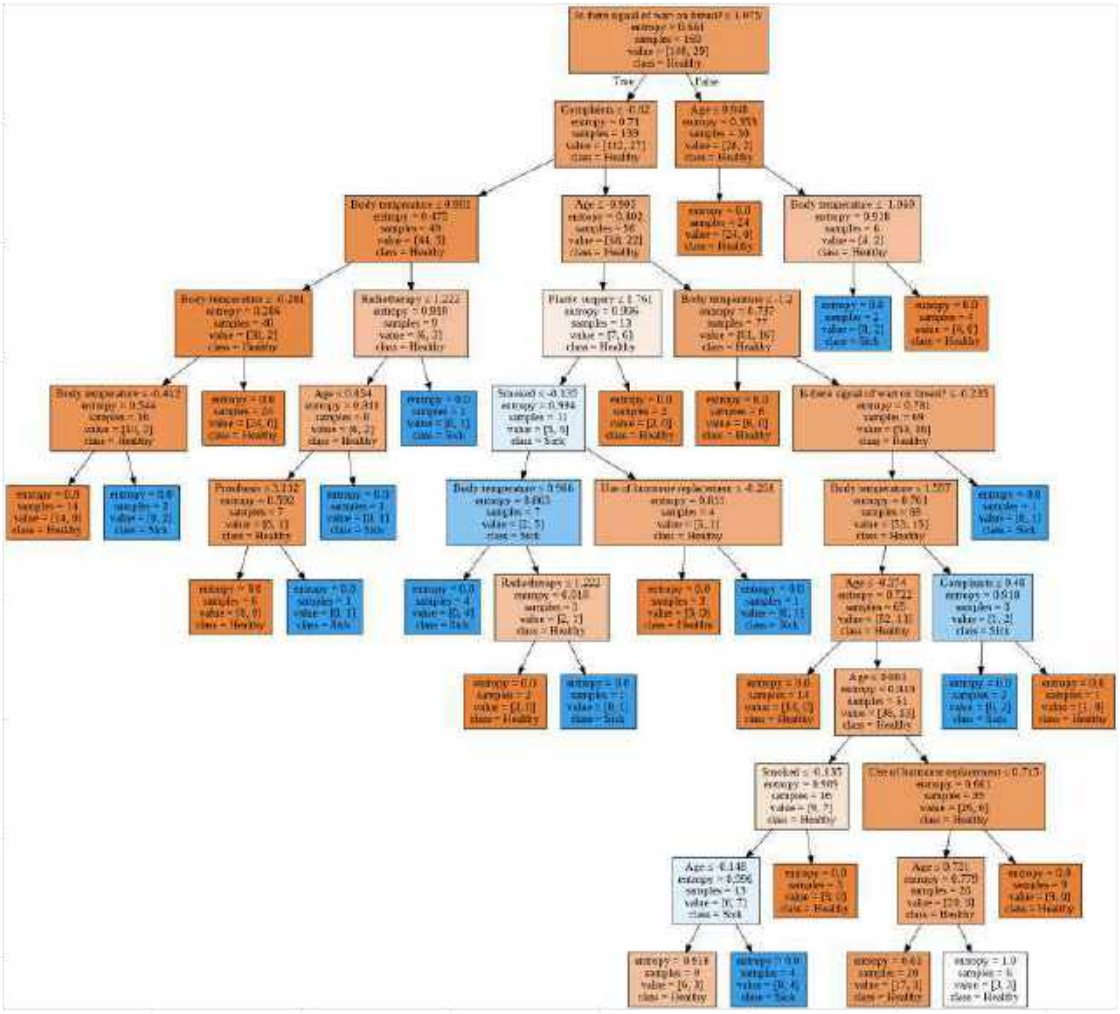
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))



MÉTRICAS EN ENTRENAMIENTO
Accuracy: 92.90%
Precision: 79.31%
Recall: 79.31%
F1-Score: 79.31%

MÉTRICAS EN VALIDACIÓN
Accuracy: 72.09%
Precision: 45.45%
Recall: 45.45%
F1-Score: 45.45%

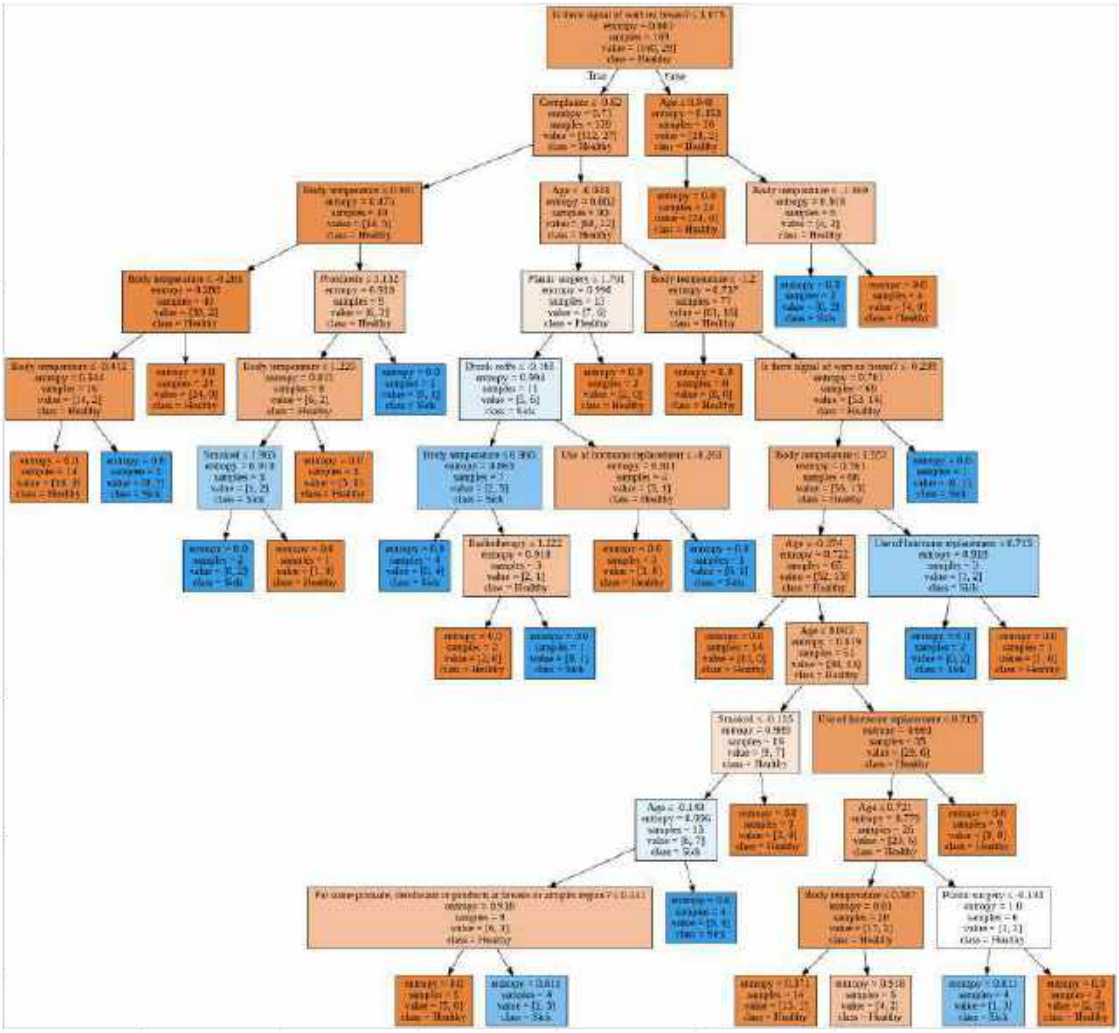
Decision Tree max_depth = 10



MÉTRICAS EN ENTRENAMIENTO
Accuracy: 94.67%
Precision: 100.00%
Recall: 68.97%
F1-Score: 81.63%

MÉTRICAS EN VALIDACIÓN
Accuracy: 72.09%
Precision: 42.86%
Recall: 27.27%
F1-Score: 33.33%

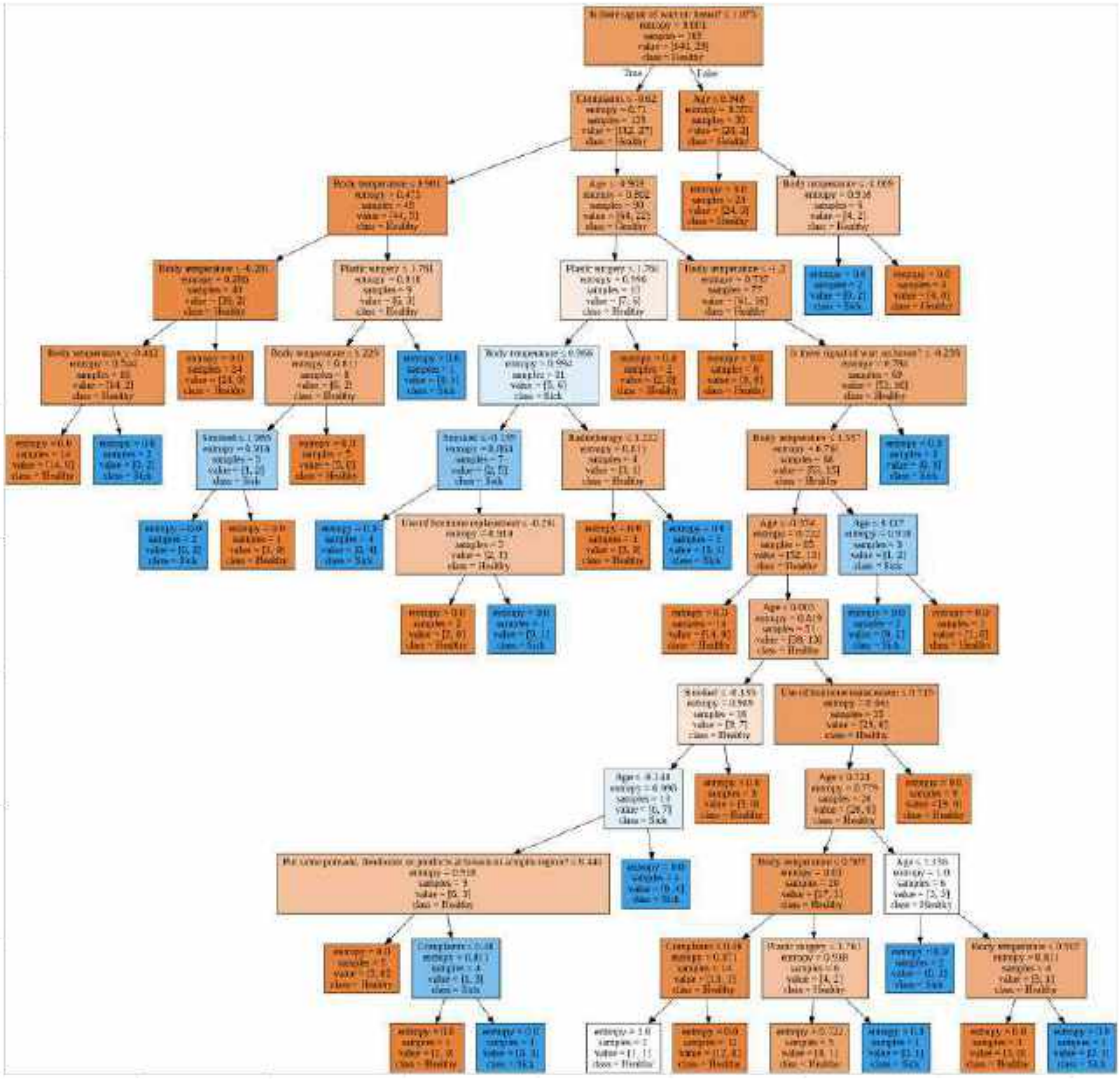
Decision Tree max_depth = 11



MÉTRICAS EN ENTRENAMIENTO
Accuracy: 97.04%
Precision: 92.86%
Recall: 89.66%
F1-Score: 91.23%

MÉTRICAS EN VALIDACIÓN
Accuracy: 69.77%
Precision: 37.50%
Recall: 27.27%
F1-Score: 31.58%

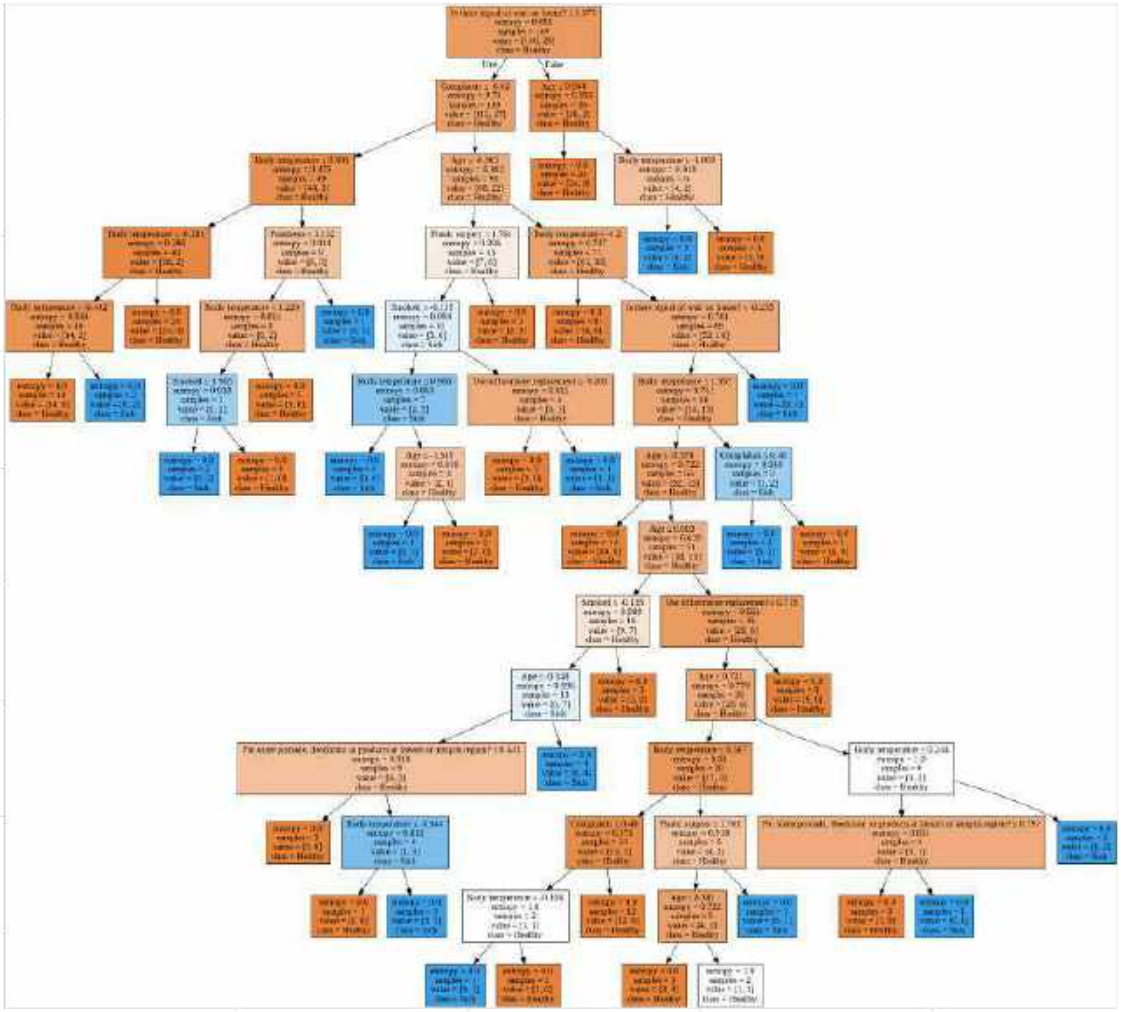
Decision Tree max_depth = 12



MÉTRICAS EN ENTRENAMIENTO
Accuracy: 98.82%
Precision: 100.00%
Recall: 93.10%
F1-Score: 96.43%

MÉTRICAS EN VALIDACIÓN
Accuracy: 69.77%
Precision: 37.50%
Recall: 27.27%
F1-Score: 31.58%

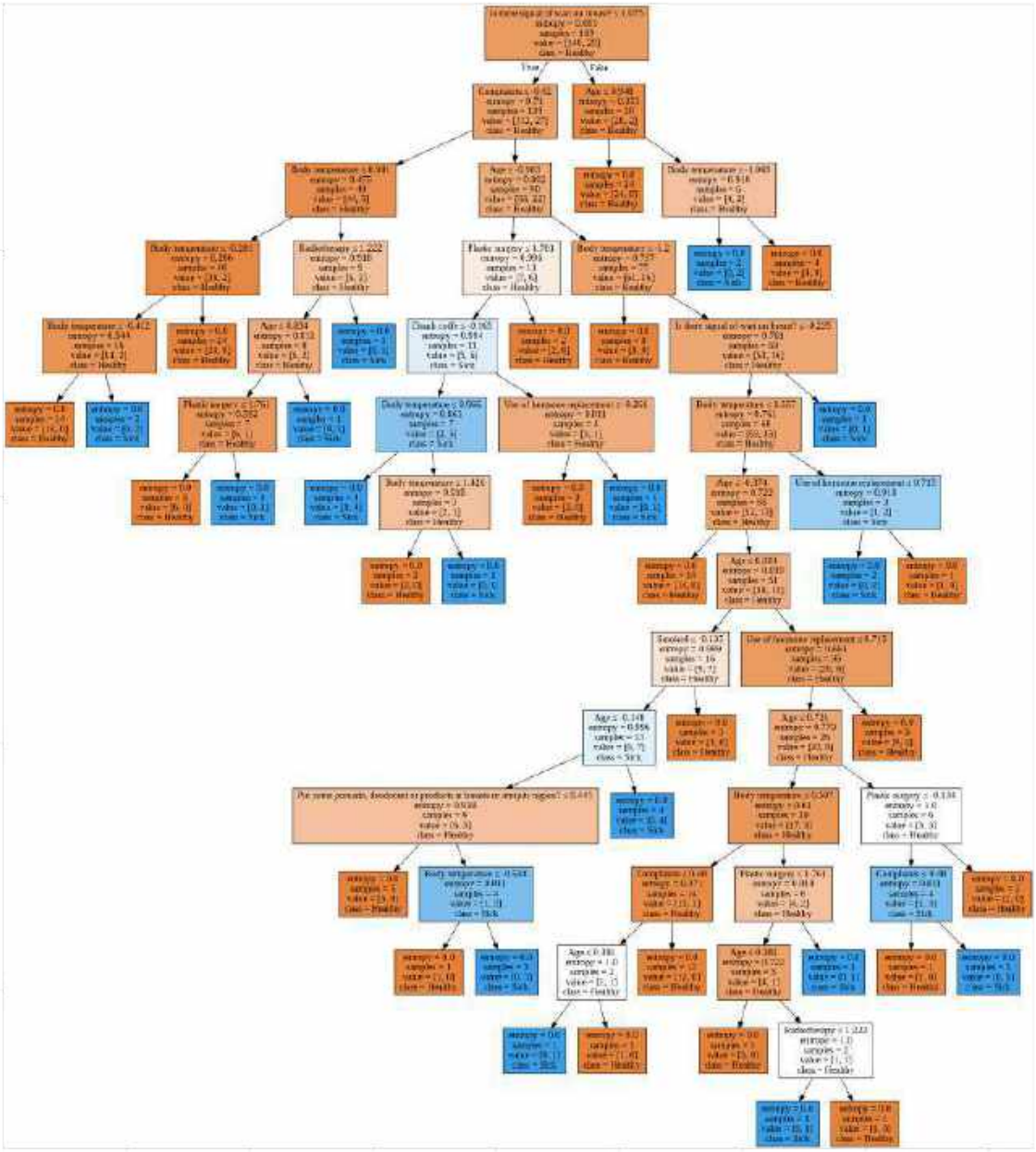
Decision Tree max_depth = 13



MÉTRICAS EN ENTRENAMIENTO
Accuracy: 99.41%
Precision: 100.00%
Recall: 96.55%
F1-Score: 98.25%

MÉTRICAS EN VALIDACIÓN
Accuracy: 69.77%
Precision: 37.50%
Recall: 27.27%
F1-Score: 31.58%

Decision Tree max_depth = 14

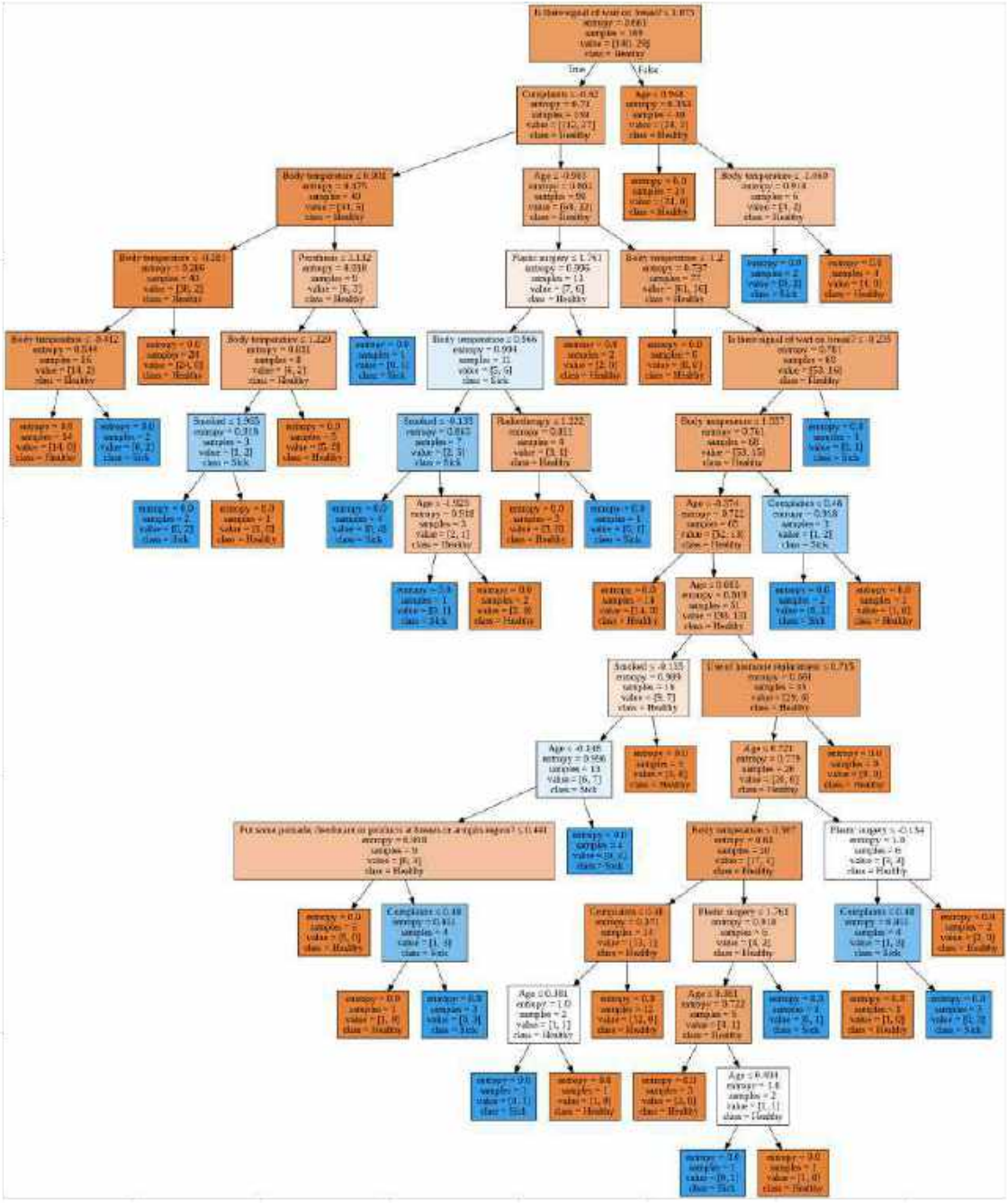


MÉTRICAS EN ENTRENAMIENTO
Accuracy: 100.00%
Precision: 100.00%
Recall: 100.00%
F1-Score: 100.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 67.44%
Precision: 33.33%

Recall: 27.27%
F1-Score: 30.00%

Decision Tree max_depth = 15



MÉTRICAS EN ENTRENAMIENTO

```
Accuracy: 100.00%
Precision: 100.00%
Recall: 100.00%
F1-Score: 100.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 67.44%
Precision: 33.33%
Recall: 27.27%
F1-Score: 30.00%
```

[155]:
```
f1_tree_list = np.array(f1_tree_list)
print("max_depth con mejor desempeño: %d"%(np.where(f1_tree_list==np.
 ↪max(f1_tree_list))[0][0]+4))
print("Mejor F1-Score en validación: %0.2f%%"%(100*f1_tree_list[np.
 ↪where(f1_tree_list==np.max(f1_tree_list))[0][0]]))
```

```
max_depth con mejor desempeño: 9
Mejor F1-Score en validación: 45.45%
```

Con base en los resultados anteriores, el penúltimo modelo (max_depth = 9) fue el que mejor f1-Score (45.45%) obtuvo en la validación.

Como se puede observar en los resultado existe evidencia en el experimento que indica a que quizá el aumento del parámetro max_depth traería mejores resultados.

3.0.2 Random Forest

Este modelo son un conjunto de árboles de decisión que se implementan en subconjuntos del conjunto de entrenamiento. El promedio del rendimiento de cada uno de estos árboles es el valor resultante.

[156]:
```
# Importamos el método auxiliar en scikit learn
from sklearn.ensemble import RandomForestClassifier
```

Se realizará el experimento con base en el numero de árboles posibles. Por default la biblioteca utiliza 100, pero los datos que tenemos son menos de 200.

[157]:
```
# Realizamos for para probar con diferentes valors de número de árboles␣
 ↪(n_estimators)

# Definimos los números de árboles a testear
n_trees = [2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 50]
forest_models = []
f1_forest_list = []

for i in n_trees:
```

```
 # Definimos el modelo
 model = RandomForestClassifier(criterion="entropy", n_estimators =  i,␣
↪max_depth=9)

 # Ajustamos el modelo
 model.fit(X_train2,train2['Diagnosis'].to_numpy())

 print("RANDOM FOREST n_estimators = %d"%i)

 # Calculando las predicciones Y_train_pred y Y_test_pred
 Y_train_pred = model.predict(X_train2)
 Y_test_pred = model.predict(X_test2)

 # Desplegando métrica del rendimiento en el entrenamiento
 print()
 print("MÉTRICAS EN ENTRENAMIENTO")
 print("Accuracy: %0.2f%%"%(100*accuracy_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
 print("Precision: %0.2f%%"%(100*precision_score(train2['Diagnosis'].
↪to_numpy(), Y_train_pred)))
 print("Recall: %0.2f%%"%(100*recall_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
 print("F1-Score: %0.2f%%"%(100*f1_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
 print()
 print("MÉTRICAS EN VALIDACIÓN")
 print("Accuracy: %0.2f%%"%(100*accuracy_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print("Precision: %0.2f%%"%(100*precision_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print("Recall: %0.2f%%"%(100*recall_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print("F1-Score: %0.2f%%"%(100*f1_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
 print()
 print()

 forest_models.append(model)
 f1_forest_list.append(f1_score(test2['Diagnosis'].to_numpy(), Y_test_pred))
```

```
RANDOM FOREST n_estimators = 2

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 91.12%
Precision: 79.17%
Recall: 65.52%
F1-Score: 71.70%
```

```
MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 3

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 88.17%
Precision: 76.47%
Recall: 44.83%
F1-Score: 56.52%

MÉTRICAS EN VALIDACIÓN
Accuracy: 72.09%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 4

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 89.94%
Precision: 87.50%
Recall: 48.28%
F1-Score: 62.22%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 5

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 95.27%
Precision: 95.65%
Recall: 75.86%
F1-Score: 84.62%

MÉTRICAS EN VALIDACIÓN
Accuracy: 76.74%
```

```
Precision: 100.00%
Recall: 9.09%
F1-Score: 16.67%


RANDOM FOREST n_estimators = 6

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 94.67%
Precision: 100.00%
Recall: 68.97%
F1-Score: 81.63%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 7

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 93.49%
Precision: 100.00%
Recall: 62.07%
F1-Score: 76.60%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Precision: 0.00%
```

```
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 8

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 95.27%
Precision: 100.00%
Recall: 72.41%
F1-Score: 84.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 9

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 90.53%
Precision: 100.00%
Recall: 44.83%
F1-Score: 61.90%

MÉTRICAS EN VALIDACIÓN
Accuracy: 69.77%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


RANDOM FOREST n_estimators = 10

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 92.90%
Precision: 100.00%
Recall: 58.62%
F1-Score: 73.91%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%
```

```
RANDOM FOREST n_estimators = 20

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 94.67%
Precision: 100.00%
Recall: 68.97%
F1-Score: 81.63%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

RANDOM FOREST n_estimators = 50

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 97.04%
Precision: 100.00%
Recall: 82.76%
F1-Score: 90.57%

MÉTRICAS EN VALIDACIÓN
Accuracy: 72.09%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%
```

[158]:

```python
f1_forest_list = np.array(f1_forest_list)
```

```
print("n_estimators con mejor desempeño: %d"%n_trees[np.where(np.
 ↪max(f1_forest_list)==f1_forest_list)[0][0]])
print("Mejor F1-Score en validación: %0.2f%%"%(100*f1_forest_list[np.
 ↪where(f1_forest_list==np.max(f1_forest_list))[0][0]]))
```

```
n_estimators con mejor desempeño: 5
Mejor F1-Score en validación: 16.67%
```

3.0.3 Máquina de Soporte Vectorial SVM

Este algoritmo encuentra un modelo o frontera que separa a los dos clases de interés por una recta o hiperplano (en función de la dimensión de las características). Esta frontera se calcula al maximizar la distancia o margen que existe entre los puntos de la frontera de cada clase y dicha frontera. Desde otro punto de vista, minimiza una función de costo que está asi misma está en función del producto punto de un vector de pesos y el vector de características (el vector de pesos es siempre ortogonal a la frontera que se busca), esta función también es conocida como entropía cruzada (cross-entropy). Siendo esta función de costo una ligera modificación de la función de costo utilizada por la Regresión Logística.

En muchos problemas la separabilidad lineal de las clases no es posible, por tanto este algoritmo propone una tranformación (Kernelization) previa de las cracterísticas para ("con suerte") mejorar la separabilidad lineal. Esta transformación puede ser lineal (sumar y multiplicar por escalares), polinomial (multiplicar los valores entre esí, elevar al cuadrado, etc.), gaussiana (convertir los datos a una distribución normal centrada en el origen), entre otras.

[159]:
```
# Importando método auxiliar de Scikit-Learn
from sklearn import svm
```

Se compararán todos los modelos resultantes de cada uno de los kernels propuestos en la biblioteca de Scikit-learn.

Otro parámetro que podría ser ajustado es C, el cual es un parámetro que trabaja sobre el término de regularización de la función de costo. Es el inverso de la tasa de olvido de la Regresión logística. Al incrementarse este parámetro teóricamente aumenta el margen que separa a los puntos de la frontera calculada por el algoritmo (vectores de soporte). Por tanto la "sintonización" es una tarea no trivial.

[160]:
```
# Realizamos for para probar con diferentes kernels en la SVM

# Definimos los kernels
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
svm_models = []
c_list = [0.01, 0.1, 1, 3, 10]

# Aquí almacenaremos los valores F1-Score de los diferentes modelos
f1_matrix_svm = np.zeros((len(kernels),len(c_list)))

for i in range(len(kernels)):
  for j in range(len(c_list)):
```

```
    # Definimos el modelo
    model = svm.SVC(C=c_list[j], kernel=kernels[i])

    # Ajustamos el modelo
    model.fit(X_train2,train2['Diagnosis'].to_numpy())

    print("SVM con C = %s, %s kernel"%(c_list[j],kernels[i]))

    # Calculando las predicciones Y_train_pred y Y_test_pred
    Y_train_pred = model.predict(X_train2)
    Y_test_pred = model.predict(X_test2)

    # Desplegando métrica del rendimiento en el entrenamiento
    print()
    print("MÉTRICAS EN ENTRENAMIENTO")
    print("Accuracy: %0.2f%%"%(100*accuracy_score(train2['Diagnosis'].
↪to_numpy(), Y_train_pred)))
    print("Precision: %0.2f%%"%(100*precision_score(train2['Diagnosis'].
↪to_numpy(), Y_train_pred)))
    print("Recall: %0.2f%%"%(100*recall_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
    print("F1-Score: %0.2f%%"%(100*f1_score(train2['Diagnosis'].to_numpy(),␣
↪Y_train_pred)))
    print()
    print("MÉTRICAS EN VALIDACIÓN")
    print("Accuracy: %0.2f%%"%(100*accuracy_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
    print("Precision: %0.2f%%"%(100*precision_score(test2['Diagnosis'].
↪to_numpy(), Y_test_pred)))
    print("Recall: %0.2f%%"%(100*recall_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
    print("F1-Score: %0.2f%%"%(100*f1_score(test2['Diagnosis'].to_numpy(),␣
↪Y_test_pred)))
    print()
    print()

    svm_models.append(model)
    f1_matrix_svm[i,j] = f1_score(test2['Diagnosis'].to_numpy(), Y_test_pred)
```

```
SVM con C = 0.01, linear kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%
```

```
MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 0.1, linear kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 1, linear kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 3, linear kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
```

```
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 10, linear kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 0.01, poly kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 0.1, poly kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 83.43%
Precision: 100.00%
Recall: 3.45%
F1-Score: 6.67%

MÉTRICAS EN VALIDACIÓN
Accuracy: 67.44%
Precision: 28.57%
Recall: 18.18%
F1-Score: 22.22%
```

```
SVM con C = 1, poly kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 86.98%
Precision: 100.00%
Recall: 24.14%
F1-Score: 38.89%

MÉTRICAS EN VALIDACIÓN
Accuracy: 58.14%
Precision: 18.18%
Recall: 18.18%
F1-Score: 18.18%


SVM con C = 3, poly kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 88.17%
Precision: 100.00%
Recall: 31.03%
F1-Score: 47.37%

MÉTRICAS EN VALIDACIÓN
Accuracy: 60.47%
Precision: 20.00%
Recall: 18.18%
F1-Score: 19.05%


SVM con C = 10, poly kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 90.53%
Precision: 100.00%
Recall: 44.83%
F1-Score: 61.90%

MÉTRICAS EN VALIDACIÓN
Accuracy: 53.49%
Precision: 15.38%
Recall: 18.18%
F1-Score: 16.67%


SVM con C = 0.01, rbf kernel
```

```
MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 0.1, rbf kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 1, rbf kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 3, rbf kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 86.98%
Precision: 100.00%
Recall: 24.14%
F1-Score: 38.89%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 10, rbf kernel

MÉTRICAS EN ENTRENAMIENTO
```

```
Accuracy: 90.53%
Precision: 93.33%
Recall: 48.28%
F1-Score: 63.64%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 0.01, sigmoid kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 0.1, sigmoid kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%


SVM con C = 1, sigmoid kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
```

```
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 65.12%
Precision: 33.33%
Recall: 36.36%
F1-Score: 34.78%


SVM con C = 3, sigmoid kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 79.88%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 53.49%
Precision: 33.33%
Recall: 81.82%
F1-Score: 47.37%


SVM con C = 10, sigmoid kernel

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 75.74%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%

MÉTRICAS EN VALIDACIÓN
Accuracy: 55.81%
Precision: 34.62%
Recall: 81.82%
F1-Score: 48.65%



/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Se observa que los mejores reconocedores en la validación son los kernels que tiene valores de C , sin embargo en el conjunto de entrenamiento no tiene un buen rendimiento al no poder discriminar a ningún positivo. Es decir, el F1-Score es 0%.

El modelo obtenido con un kernel polinomial con C = 3 tiene un F1-score de 47.37% y 19.05% en el entrenamiento y validación respectivamente.

3.0.4 Regresión Logística

La regresión logística es un modelo lineal, tiene una capacidad limitada. También podría interpretarse a la Regresión Logística como una neurona artificial.

[161]:
```python
# Importando el método auxiliar
from sklearn.linear_model import LogisticRegression
```

[162]:
```python
LR = LogisticRegression(C=0.01, solver='liblinear').
 ↪fit(X_train2,train2['Diagnosis'].to_numpy())
```

[163]:
```python
# Calculando las predicciones para LR
Y_train_pred = LR.predict(X_train2)
Y_train_proba = LR.predict_proba(X_train2)[:,1] # Arreglo de probabilidades
Y_test_pred = LR.predict(X_test2)
Y_test_proba = LR.predict_proba(X_test2)[:,1] # Arreglo de probabilidades
```

[164]:
```python
# Desplegando métrica del rendimiento
print("REGRESIÓN LOGÍSTICA")
print()
print("MÉTRICAS EN ENTRENAMIENTO")
print("Accuracy: %0.2f%%"%(100*accuracy_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
print("Precision: %0.2f%%"%(100*precision_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
```

```
print("Recall: %0.2f%%"%(100*recall_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
print("F1-Score: %0.2f%%"%(100*f1_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
fp, tp, _ = roc_curve(train2['Diagnosis'].to_numpy(), Y_train_proba)
print("AUC: %0.2f%%"%(100*auc(fp, tp)))
print()
print("MÉTRICAS EN VALIDACIÓN")
print("Accuracy: %0.2f%%"%(100*accuracy_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
print("Precision: %0.2f%%"%(100*precision_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
print("Recall: %0.2f%%"%(100*recall_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
print("F1-Score: %0.2f%%"%(100*f1_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
fp, tp, _ = roc_curve(test2['Diagnosis'].to_numpy(), Y_test_proba)
print("AUC: %0.2f%%"%(100*auc(fp, tp)))
print()
```

```
REGRESIÓN LOGÍSTICA

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 82.84%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%
AUC: 68.40%

MÉTRICAS EN VALIDACIÓN
Accuracy: 74.42%
Precision: 0.00%
Recall: 0.00%
F1-Score: 0.00%
AUC: 67.90%


/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1272:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

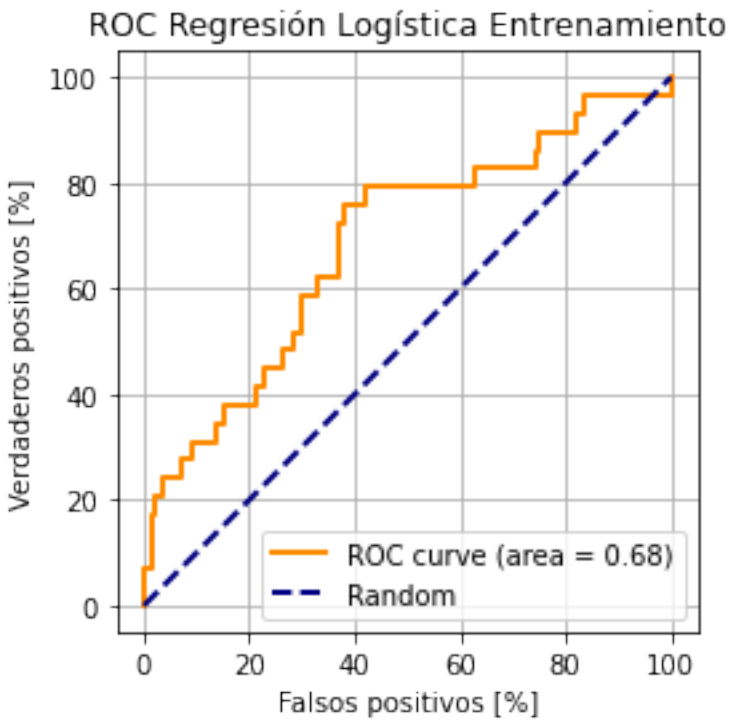
Antes de continuar de definirá una función para graficar la curva ROC que se utilizará también para la evaluación de la Red Neuronal de 3 capas ocultas.

[165]:
```
# Definiendo la función para desplegar la curva ROC
def plot_roc(name, labels, predictions, name_fig):
    fp, tp, _ = roc_curve(labels, predictions)
```

```
    auc1 = auc(fp, tp)
    lw=2

    plt.plot(100*fp, 100*tp, linewidth=2, lw=lw, color='darkorange', label='ROC␣
 ↪curve (area = %0.2f)' % auc1)
    plt.plot([0, 100], [0, 100], color='navy', lw=lw,␣
 ↪linestyle='--',label='Random')
    plt.xlabel('Falsos positivos [%]')
    plt.ylabel('Verdaderos positivos [%]')
    plt.xlim([-5,105])
    plt.ylim([-5,105])
    plt.grid(True)
    ax = plt.gca()
    ax.set_aspect('equal')
    plt.title(name)
    plt.legend(loc="lower right")
    plt.savefig('ROC_'+name_fig+'.png')
```

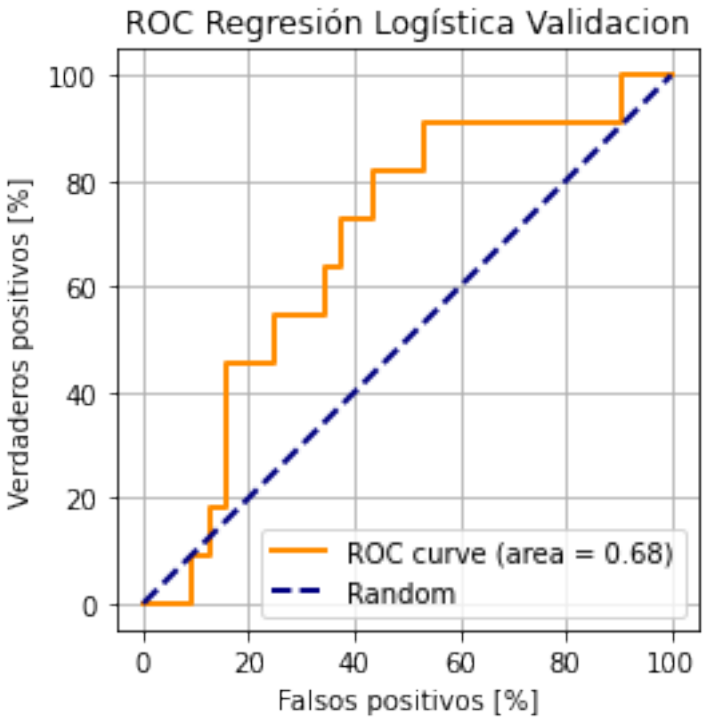
[166]:
```
# Curva ROC de entrenamiento
plot_roc('ROC Regresión Logística Entrenamiento', train2['Diagnosis'].
 ↪to_numpy(), Y_train_proba, name_fig='RegLog')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7:
MatplotlibDeprecationWarning: Saw kwargs ['lw', 'linewidth'] which are all
aliases for 'linewidth'.  Kept value from 'linewidth'.  Passing multiple aliases
for the same property will raise a TypeError in 3.3.
  import sys
```



[167]:
```
# Curva de Roc de Validación
plot_roc('ROC Regresión Logística Validacion', test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_proba, name_fig='RegLog')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7:
MatplotlibDeprecationWarning: Saw kwargs ['lw', 'linewidth'] which are all
aliases for 'linewidth'.  Kept value from 'linewidth'.  Passing multiple aliases
for the same property will raise a TypeError in 3.3.
  import sys
```



3.0.5 Red Neuronal Artificial

Se propone una red neuronal artificial con tres capas ocultas.

[168]:
```
# importando bibliotecas auxiliares
import tensorflow as tf
from tensorflow.keras.applications import *
from tensorflow.keras.layers import Dense
import tensorflow_addons as tfa
from tensorflow.keras import Model, Input
from mega import Mega
```

[169]:
```
Y_train = train2['Diagnosis'].to_numpy()
Y_test = test2['Diagnosis'].to_numpy()
```

[170]:
```
# Arreglando el problema de clases desbalanceadas estableciendo un peso por␣
 ↪clase en el función de costo
w_p=np.sum(Y_train==0)/Y_train.shape[0]
w_n=np.sum(Y_train==1)/Y_train.shape[0]
```

[171]:
```
# Creando diccionario para establecer el peso de cada clase de salida
class_weights = { 0 : w_n , 1 : w_p }
```

[172]:
```
# Creando el modelo de la red neuronal de tres capasocultas con 10,000 unidades␣
 ↪por capa
rnet = tf.keras.Sequential()
rnet.add(Dense(10000, activation="relu", input_shape=(X_train2.shape[1],)))
rnet.add(Dense(10000, activation="relu"))
rnet.add(Dense(10000, activation="relu"))
rnet.add(Dense(1, activation='sigmoid'))
```

[173]:
```
# Visualizando el modelo
rnet.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10000)             120000
_________________________________________________________________
dense_1 (Dense)              (None, 10000)             100010000
_________________________________________________________________
dense_2 (Dense)              (None, 10000)             100010000
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 10001
=================================================================
Total params: 200,150,001
Trainable params: 200,150,001
Non-trainable params: 0
_________________________________________________________________
```

[174]:
```
rnet.input.shape
```

[174]:
```
TensorShape([None, 11])
```

[175]:
```
# Definimos el optimizador utilizando el método de Adam
# Este método es una modificación del algoritmo de
# gradiente descendiente
opt=tf.keras.optimizers.Adam(learning_rate=0.001 ,clipvalue=1)
```

[176]:
```
# Definiendo las métricas necesarias para dar seguimiento al modelo
METRICS = [
  tf.keras.metrics.BinaryAccuracy(name='accuracy'),
  tf.keras.metrics.Precision(name='precision'),
  tf.keras.metrics.Recall(name='recall'),
  tfa.metrics.F1Score(num_classes=2, average="micro", threshold=0.5),
  tf.keras.metrics.AUC(name='AUC')
]
```

[177]:
```
# Compilando el modelo
rnet.compile(
```

```
    optimizer=opt,
    loss = 'binary_crossentropy',
    metrics=METRICS
)
```

[178]:
```
# Con este objeto se guardará el mejor modelo con base a la métrica f1_score
sv = tf.keras.callbacks.ModelCheckpoint(filepath='rnet_clinic_data.
 ↪h5',monitor='val_f1_score', verbose=0, save_best_only=True,
       save_weights_only=False, mode='max', save_freq='epoch')
```

[179]:
```
# Tamaño del batch
# Debido a que es un conjunto de datos relativamente pequeño se utilizará todos␣
 ↪los ejemplos como batch
BS = X_train2.shape[0]
```

[180]:
```
# Entrenando el modelo y almacenando los valores en history_1
history_1 = rnet.fit(X_train2, Y_train, batch_size=BS,
                         steps_per_epoch=len(X_train2) // BS,␣
 ↪validation_data=(X_test2, test2['Diagnosis'].to_numpy()),
                         epochs=100, verbose=2, class_weight=class_weights,␣
 ↪callbacks=[sv])
```

```
Epoch 1/100
1/1 - 19s - loss: 0.1970 - accuracy: 0.4024 - precision: 0.1786 - recall: 0.6897
- f1_score: 0.2837 - AUC: 0.4954 - val_loss: 1.0234 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6009
Epoch 2/100
1/1 - 0s - loss: 0.1895 - accuracy: 0.7278 - precision: 0.3509 - recall: 0.6897
- f1_score: 0.4651 - AUC: 0.7355 - val_loss: 21.8893 - val_accuracy: 0.2558 -
val_precision: 0.2558 - val_recall: 1.0000 - val_f1_score: 0.4074 - val_AUC:
0.5000
Epoch 3/100
1/1 - 0s - loss: 1.7140 - accuracy: 0.1716 - precision: 0.1716 - recall: 1.0000
- f1_score: 0.2929 - AUC: 0.4862 - val_loss: 1.8645 - val_accuracy: 0.2558 -
val_precision: 0.2558 - val_recall: 1.0000 - val_f1_score: 0.4074 - val_AUC:
0.5838
Epoch 4/100
1/1 - 0s - loss: 0.2126 - accuracy: 0.1953 - precision: 0.1758 - recall: 1.0000
- f1_score: 0.2990 - AUC: 0.7394 - val_loss: 1.3616 - val_accuracy: 0.7442 -
val_precision: 0.0000e+00 - val_recall: 0.0000e+00 - val_f1_score: 0.0000e+00 -
val_AUC: 0.5355
Epoch 5/100
1/1 - 0s - loss: 0.2456 - accuracy: 0.8284 - precision: 0.0000e+00 - recall:
0.0000e+00 - f1_score: 0.0000e+00 - AUC: 0.6456 - val_loss: 0.6925 -
val_accuracy: 0.7442 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00 -
val_f1_score: 0.0000e+00 - val_AUC: 0.5483
Epoch 6/100
```

```
1/1 - 0s - loss: 0.1925 - accuracy: 0.8284 - precision: 0.0000e+00 - recall:
0.0000e+00 - f1_score: 0.0000e+00 - AUC: 0.7494 - val_loss: 0.6267 -
val_accuracy: 0.6512 - val_precision: 0.3000 - val_recall: 0.2727 -
val_f1_score: 0.2857 - val_AUC: 0.6548
Epoch 7/100
1/1 - 20s - loss: 0.1853 - accuracy: 0.7160 - precision: 0.3333 - recall: 0.6552
- f1_score: 0.4419 - AUC: 0.7885 - val_loss: 0.6373 - val_accuracy: 0.6279 -
val_precision: 0.3913 - val_recall: 0.8182 - val_f1_score: 0.5294 - val_AUC:
0.6406
Epoch 8/100
1/1 - 0s - loss: 0.1779 - accuracy: 0.6568 - precision: 0.3210 - recall: 0.8966
- f1_score: 0.4727 - AUC: 0.7900 - val_loss: 0.5730 - val_accuracy: 0.6977 -
val_precision: 0.3750 - val_recall: 0.2727 - val_f1_score: 0.3158 - val_AUC:
0.6776
Epoch 9/100
1/1 - 0s - loss: 0.1649 - accuracy: 0.7101 - precision: 0.3438 - recall: 0.7586
- f1_score: 0.4731 - AUC: 0.7910 - val_loss: 0.6261 - val_accuracy: 0.6977 -
val_precision: 0.2500 - val_recall: 0.0909 - val_f1_score: 0.1333 - val_AUC:
0.6491
Epoch 10/100
1/1 - 0s - loss: 0.1532 - accuracy: 0.7278 - precision: 0.3559 - recall: 0.7241
- f1_score: 0.4773 - AUC: 0.8047 - val_loss: 0.7266 - val_accuracy: 0.6744 -
val_precision: 0.4118 - val_recall: 0.6364 - val_f1_score: 0.5000 - val_AUC:
0.6648
Epoch 11/100
1/1 - 20s - loss: 0.1374 - accuracy: 0.6982 - precision: 0.3472 - recall: 0.8621
- f1_score: 0.4950 - AUC: 0.8361 - val_loss: 0.9180 - val_accuracy: 0.6744 -
val_precision: 0.4211 - val_recall: 0.7273 - val_f1_score: 0.5333 - val_AUC:
0.6776
Epoch 12/100
1/1 - 0s - loss: 0.1246 - accuracy: 0.6805 - precision: 0.3418 - recall: 0.9310
- f1_score: 0.5000 - AUC: 0.8595 - val_loss: 1.4172 - val_accuracy: 0.6744 -
val_precision: 0.4000 - val_recall: 0.5455 - val_f1_score: 0.4615 - val_AUC:
0.5696
Epoch 13/100
1/1 - 0s - loss: 0.1107 - accuracy: 0.7278 - precision: 0.3867 - recall: 1.0000
- f1_score: 0.5577 - AUC: 0.8993 - val_loss: 2.4794 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6023
Epoch 14/100
1/1 - 0s - loss: 0.1053 - accuracy: 0.7515 - precision: 0.4030 - recall: 0.9310
- f1_score: 0.5625 - AUC: 0.8885 - val_loss: 2.3490 - val_accuracy: 0.6512 -
val_precision: 0.3571 - val_recall: 0.4545 - val_f1_score: 0.4000 - val_AUC:
0.6108
Epoch 15/100
1/1 - 0s - loss: 0.0939 - accuracy: 0.7515 - precision: 0.4085 - recall: 1.0000
- f1_score: 0.5800 - AUC: 0.9353 - val_loss: 2.6781 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
```

```
0.6037
Epoch 16/100
1/1 - 0s - loss: 0.0856 - accuracy: 0.7811 - precision: 0.4394 - recall: 1.0000
- f1_score: 0.6105 - AUC: 0.9388 - val_loss: 3.5976 - val_accuracy: 0.7209 -
val_precision: 0.4444 - val_recall: 0.3636 - val_f1_score: 0.4000 - val_AUC:
0.5994
Epoch 17/100
1/1 - 0s - loss: 0.0827 - accuracy: 0.8107 - precision: 0.4717 - recall: 0.8621
- f1_score: 0.6098 - AUC: 0.9298 - val_loss: 3.8110 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6193
Epoch 18/100
1/1 - 0s - loss: 0.0712 - accuracy: 0.8284 - precision: 0.5000 - recall: 0.9655
- f1_score: 0.6588 - AUC: 0.9562 - val_loss: 4.2941 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6179
Epoch 19/100
1/1 - 0s - loss: 0.0676 - accuracy: 0.8402 - precision: 0.5179 - recall: 1.0000
- f1_score: 0.6824 - AUC: 0.9610 - val_loss: 5.0223 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.5980
Epoch 20/100
1/1 - 0s - loss: 0.0620 - accuracy: 0.8698 - precision: 0.5686 - recall: 1.0000
- f1_score: 0.7250 - AUC: 0.9633 - val_loss: 5.8166 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6080
Epoch 21/100
1/1 - 0s - loss: 0.0585 - accuracy: 0.8935 - precision: 0.6222 - recall: 0.9655
- f1_score: 0.7568 - AUC: 0.9688 - val_loss: 6.3635 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6193
Epoch 22/100
1/1 - 0s - loss: 0.0537 - accuracy: 0.8817 - precision: 0.5918 - recall: 1.0000
- f1_score: 0.7436 - AUC: 0.9744 - val_loss: 6.8696 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6080
Epoch 23/100
1/1 - 0s - loss: 0.0495 - accuracy: 0.8757 - precision: 0.5800 - recall: 1.0000
- f1_score: 0.7342 - AUC: 0.9800 - val_loss: 7.6194 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6080
Epoch 24/100
1/1 - 0s - loss: 0.0453 - accuracy: 0.9053 - precision: 0.6444 - recall: 1.0000
- f1_score: 0.7838 - AUC: 0.9826 - val_loss: 8.5225 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6136
Epoch 25/100
1/1 - 0s - loss: 0.0436 - accuracy: 0.9408 - precision: 0.7568 - recall: 0.9655
```

```
- f1_score: 0.8485 - AUC: 0.9840 - val_loss: 9.1955 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6151
Epoch 26/100
1/1 - 0s - loss: 0.0401 - accuracy: 0.9172 - precision: 0.6744 - recall: 1.0000
- f1_score: 0.8056 - AUC: 0.9853 - val_loss: 9.8334 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6094
Epoch 27/100
1/1 - 0s - loss: 0.0369 - accuracy: 0.9231 - precision: 0.6905 - recall: 1.0000
- f1_score: 0.8169 - AUC: 0.9889 - val_loss: 10.6670 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6179
Epoch 28/100
1/1 - 0s - loss: 0.0341 - accuracy: 0.9586 - precision: 0.8056 - recall: 1.0000
- f1_score: 0.8923 - AUC: 0.9893 - val_loss: 11.2824 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6264
Epoch 29/100
1/1 - 0s - loss: 0.0304 - accuracy: 0.9408 - precision: 0.7436 - recall: 1.0000
- f1_score: 0.8529 - AUC: 0.9931 - val_loss: 11.9747 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6136
Epoch 30/100
1/1 - 0s - loss: 0.0280 - accuracy: 0.9467 - precision: 0.7632 - recall: 1.0000
- f1_score: 0.8657 - AUC: 0.9941 - val_loss: 12.7175 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6108
Epoch 31/100
1/1 - 0s - loss: 0.0266 - accuracy: 0.9645 - precision: 0.8485 - recall: 0.9655
- f1_score: 0.9032 - AUC: 0.9937 - val_loss: 13.1476 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6023
Epoch 32/100
1/1 - 0s - loss: 0.0240 - accuracy: 0.9527 - precision: 0.7838 - recall: 1.0000
- f1_score: 0.8788 - AUC: 0.9948 - val_loss: 13.9942 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6278
Epoch 33/100
1/1 - 0s - loss: 0.0238 - accuracy: 0.9645 - precision: 0.8485 - recall: 0.9655
- f1_score: 0.9032 - AUC: 0.9943 - val_loss: 14.3561 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6065
Epoch 34/100
1/1 - 0s - loss: 0.0240 - accuracy: 0.9467 - precision: 0.7632 - recall: 1.0000
- f1_score: 0.8657 - AUC: 0.9942 - val_loss: 15.3884 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6307
```

```
Epoch 35/100
1/1 - 0s - loss: 0.0215 - accuracy: 0.9645 - precision: 0.8485 - recall: 0.9655
- f1_score: 0.9032 - AUC: 0.9958 - val_loss: 15.9998 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6392
Epoch 36/100
1/1 - 0s - loss: 0.0206 - accuracy: 0.9704 - precision: 0.8529 - recall: 1.0000
- f1_score: 0.9206 - AUC: 0.9948 - val_loss: 15.7987 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6151
Epoch 37/100
1/1 - 0s - loss: 0.0239 - accuracy: 0.9527 - precision: 0.7838 - recall: 1.0000
- f1_score: 0.8788 - AUC: 0.9968 - val_loss: 16.5662 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6392
Epoch 38/100
1/1 - 0s - loss: 0.0168 - accuracy: 0.9704 - precision: 0.8529 - recall: 1.0000
- f1_score: 0.9206 - AUC: 0.9970 - val_loss: 17.6060 - val_accuracy: 0.7674 -
val_precision: 0.5556 - val_recall: 0.4545 - val_f1_score: 0.5000 - val_AUC:
0.6648
Epoch 39/100
1/1 - 0s - loss: 0.0242 - accuracy: 0.9704 - precision: 0.9000 - recall: 0.9310
- f1_score: 0.9153 - AUC: 0.9963 - val_loss: 16.9739 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 40/100
1/1 - 0s - loss: 0.0245 - accuracy: 0.9527 - precision: 0.7838 - recall: 1.0000
- f1_score: 0.8788 - AUC: 0.9962 - val_loss: 17.2495 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 41/100
1/1 - 0s - loss: 0.0228 - accuracy: 0.9467 - precision: 0.7632 - recall: 1.0000
- f1_score: 0.8657 - AUC: 0.9964 - val_loss: 18.0245 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6463
Epoch 42/100
1/1 - 0s - loss: 0.0238 - accuracy: 0.9645 - precision: 0.8710 - recall: 0.9310
- f1_score: 0.9000 - AUC: 0.9958 - val_loss: 17.4599 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6236
Epoch 43/100
1/1 - 0s - loss: 0.0156 - accuracy: 0.9645 - precision: 0.8286 - recall: 1.0000
- f1_score: 0.9062 - AUC: 0.9980 - val_loss: 17.4272 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6236
Epoch 44/100
1/1 - 0s - loss: 0.0273 - accuracy: 0.9527 - precision: 0.8000 - recall: 0.9655
- f1_score: 0.8750 - AUC: 0.9931 - val_loss: 17.4772 - val_accuracy: 0.6977 -
```

```
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 45/100
1/1 - 0s - loss: 0.0191 - accuracy: 0.9704 - precision: 0.8529 - recall: 1.0000
- f1_score: 0.9206 - AUC: 0.9951 - val_loss: 17.7853 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 46/100
1/1 - 0s - loss: 0.0186 - accuracy: 0.9704 - precision: 0.8750 - recall: 0.9655
- f1_score: 0.9180 - AUC: 0.9968 - val_loss: 17.5783 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 47/100
1/1 - 0s - loss: 0.0178 - accuracy: 0.9704 - precision: 0.8750 - recall: 0.9655
- f1_score: 0.9180 - AUC: 0.9968 - val_loss: 17.5426 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 48/100
1/1 - 0s - loss: 0.0171 - accuracy: 0.9704 - precision: 0.8529 - recall: 1.0000
- f1_score: 0.9206 - AUC: 0.9967 - val_loss: 17.9089 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6392
Epoch 49/100
1/1 - 0s - loss: 0.0141 - accuracy: 0.9704 - precision: 0.8529 - recall: 1.0000
- f1_score: 0.9206 - AUC: 0.9983 - val_loss: 18.7243 - val_accuracy: 0.7674 -
val_precision: 0.5556 - val_recall: 0.4545 - val_f1_score: 0.5000 - val_AUC:
0.6477
Epoch 50/100
1/1 - 0s - loss: 0.0139 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9978 - val_loss: 19.0400 - val_accuracy: 0.7674 -
val_precision: 0.5556 - val_recall: 0.4545 - val_f1_score: 0.5000 - val_AUC:
0.6562
Epoch 51/100
1/1 - 0s - loss: 0.0157 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9967 - val_loss: 18.4379 - val_accuracy: 0.7442 -
val_precision: 0.5000 - val_recall: 0.4545 - val_f1_score: 0.4762 - val_AUC:
0.6392
Epoch 52/100
1/1 - 0s - loss: 0.0128 - accuracy: 0.9704 - precision: 0.8529 - recall: 1.0000
- f1_score: 0.9206 - AUC: 0.9984 - val_loss: 18.0517 - val_accuracy: 0.7209 -
val_precision: 0.4545 - val_recall: 0.4545 - val_f1_score: 0.4545 - val_AUC:
0.6307
Epoch 53/100
1/1 - 0s - loss: 0.0124 - accuracy: 0.9763 - precision: 0.8788 - recall: 1.0000
- f1_score: 0.9355 - AUC: 0.9985 - val_loss: 17.8819 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 54/100
```

```
1/1 - 0s - loss: 0.0115 - accuracy: 0.9763 - precision: 0.8788 - recall: 1.0000
- f1_score: 0.9355 - AUC: 0.9983 - val_loss: 17.8480 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6307
Epoch 55/100
1/1 - 0s - loss: 0.0116 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9985 - val_loss: 17.9029 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6307
Epoch 56/100
1/1 - 0s - loss: 0.0107 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9986 - val_loss: 17.9997 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6236
Epoch 57/100
1/1 - 0s - loss: 0.0105 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9991 - val_loss: 18.1896 - val_accuracy: 0.6744 -
val_precision: 0.3846 - val_recall: 0.4545 - val_f1_score: 0.4167 - val_AUC:
0.6236
Epoch 58/100
1/1 - 0s - loss: 0.0104 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9994 - val_loss: 18.4909 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 59/100
1/1 - 0s - loss: 0.0089 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9995 - val_loss: 18.8378 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 60/100
1/1 - 0s - loss: 0.0090 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9995 - val_loss: 19.1431 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 61/100
1/1 - 0s - loss: 0.0093 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9994 - val_loss: 19.2762 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 62/100
1/1 - 0s - loss: 0.0082 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9995 - val_loss: 19.3505 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 63/100
1/1 - 0s - loss: 0.0082 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9995 - val_loss: 19.4143 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
```

```
0.6307
Epoch 64/100
1/1 - 0s - loss: 0.0083 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9995 - val_loss: 19.4823 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 65/100
1/1 - 0s - loss: 0.0080 - accuracy: 0.9822 - precision: 0.9062 - recall: 1.0000
- f1_score: 0.9508 - AUC: 0.9995 - val_loss: 19.5538 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 66/100
1/1 - 0s - loss: 0.0076 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9995 - val_loss: 19.6339 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 67/100
1/1 - 0s - loss: 0.0074 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9995 - val_loss: 19.7363 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 68/100
1/1 - 0s - loss: 0.0071 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9995 - val_loss: 19.8766 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 69/100
1/1 - 0s - loss: 0.0070 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9995 - val_loss: 20.0591 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 70/100
1/1 - 0s - loss: 0.0068 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9998 - val_loss: 20.2738 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 71/100
1/1 - 0s - loss: 0.0064 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9998 - val_loss: 20.5080 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 72/100
1/1 - 0s - loss: 0.0062 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9998 - val_loss: 20.7547 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 73/100
1/1 - 0s - loss: 0.0062 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
```

```
- f1_score: 0.9667 - AUC: 0.9998 - val_loss: 21.0149 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 74/100
1/1 - 0s - loss: 0.0059 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 0.9998 - val_loss: 21.2813 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 75/100
1/1 - 0s - loss: 0.0057 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 21.5477 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 76/100
1/1 - 0s - loss: 0.0054 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 21.8068 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 77/100
1/1 - 0s - loss: 0.0053 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 22.0738 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6307
Epoch 78/100
1/1 - 0s - loss: 0.0050 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 22.3278 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 79/100
1/1 - 0s - loss: 0.0047 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 22.5584 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 80/100
1/1 - 0s - loss: 0.0045 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 22.7782 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 81/100
1/1 - 0s - loss: 0.0041 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 22.9745 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6236
Epoch 82/100
1/1 - 0s - loss: 0.0039 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 23.1480 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6321
```

```
Epoch 83/100
1/1 - 0s - loss: 0.0036 - accuracy: 0.9941 - precision: 0.9667 - recall: 1.0000
- f1_score: 0.9831 - AUC: 1.0000 - val_loss: 23.3152 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6321
Epoch 84/100
1/1 - 0s - loss: 0.0034 - accuracy: 0.9882 - precision: 0.9355 - recall: 1.0000
- f1_score: 0.9667 - AUC: 1.0000 - val_loss: 23.4588 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6321
Epoch 85/100
1/1 - 0s - loss: 0.0031 - accuracy: 0.9941 - precision: 0.9667 - recall: 1.0000
- f1_score: 0.9831 - AUC: 1.0000 - val_loss: 23.6153 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6250
Epoch 86/100
1/1 - 0s - loss: 0.0028 - accuracy: 0.9941 - precision: 0.9667 - recall: 1.0000
- f1_score: 0.9831 - AUC: 1.0000 - val_loss: 23.7509 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6250
Epoch 87/100
1/1 - 0s - loss: 0.0026 - accuracy: 0.9941 - precision: 0.9667 - recall: 1.0000
- f1_score: 0.9831 - AUC: 1.0000 - val_loss: 23.8905 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6250
Epoch 88/100
1/1 - 0s - loss: 0.0023 - accuracy: 0.9941 - precision: 0.9667 - recall: 1.0000
- f1_score: 0.9831 - AUC: 1.0000 - val_loss: 24.0182 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6250
Epoch 89/100
1/1 - 0s - loss: 0.0022 - accuracy: 0.9941 - precision: 0.9667 - recall: 1.0000
- f1_score: 0.9831 - AUC: 1.0000 - val_loss: 24.1445 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 90/100
1/1 - 0s - loss: 0.0020 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 24.2824 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 91/100
1/1 - 0s - loss: 0.0018 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 24.4226 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 92/100
1/1 - 0s - loss: 0.0016 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 24.5707 - val_accuracy: 0.6977 -
```

```
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 93/100
1/1 - 0s - loss: 0.0015 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 24.7286 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 94/100
1/1 - 0s - loss: 0.0014 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 24.8937 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 95/100
1/1 - 0s - loss: 0.0013 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 25.0731 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 96/100
1/1 - 0s - loss: 0.0011 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 25.2616 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 97/100
1/1 - 0s - loss: 0.0011 - accuracy: 1.0000 - precision: 1.0000 - recall: 1.0000
- f1_score: 1.0000 - AUC: 1.0000 - val_loss: 25.4490 - val_accuracy: 0.6977 -
val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 - val_AUC:
0.6179
Epoch 98/100
1/1 - 0s - loss: 9.6591e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - f1_score: 1.0000 - AUC: 1.0000 - val_loss: 25.6358 - val_accuracy:
0.6977 - val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 -
val_AUC: 0.6179
Epoch 99/100
1/1 - 0s - loss: 8.7960e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - f1_score: 1.0000 - AUC: 1.0000 - val_loss: 25.8290 - val_accuracy:
0.6977 - val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 -
val_AUC: 0.6179
Epoch 100/100
1/1 - 0s - loss: 7.9968e-04 - accuracy: 1.0000 - precision: 1.0000 - recall:
1.0000 - f1_score: 1.0000 - AUC: 1.0000 - val_loss: 26.0263 - val_accuracy:
0.6977 - val_precision: 0.4167 - val_recall: 0.4545 - val_f1_score: 0.4348 -
val_AUC: 0.6179
```

[185]:
```
# Primera iteración con el F1-score de validación más alto
best_iter=np.where(np.max(history_1.history['val_f1_score'])==history_1.
 ↪history['val_f1_score'])[0][0]
print(best_iter)
```

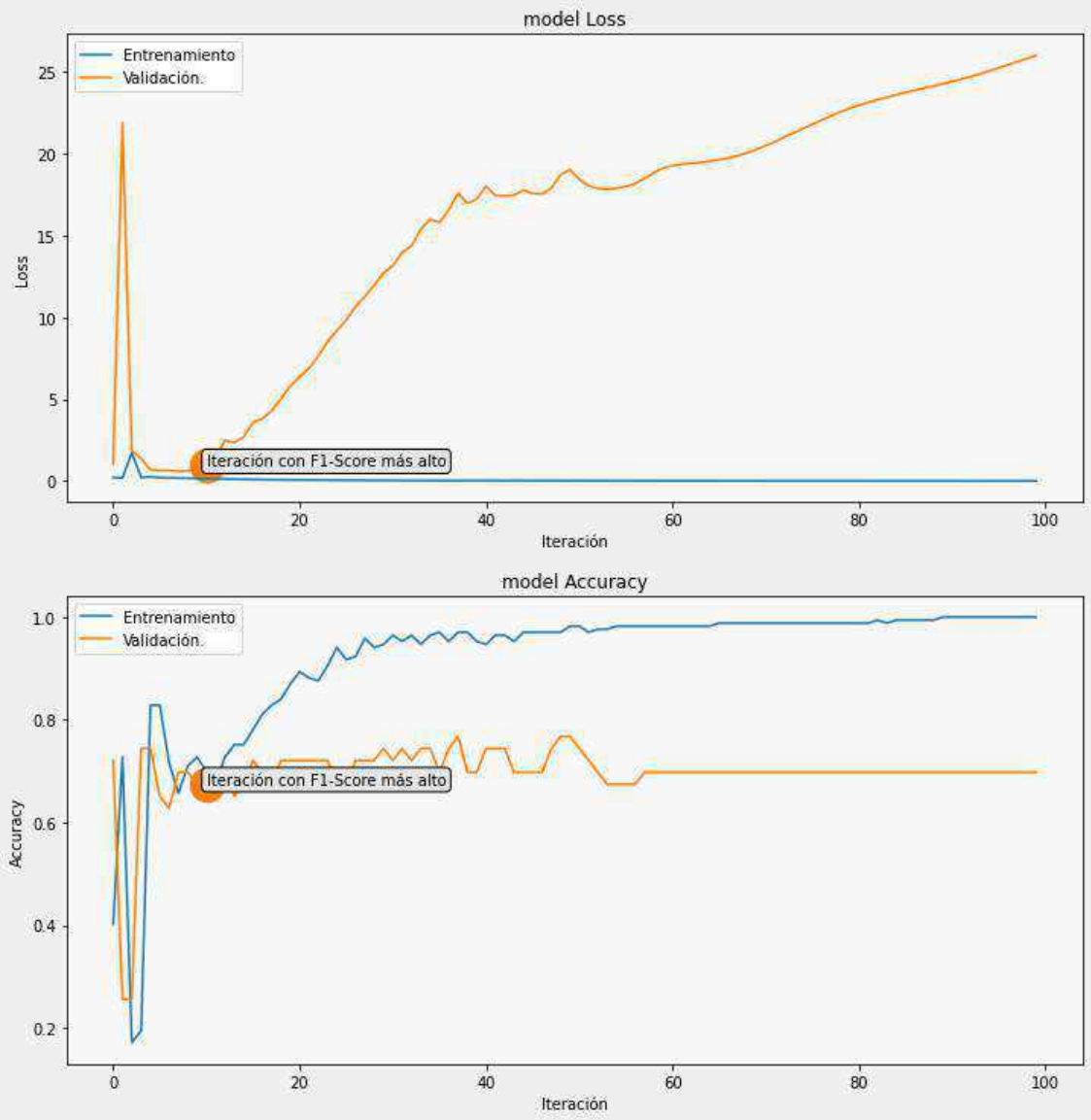
```
10
```

[186]:
```
# F1-Score más alto
history_1.history['val_f1_score'][best_iter]
```

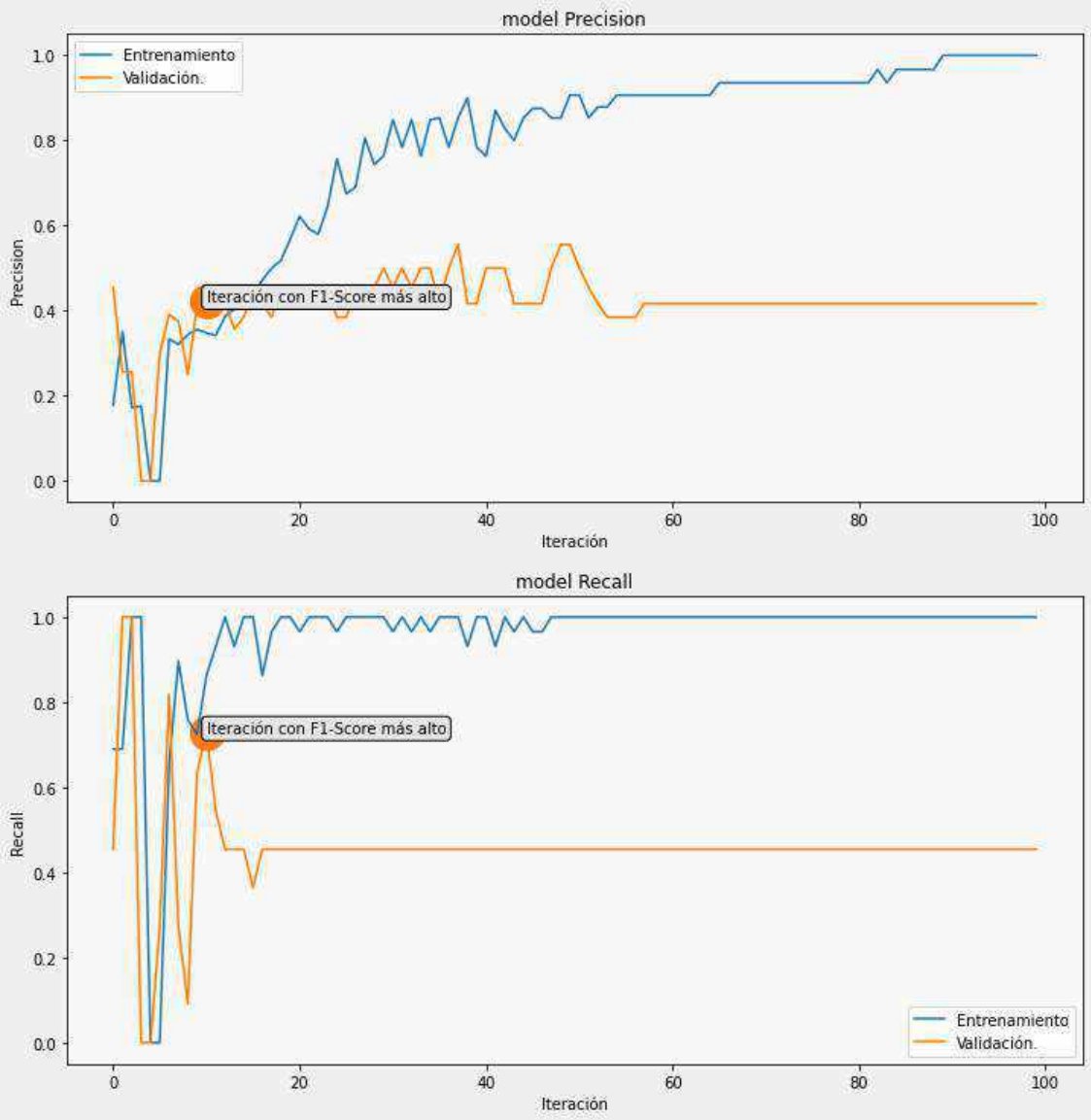
[186]:
```
0.5333333015441895
```

[187]:
```
# Definiendo función para visualizar curvas de entrenamiento
def display_training_curves(training, validation, title, subplot):   ####2
    if subplot%10==1: # set up the subplots on the first call
        plt.subplots(figsize=(10,10), facecolor='#F0F0F0')
        plt.tight_layout()
    ax = plt.subplot(subplot)
    ax.set_facecolor('#F8F8F8')
    ax.plot(training)
    ax.plot(validation)
    ax.set_title('model '+ title)
    ax.set_ylabel(title)
    #ax.set_ylim(0.28,1.05)
    ax.set_xlabel('Iteración')
    ax.legend(['Entrenamiento', 'Validación.'])
    ax.scatter(best_iter, validation[best_iter], c='#ff7f0e', s=500)
    bbox = dict(boxstyle="round", fc="0.9")
    ax.annotate("Iteración con F1-Score más alto", (best_iter,␣
 ↪validation[best_iter]), bbox=bbox)
    plt.savefig(str(title)+'.png')
```

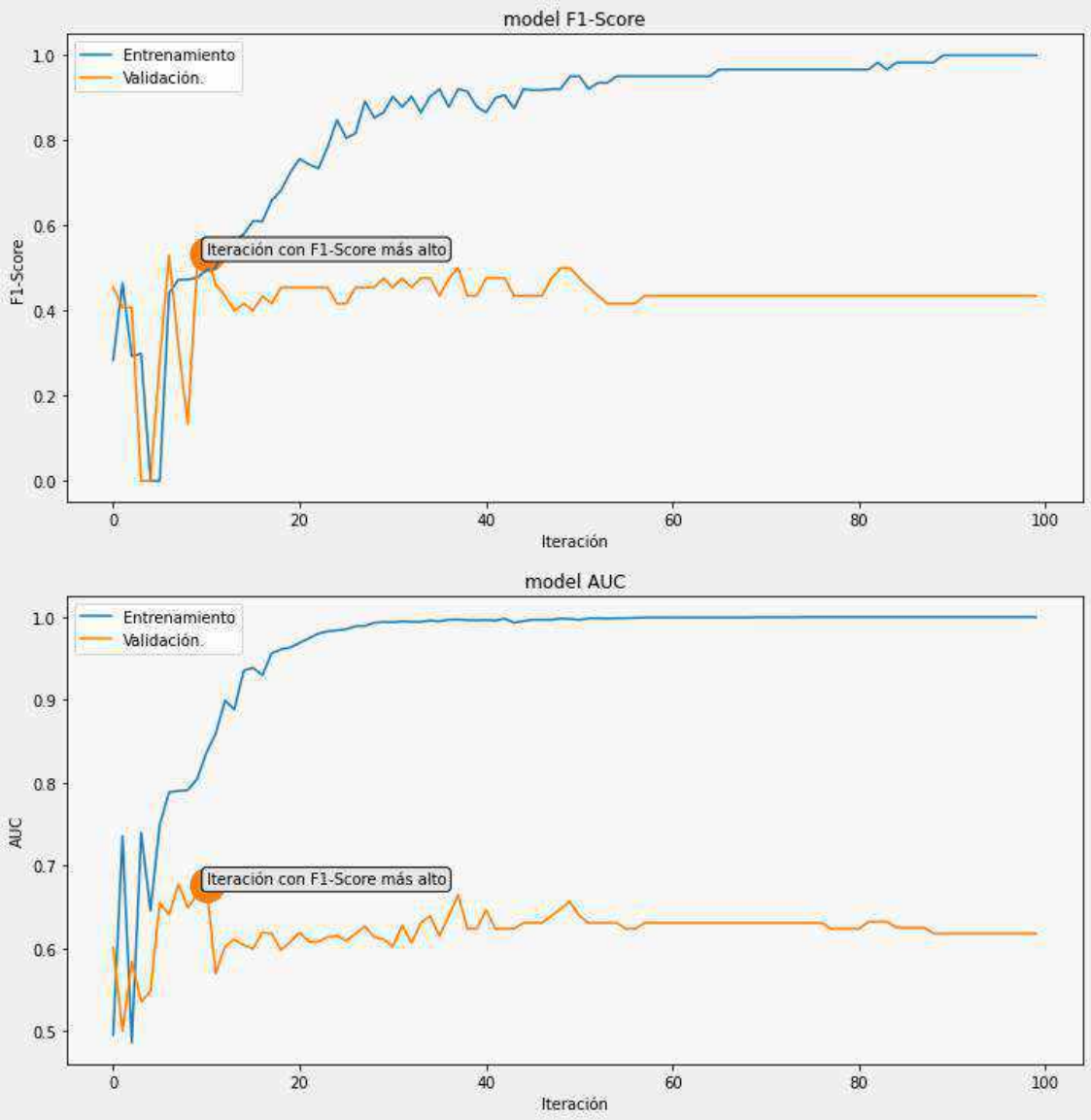
[189]:
```
# Desplegando curvas (loss & accuracy)
display_training_curves(history_1.history['loss'], history_1.
 ↪history['val_loss'], 'Loss', 211)
display_training_curves(history_1.history['accuracy'], history_1.
 ↪history['val_accuracy'], 'Accuracy', 212)
```



[190]:
```
# Desplegando curvas (precision & recall)
display_training_curves(history_1.history['precision'], history_1.
 ↪history['val_precision'], 'Precision', 211)
display_training_curves(history_1.history['recall'], history_1.
 ↪history['val_recall'], 'Recall', 212)
```



[191]:
```
# Desplegando curvas (f1_score & AUC)
display_training_curves(history_1.history['f1_score'], history_1.
 ↪history['val_f1_score'], 'F1-Score', 211)
display_training_curves(history_1.history['AUC'], history_1.history['val_AUC'],␣
 ↪'AUC', 212)
```



[193]:
```
# Cargamos el modelo con la mejor métrica de F1-Score en validación
rnet.load_weights("rnet_clinic_data.h5")
```

[194]:
```
# Calculando las predicciones para Red Neuronal
Y_train_pred = (rnet.predict(X_train2) > 0.5).astype("int32")
Y_train_proba = rnet.predict(X_train2) # Arreglo de probabilidades
Y_test_pred = (rnet.predict(X_test2) > 0.5).astype("int32")
Y_test_proba = rnet.predict(X_test2) # Arreglo de probabilidades
```

[195]:
```
# Desplegando métrica del rendimiento
print("RED NEURONAL")
print()
```

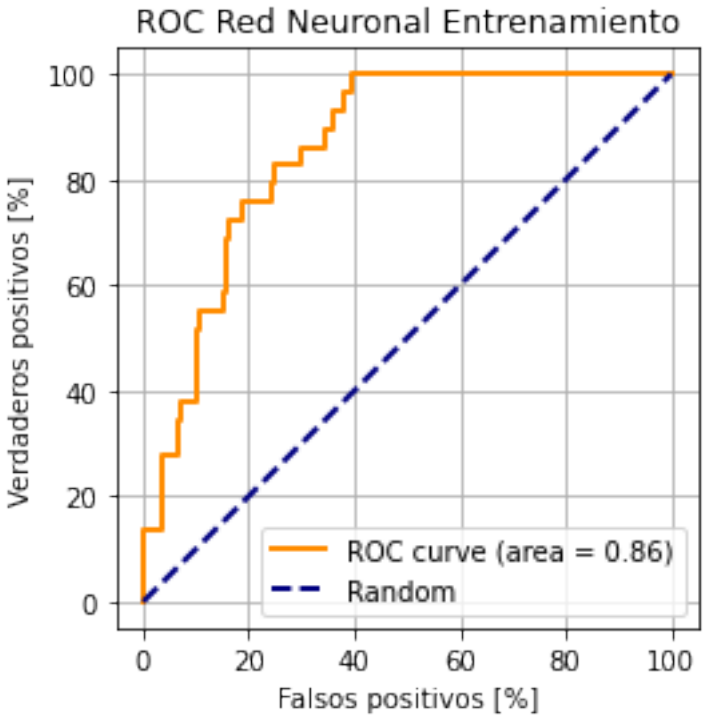
```
print("MÉTRICAS EN ENTRENAMIENTO")
print("Accuracy: %0.2f%%"%(100*accuracy_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
print("Precision: %0.2f%%"%(100*precision_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
print("Recall: %0.2f%%"%(100*recall_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
print("F1-Score: %0.2f%%"%(100*f1_score(train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_pred)))
fp, tp, _ = roc_curve(train2['Diagnosis'].to_numpy(), Y_train_proba)
print("AUC: %0.2f%%"%(100*auc(fp, tp)))
print()
print("MÉTRICAS EN VALIDACIÓN")
print("Accuracy: %0.2f%%"%(100*accuracy_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
print("Precision: %0.2f%%"%(100*precision_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
print("Recall: %0.2f%%"%(100*recall_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
print("F1-Score: %0.2f%%"%(100*f1_score(test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_pred)))
fp, tp, _ = roc_curve(test2['Diagnosis'].to_numpy(), Y_test_proba)
print("AUC: %0.2f%%"%(100*auc(fp, tp)))
print()
```

```
RED NEURONAL

MÉTRICAS EN ENTRENAMIENTO
Accuracy: 68.05%
Precision: 34.18%
Recall: 93.10%
F1-Score: 50.00%
AUC: 85.91%

MÉTRICAS EN VALIDACIÓN
Accuracy: 67.44%
Precision: 42.11%
Recall: 72.73%
F1-Score: 53.33%
AUC: 65.34%
```

[196]:
```
# Curva ROC de entrenamiento
plot_roc('ROC Red Neuronal Entrenamiento', train2['Diagnosis'].to_numpy(),␣
 ↪Y_train_proba, name_fig='RNET')
```

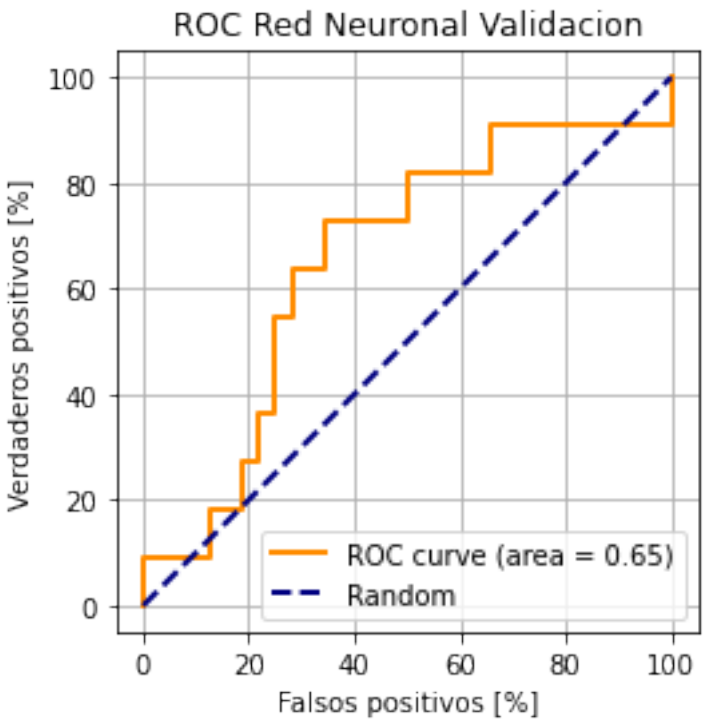
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7:
MatplotlibDeprecationWarning: Saw kwargs ['lw', 'linewidth'] which are all
```

```
aliases for 'linewidth'.  Kept value from 'linewidth'.  Passing multiple aliases
for the same property will raise a TypeError in 3.3.
  import sys
```



[197]:
```
# Curva de Roc de Validación
plot_roc('ROC Red Neuronal Validacion', test2['Diagnosis'].to_numpy(),␣
 ↪Y_test_proba, name_fig='RNET')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:7:
MatplotlibDeprecationWarning: Saw kwargs ['lw', 'linewidth'] which are all
aliases for 'linewidth'.  Kept value from 'linewidth'.  Passing multiple aliases
for the same property will raise a TypeError in 3.3.
  import sys
```



4 Conclusión

En esta libreta se exploraron los datos clínicos de los conjuntos de entrenamiento y validación.

Se propusieron diferentes modelos con el objetivo de encontrar alguno que pudiese discriminar correctamente a las dos clases (corractamente de forma razonable en función de algún umbral arbitrario de la métrica F1-Score). Lamentablemente ninguno de los reconocedores mostró resultados alentadores.

En el siguiente Anexo se utilizarán estos datos del historial pero se utilizará una técnica llamada Análisis de Compnenentes Principales.

[ ]:
```
# Guardamos 4 archivos finales para facilitar el análisis del Anexo siguiente.

np.save("X_train2.npy",X_train2.npy)
np.save("X_test2.npy",X_test2.npy)
np.save("Y_train2.npy",Y_train.npy)
np.save("Y_test2.npy",Y_test.npy)
```