"Estudio de modelos de Machine Learning para la predicción de la obesidad: Un Análisis Comparativo"

Autor: Carlos Matías Sáez

f72masac@gmail.com

Máster Big Data, Data Science & Inteligencia Artificial

Curso 2023-2024

Universidad Complutense de Madrid

Índice

1. Introducción y objetivo

El presente estudio es la tarea del módulo de Machine Learning del máster *"Máster Big Data, Data Science & Inteligencia Artificial"* de la UCM. Este estudio analiza una base de datos que se presentará en el siguiente apartado con el objetivo de predecir la obesidad de los individuos.

Para lograr este objetivo se han utilizado distintos métodos de Machine Learning usando Python y Jupyter Notebook.

El procedimiento seguido a lo largo de este estudio para mostrar los resultados se ha hecho de forma que se ha explicado el código utilizado y a continuación su resultado. En este documento se muestran los resultados obtenidos, pero la forma óptima y cómoda de ver los resultados es a través del notebook de Jupyter. *

Cada método tendrá sus ventajas y desventajas y podrá aplicarse dependiendo de un caso u otro, pero se ha priorizado mostrar cómo se pueden implementar estos modelos y cómo ver su validez antes que los resultados, ya que como veremos en muchos de estos métodos será necesaria por ejemplo una base de datos mayor para obtener resultados donde no haya sobreajuste.

---

*El formato del código y de los resultados vendrán dados de la siguiente forma para intentar imitar el notebook de Jupyter.

- Se presenta la siguiente instrucción:

Código de Python vendrá dado por el siguiente formato:

```
# Este es el código de Python
print("Este es el resultado")
```

El resultado vendrá dado por una captura:

```
Este es el resultado
```

Comentarios

2. Introducción a la base de datos

Los datos consisten en la estimación de los niveles de obesidad en personas de los países de México, Perú y Colombia, con edades entre 14 y 61 años y diversos hábitos alimenticios y condiciones físicas. Los datos fueron recolectados utilizando una plataforma web con una encuesta donde usuarios anónimos respondieron cada pregunta, luego la información fue procesada obteniendo 17 atributos y 2111 registros.

Las variables o parámetros del dataset se presentan a continuación.

Los atributos relacionados con los hábitos alimenticios son:

- Consumo frecuente de alimentos altos en calorías (FAVC)
- Frecuencia de consumo de vegetales (FCVC)
- Número de comidas principales (NCP)
- Consumo de alimentos entre comidas (CAEC)
- Consumo diario de agua (CH20)
- Consumo de alcohol (CALC)
- Familia con historial de sobrepeso (family_history_with_overweight)

Los atributos relacionados con la condición física son:

- Monitoreo del consumo de calorías (SCC)
- Frecuencia de actividad física (FAF)
- Tiempo utilizando dispositivos tecnológicos (TUE)
- Medio de transporte utilizado (MTRANS)

Variables anatómicas del individuo:

- Género (Gender)
- Edad (Age)
- Altura (Height)
- Peso (Weight)
- Clasificación según IMC (NObeyesdad)

3. Depuración de datos

Los datos deben ser depurados para poder trabajar con ellos y usarlos con los métodos de machine learning. La depuración de datos es un proceso fundamental en el análisis de datos que implica identificar y corregir errores, inconsistencias o valores atípicos en una base de datos. Este proceso asegura que los datos sean precisos, coherentes y confiables para su posterior análisis.

Los pasos que se han seguido para la depuración de datos son los siguientes.

- Se han cargado las librerías necesarias para todo el estudio:

```
# Instalamos las librerías necesarias

import os #chdir() permite cambiar el directorio actual
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import sklearn
import tensorflow as tf
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score, precision_score, recall_score, f1_score, roc_auc_score,
roc_curve, auc
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_selection import chi2, SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.ensemble import StackingClassifier
```

- Se selecciona una semilla:

```
seed = 3314
```

La "seed" (semilla) se refiere a un número que se utiliza para inicializar un generador de números aleatorios. Esta semilla determina el punto de inicio para la secuencia de números pseudoaleatorios generados por el algoritmo y nos asegura que el proceso de entrenamiento y evaluación produzca los mismos resultados cada vez que se ejecute el código.

- Se cargan los datos en un dataframe "df" y se toma una muestra aleatoria de 1000 registros:

```
df = pd.read_csv(r"C:\Users\charly\Desktop\MASTER\ML
\datos_practica_miss.csv")

# Tomamos una muestra aleatoria de 1000 registros
df = df.sample(n=1000, random_state=seed)
```

- Se hace una exploración inicial del dataset para conocer las columnas de este:

```
df.head()
```

	Unnamed: 0	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP
519	519	Female	18.535075	1.688025	45.000000	no	yes	3.000000	3.000000
826	826	Female	40.000000	1.561109	62.871794	yes	yes	2.948248	3.000000
838	838	Male	26.000000	1.745033	80.000000	yes	yes	2.217267	1.193589
1718	1718	Male	25.879411	1.765464	114.144378	yes	yes	1.626369	3.000000
856	856	Female	NaN	1.700000	74.244004	NaN	yes	2.000000	NaN

CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
Sometimes	no	3.000000	yes	2.539762	1.283673	no	Public_Transportation	Insufficient_Weight
Sometimes	no	2.429911	no	0.119640	0.360193	Sometimes	Automobile	Overweight_Level_I
Sometimes	no	2.000000	no	2.000000	0.000000	Sometimes	Public_Transportation	Overweight_Level_I
Sometimes	no	2.109697	no	1.352973	0.076693	Sometimes	Public_Transportation	Obesity_Type_II
Sometimes	no	NaN	no	1.722053	0.000000	NaN	Automobile	Overweight_Level_I

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, 519 to 179
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Unnamed: 0                      1000 non-null   int64
 1   Gender                          976 non-null    object
 2   Age                             978 non-null    float64
 3   Height                          980 non-null    float64
 4   Weight                          982 non-null    float64
 5   family_history_with_overweight  978 non-null    object
 6   FAVC                            977 non-null    object
 7   FCVC                            976 non-null    float64
 8   NCP                             973 non-null    float64
 9   CAEC                            976 non-null    object
 10  SMOKE                           988 non-null    object
 11  CH2O                            982 non-null    float64
 12  SCC                             985 non-null    object
 13  FAF                             975 non-null    float64
 14  TUE                             980 non-null    float64
 15  CALC                            979 non-null    object
 16  MTRANS                          978 non-null    object
 17  NObeyesdad                      975 non-null    object
dtypes: float64(8), int64(1), object(9)
memory usage: 148.4+ KB
```

Se puede comprobar las columnas del dataset en "Column" y su correspondiente tipo en "Dtype".

- Se comprueba si hay valores duplicados:

```
# Comprobamos si hay valores duplicados
df.duplicated().sum()
```

```
0
```

En este caso no hay datos duplicados. Los datos duplicados pueden causar problemas en el análisis de datos y pueden llevar a conclusiones erróneas si no se abordan adecuadamente. En el caso de que hubiese se habrían eliminado así:

```
df.drop_duplicates(inplace=True)
```

- Se comprueba si hay valores nulos (missing values):

```
df.isnull().sum()
```

```
Gender                            24
Age                               22
Height                            20
Weight                            18
family_history_with_overweight    22
FAVC                              23
FCVC                              24
NCP                               27
CAEC                              24
SMOKE                             12
CH2O                              18
SCC                               15
FAF                               25
TUE                               20
CALC                              21
MTRANS                            22
NObeyesdad                        25
dtype: int64
```

Por tanto, para cada columna tenemos esos valores nulos. Cuando un conjunto de datos contiene valores nulos es importante abordarlos de manera adecuada, ya que pueden afectar la calidad de los análisis y los modelos de machine learning. Como indica la teoría, se podrán eliminar si los valores nulos de cada columna representan menos del 5% de los datos. Existen distintas formas de tratar los missing en función del caso, como puede ser imputarlos con la media o la moda o utilizar técnicas de regresión.

- Se comprueba el porcentaje de valores nulos en cada columna:

```
df.isnull().mean() * 100
```

```
Gender                            2.414487
Age                               2.213280
Height                            2.012072
Weight                            1.810865
family_history_with_overweight    2.213280
FAVC                              2.313883
FCVC                              2.414487
NCP                               2.716298
CAEC                              2.414487
SMOKE                             1.207243
CH2O                              1.810865
SCC                               1.509054
FAF                               2.515091
TUE                               2.012072
CALC                              2.112676
MTRANS                            2.213280
NObeyesdad                        2.515091
dtype: float64
```

Por tanto, en todas las columnas el porcentaje de valores nulos es menor del 5%.

- Entonces se pueden eliminar y comprobar que ya no hay:

```
df.dropna(inplace=True)
df.isnull().sum()
```

```
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

4. Exploración de datos (Exploratory Data Analysis)

- En primer lugar, se procede a clasificar las columnas en categóricas y numéricas:

```
to_cat = df.loc[:, df.nunique() < 8].columns
df[to_cat] = df[to_cat].astype("category").copy()
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 927 entries, 519 to 1768
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          927 non-null    category
 1   Age                             927 non-null    float64
 2   Height                          927 non-null    float64
 3   Weight                          927 non-null    float64
 4   family_history_with_overweight  927 non-null    category
 5   FAVC                            927 non-null    category
 6   FCVC                            927 non-null    float64
 7   NCP                             927 non-null    float64
 8   CAEC                            927 non-null    category
 9   SMOKE                           927 non-null    category
 10  CH2O                            927 non-null    float64
 11  SCC                             927 non-null    category
 12  FAF                             927 non-null    float64
 13  TUE                             927 non-null    float64
 14  CALC                            927 non-null    category
 15  MTRANS                          927 non-null    category
 16  NObeyesdad                      927 non-null    category
dtypes: category(9), float64(8)
memory usage: 74.8 KB
```

Se comprueba como se ha clasificado en categóricas las columnas con dtype *"category"* y las numéricas con *"float64"*. Además, se comprueba que tras eliminar los missings, nuestra muestra ha pasado de tener 1000 instancias a 927.

- Se crean listas con las columnas categóricas y numéricas para su posterior uso:

```
cat_cols = df.select_dtypes(include='category').columns.to_list()
cat_cols.remove('NObeyesdad')
num_cols = df.select_dtypes(include='number').columns.to_list()
```

- Se hace un analisis de las variables numéricas:

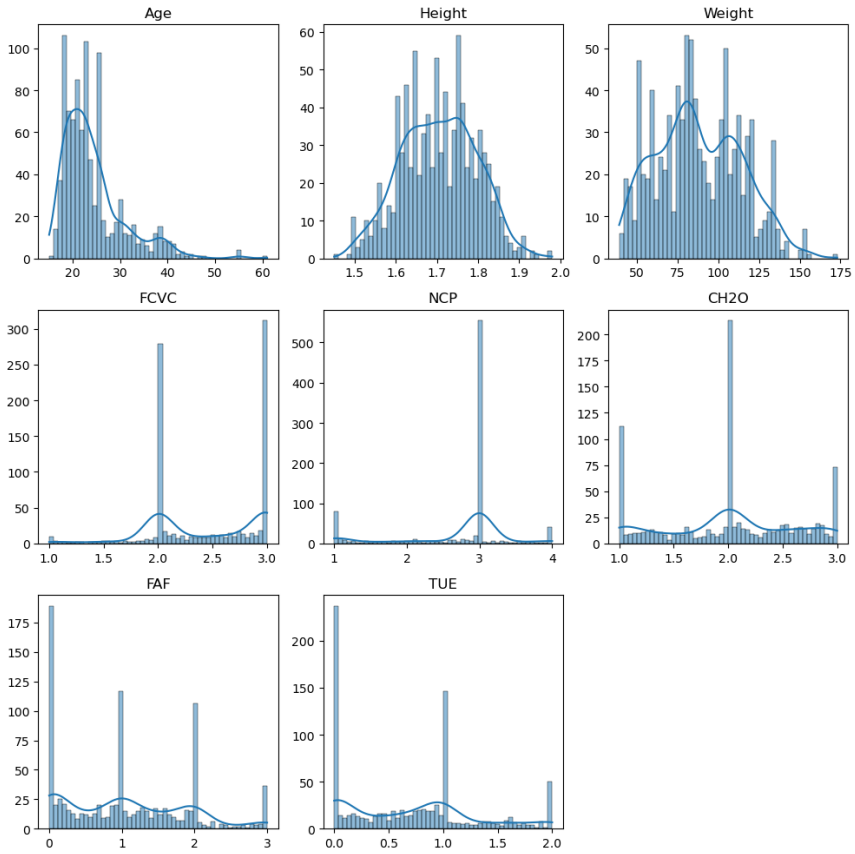
```
df.describe().round(2).style.format(precision=2).background_gradient(
    vmax=75, cmap="Blues"
)
```

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE
count	927.00	927.00	927.00	927.00	927.00	927.00	927.00	927.00
mean	24.20	1.70	87.84	2.44	2.71	2.00	1.04	0.69
std	6.48	0.09	26.11	0.51	0.75	0.62	0.85	0.60
min	15.00	1.45	39.10	1.00	1.00	1.00	0.00	0.00
25%	19.67	1.64	68.00	2.00	2.73	1.53	0.18	0.03
50%	22.84	1.70	85.00	2.46	3.00	2.00	1.00	0.72
75%	26.00	1.77	107.35	3.00	3.00	2.50	1.72	1.00
max	61.00	1.98	173.00	3.00	4.00	3.00	3.00	2.00

Se observan distintos valores para las columnas numéricas tales como la media, los percentiles o los máximos y mínimos de cada una.

- Para entender mejor como se distribuyen las variables numéricas se procede a visualizar su distribución:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    sns.histplot(df[col], bins=50, kde=True)
    plt.title(col)
    plt.xlabel('')
    plt.ylabel('')
plt.tight_layout()
plt.show()
```

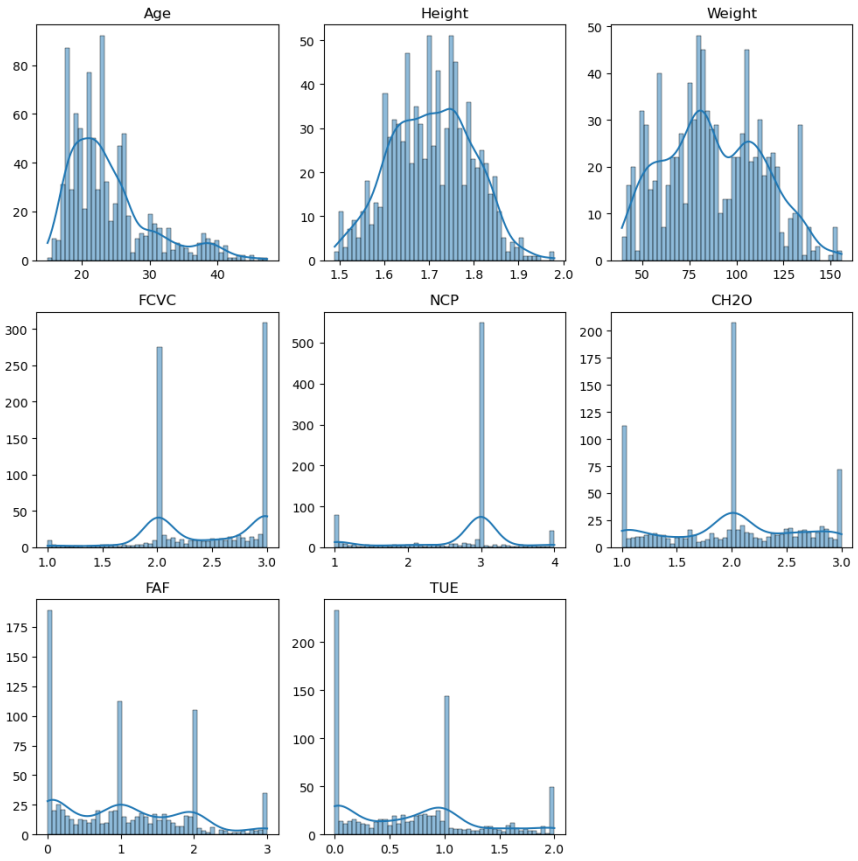


Podemos apreciar como aparecen valores atípicos en la columna "Age", "Weight" y "Height". Los valores atípicos o "*outliers*" son observaciones en un conjunto de datos que se desvían significativamente de los demás puntos de datos. Estos valores pueden surgir debido a errores en la recopilación de datos, medidas extremas o eventos raros y pueden producir errores a la hora de trabajar con los datos.

- Por tanto, procedemos a eliminarlos y a representar de nuevo las distribuciones sin valores atípicos:

```
# Eliminamos aquellos valores para los que la edad sea superior a 50
df = df[df["Age"] <= 50]

# Eliminamos aquellos valores para los que el peso sea superior a 165
df = df[df["Weight"] <= 165]

# Eliminamos aquellos valores para los que la altura sea inferior a 1.46
df = df[df["Height"] >= 1.46]
```

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(num_cols):
    plt.subplot(3, 3, i+1)
    sns.histplot(df[col], bins=50, kde=True)
    plt.title(col)
    plt.xlabel('')
    plt.ylabel('')
plt.tight_layout()
plt.show()
```



- Se describen las variables numéricas después de eliminar los outliers:

```
df[num_cols].describe().round(2).style.format(precision=2).background_gradient(
    vmax=75, cmap="Blues"
)
```

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE
count	920.00	920.00	920.00	920.00	920.00	920.00	920.00	920.00
mean	24.04	1.70	87.84	2.44	2.71	2.00	1.03	0.70
std	6.05	0.09	26.02	0.52	0.75	0.62	0.85	0.60
min	15.00	1.49	39.10	1.00	1.00	1.00	0.00	0.00
25%	19.67	1.64	68.00	2.00	2.72	1.53	0.17	0.04
50%	22.82	1.70	85.00	2.46	3.00	2.00	1.00	0.72
75%	26.00	1.77	107.61	3.00	3.00	2.50	1.72	1.00
max	47.28	1.98	155.87	3.00	4.00	3.00	3.00	2.00

- Se procede a estudiar las columnas categóricas:

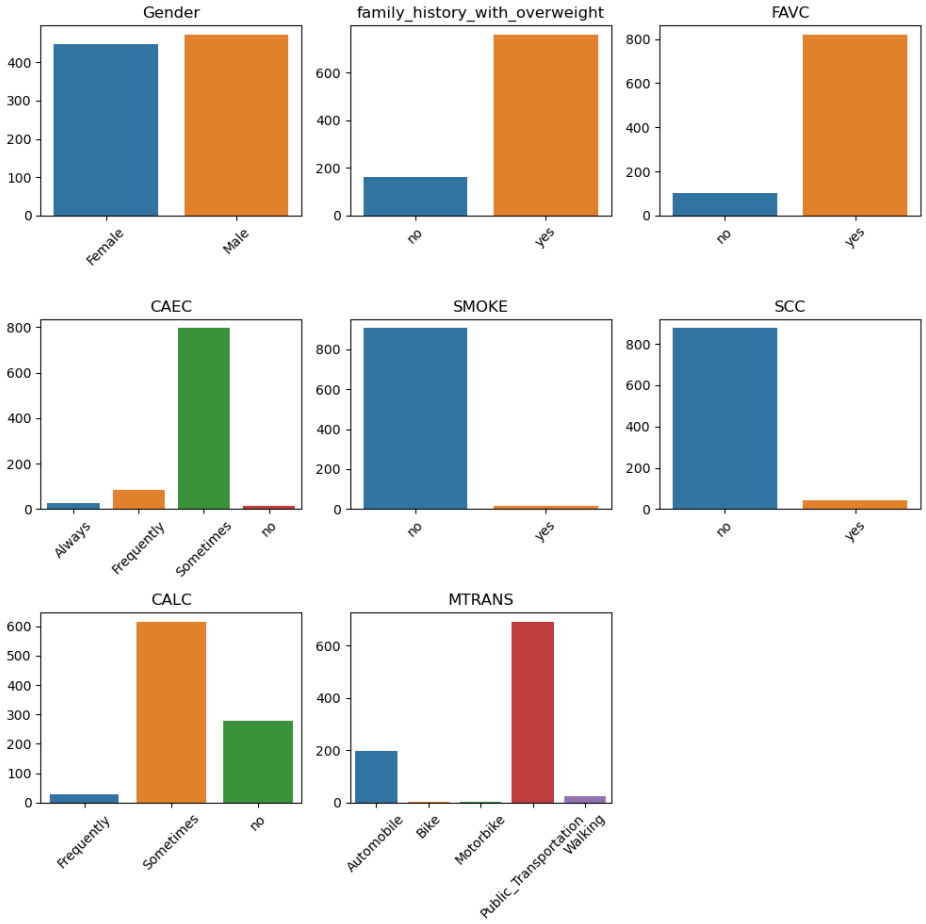
```
df[cat_cols].describe(include="category")
```

	Gender	family_history_with_overweight	FAVC	CAEC	SMOKE	SCC	CALC	MTRANS
count	920	920	920	920	920	920	920	920
unique	2	2	2	4	2	2	3	5
top	Male	yes	yes	Sometimes	no	no	Sometimes	Public_Transportation
freq	472	759	820	796	905	876	615	691

Podemos ver cómo tras eliminar los "outliers" hemos pasado de tener una muestra de 927 a 920. Además, podemos ver cuantas categorías tiene cada columna (unique), cual es la que más se repite (top) y con qué frecuencia (freq).

- Visualizamos la distribución de las variables categóricas:

```
plt.figure(figsize=(10, 10))
for i, col in enumerate(cat_cols):
    plt.subplot(3, 3, i+1)
    sns.countplot(data=df, x=col)
    plt.title(col)
    plt.xticks(rotation=45)
    plt.xlabel('')
    plt.ylabel('')
plt.tight_layout()
plt.show()
```



Detalles que apreciar, por ejemplo, es como se distribuye homogéneamente la variable del género "Gender", indicando que nuestra muestra es representativa tanto para el género femenino como el masculino. Otra apreciación es como prácticamente nadie fuma en "Smoke" o como la mayoría de la muestra tiene una familia con algún historial de sobrepeso en "family_history_with_overweight".

5. Transformación de variables (Feature Engineering)

La transformación de variables, o en ingles Feature Engineering (FE), es el proceso de transformar las columnas. Esto es necesario para mejorar el rendimiento de los modelos de machine learning ya que las características adecuadas pueden hacer que los modelos sean más precisos y generales, mientras que las características irrelevantes o incorrectas pueden afectar negativamente el rendimiento del modelo.

- Hacemos una copia del dataframe original para no modificarlo donde tendremos realizada la feature engineering

```
df2 = df.copy()
```

- Como el objetivo del estudio es crear modelos que sean capaces de predecir si un individuo padece obesidad o no, creamos una nueva columna "Obesity" que lo indique. Esto se ha hecho seleccionando aquellas categorías de "NObeyesdad" que recogen que una persona padece obesidad tales como "Obesity_Type_I", "Obesity_Type_II" y "Obesity_Type_III".

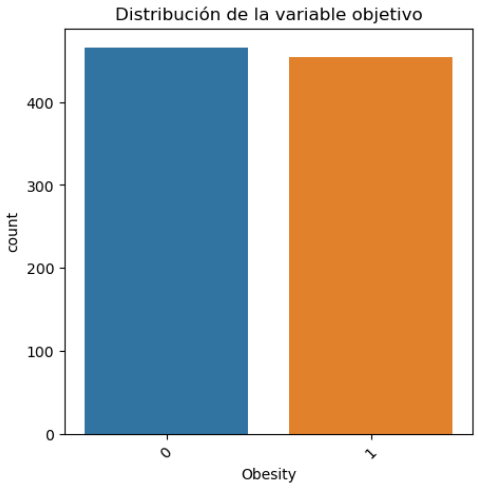
```
df2['Obesity'] = df2['NObeyesdad'].apply(lambda x: 1 if x in
['Obesity_Type_I', 'Obesity_Type_II', 'Obesity_Type_III'] else 0)
```

- Se selecciona la variable objetivo "target", que en nuestro caso será "Obesity":

```
target = 'Obesity'
```

- Comprobamos como se distribuye la variable objetivo:

```
plt.figure(figsize=(5, 5))
sns.countplot(data=df2, x=target)
plt.title('Distribución de la variable objetivo')
plt.xticks(rotation=45)
plt.show()
```



Por tanto, se puede apreciar como nuestra variable objetivo se encuentra bien balanceada. Esto significa que hay de forma equilibrada el mismo número de individuos que padecen obesidad y que no la padecen. Que la variable objetivo esté bien balanceada es importante ya que un desequilibrio en esta puede llevar a un sesgo en el modelo de predicción pudiendo ignorar la clase minoritaria resultando un modelo que tiene un rendimiento deficiente para predecir la clase minoritaria. Si no lo estuviese se podrían usar distintos métodos para equilibrarla, como por ejemplo el undersampling.

- Se transforman las columnas numéricas usando el método "minmaxscaler". Se calcula usando la siguiente formula:

$$X = (x - xmin)/(xmax - xmin)$$

```
scaler = MinMaxScaler()
df2[num_cols] = scaler.fit_transform(df2[num_cols])
```

Esto hace que los valores de las columnas numéricas pasen a estar en el intervalo 0 y 1. Esta transformación es una técnica útil para mejorar el rendimiento computacional de los modelos y permite que posteriormente podamos usar el método "select k best".

- De distinta forma, transformamos las variables categóricas con el método "OneHotEncoder":

```
encoder = OneHotEncoder(drop='first', sparse=False)
encoder.fit(df2[cat_cols])
encoded_cols = encoder.get_feature_names_out(cat_cols)
df2[encoded_cols] = encoder.transform(df2[cat_cols])
df2.drop(columns=cat_cols, inplace=True)
```

De forma que ahora cada categoría de una columna categórica será una columna independiente con valor 0 o 1 en función de si en la columna original el valor era esa categoría o no para cada instancia.

- Observamos como ha quedado nuestro dataset después de las transformaciones:

```
df2.head()
```

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	NObeyesdad	Obesity
519	0.109501	0.404850	0.050511	1.000000	0.666667	1.000000	0.846587	0.641837	Insufficient_Weight	0
826	0.774392	0.146150	0.203562	0.974124	0.666667	0.714956	0.039880	0.180096	Overweight_Level_I	0
838	0.340733	0.521053	0.350245	0.608634	0.064530	0.500000	0.666667	0.000000	Overweight_Level_I	0
1718	0.336997	0.562699	0.642651	0.313184	0.666667	0.554849	0.450991	0.038346	Obesity_Type_II	1
1055	0.557563	0.735013	0.503712	0.500000	0.666667	0.058732	0.333333	0.331825	Overweight_Level_II	0

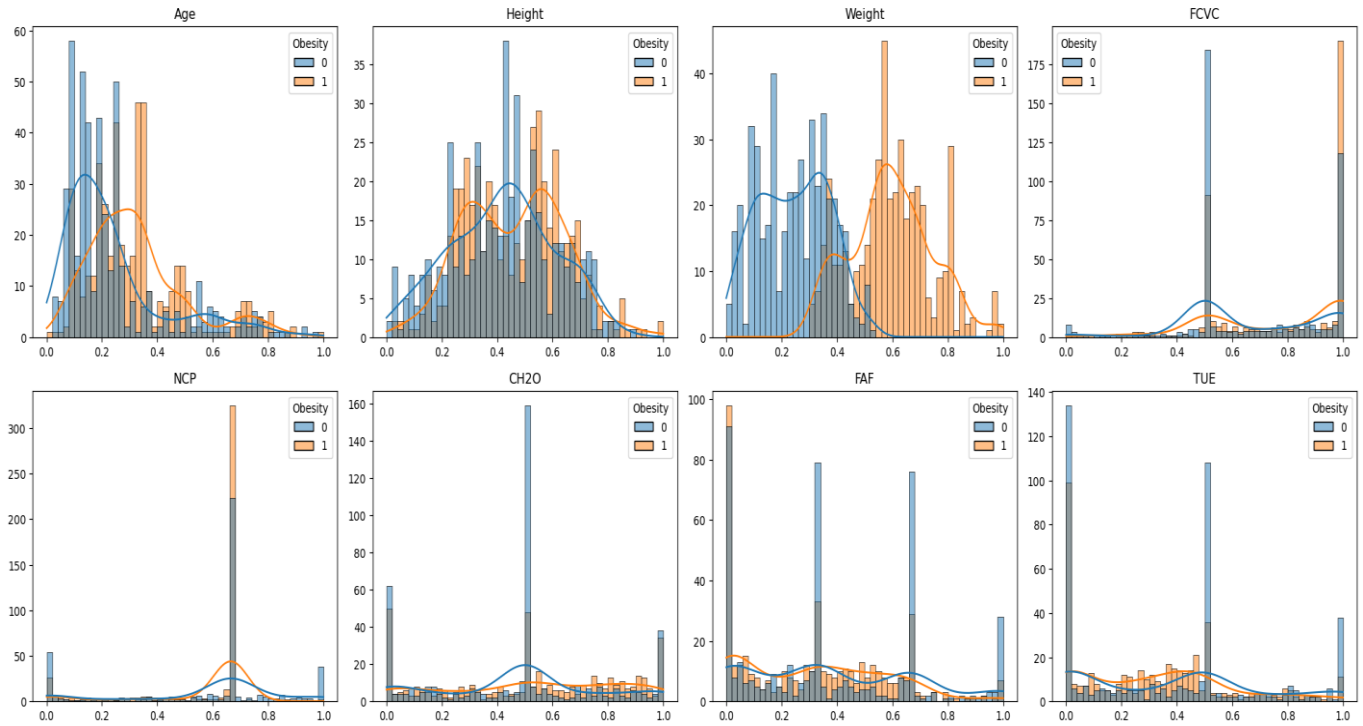
CAEC_Sometimes	CAEC_no	SMOKE_yes	SCC_yes	CALC_Sometimes	CALC_no	MTRANS_Bike	MTRANS_Motorbike	MTRANS_Public_Transportation
1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0

- Eliminamos la columna “NObeyesdad” de df2 ya que ya no nos hace falta:

```
df2.drop(columns='NObeyesdad', inplace=True)
```

- Representamos a través de distribuciones como se relacionan las variables numéricas con la variable objetivo:

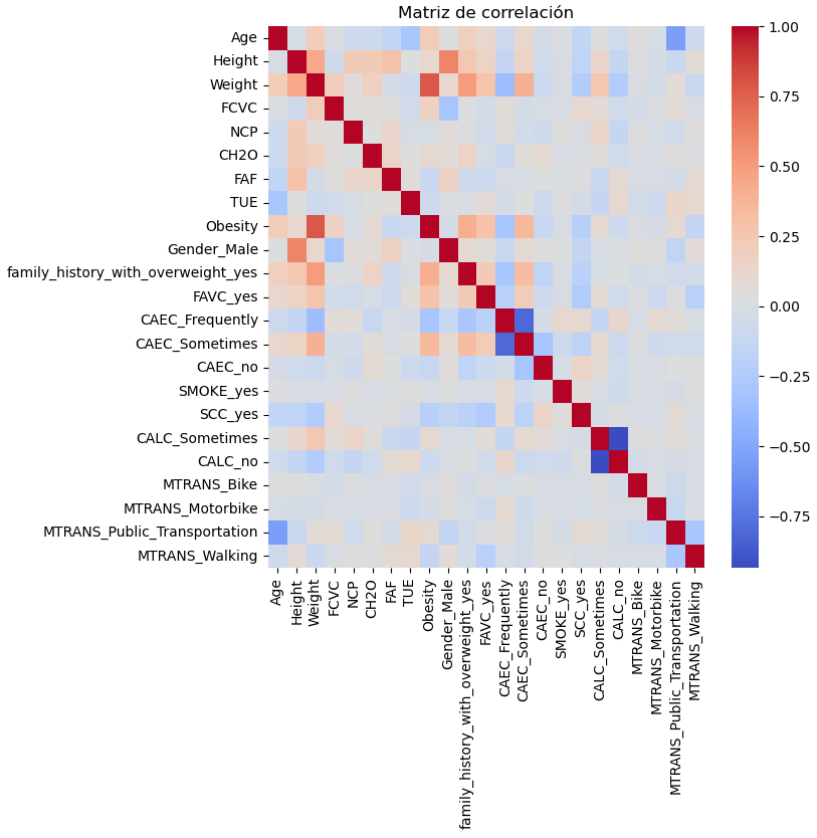
```
plt.figure(figsize=(20, 20))
for i, col in enumerate(df2[num_cols]):
    plt.subplot(5, 4, i+1)
    sns.histplot(df2, x=col, hue='Obesity', bins=50, kde=True)
    plt.title(col)
    plt.xlabel('')
    plt.ylabel('')
plt.tight_layout()
plt.show()
```



A través de las curvas de distribución y de cómo se relacionen podremos decir si esa variable está ligada o relacionada en mayor o menor medida con la variable objetivo. Como sucede por ejemplo con la variable "Weight" o con la variable "Age" (en menor medida), las curvas de distribución se pueden apreciar separadas a la largo de los valores. Esta separación clara en las distribuciones de las clases es indicativa de que la variable es un predictor fuerte de la variable objetivo y puede ser útil para la clasificación. Por otro lado, si las distribuciones se superponen significativamente como puede ser el caso de la variable "Height", puede ser más difícil para el modelo de clasificación distinguir entre las clases y tendrá una influencia menor en la variable objetivo.

- Se realiza la matriz de correlación:

```
plt.figure(figsize=(7, 7))
sns.heatmap(df2.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de correlación')
plt.show()
```



La matriz de correlación es una tabla que muestra las correlaciones entre todas las variables de un conjunto de datos. Cada celda de la matriz contiene el coeficiente de correlación entre dos variables. En este caso, podemos reafirmar lo que dijimos anteriormente sobre la variable "Weight" ya que tiene un coeficiente de correlación alto con la variable objetivo "Obesity". Otras variables que se muestran como posibles

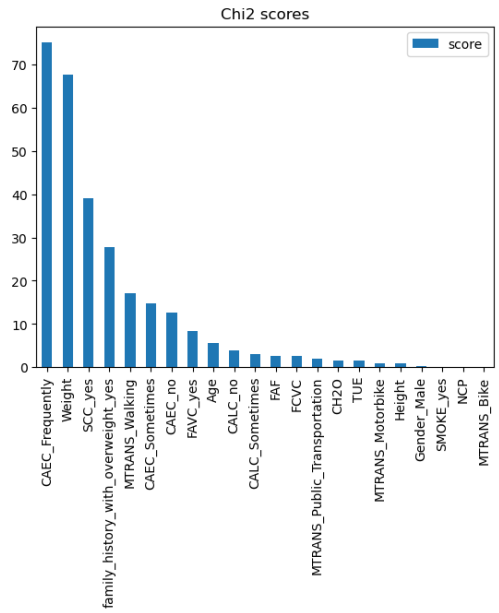
variables predictoras al tener un coeficiente de correlación alto con la variable objetivo son “CAEC_Sometimes”, “family_history_with_overweight_yes”, “FAVC_yes” o “Age”.

6. Selección de variables predictivas (Feature Selection)

La selección de variables predictivas, o en ingles Feature Selection (FS), es un proceso donde se estudia qué variables son más influyentes a la hora de predecir la variable objetivo. En este caso se han usado dos métodos distintos para comprobar la importancia de las variables: chi2 y feature importances.

- Chi2:

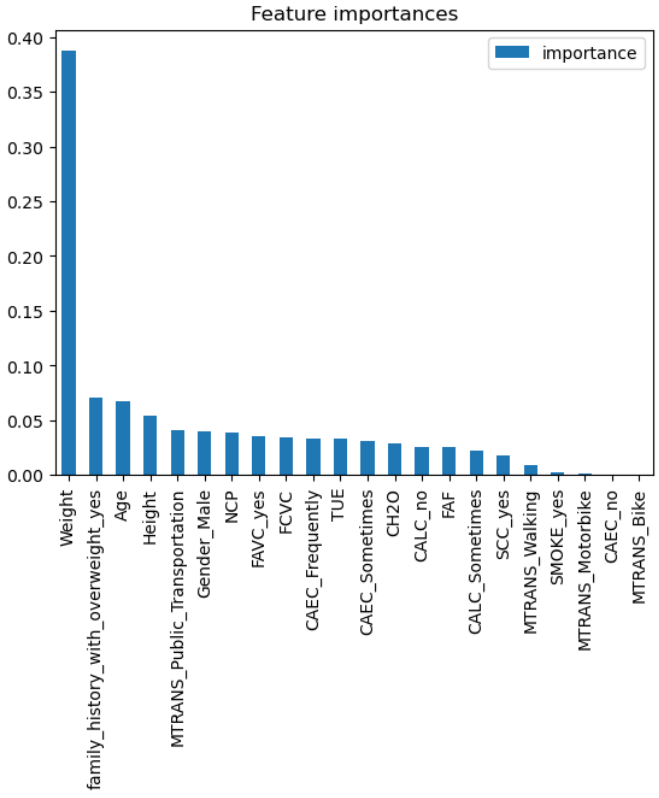
```
# Usamos chi2 para estudiar la relación entre las variables y la variable objetivo

# Separamos las variables predictoras de la variable objetivo
X = df2.drop(columns=target)
y = df2[target]

# Aplicamos el método de selección de variables
test = SelectKBest(score_func=chi2, k='all')
fit = test.fit(X, y)

fit.transform(X)

# Creamos un dataframe con los resultados
scores = pd.DataFrame(fit.scores_, index=X.columns,
columns=['score']).sort_values(by='score', ascending=False)
scores

# Visualizamos los resultados
plt.figure(figsize=(10, 5))
scores.plot(kind='bar')
plt.title('Chi2 scores')
plt.show()
```



De izquierda a derecha se muestran las variables que según la prueba chi2 son más influyentes a la hora de predecir a la variable objetivo.

- Feature importances:

```
# Usamos feature_importances_ para estudiar la relación entre las
variables y la variable objetivo

model = ExtraTreesClassifier(n_estimators=10, random_state=seed)
model.fit(X, y)

# Creamos un dataframe con los resultados
importances = pd.DataFrame(model.feature_importances_, index=X.columns,
columns=['importance']).sort_values(by='importance', ascending=False)
importances

# Visualizamos los resultados
plt.figure(figsize=(10, 5))
importances.plot(kind='bar')
plt.title('Feature importances')
plt.show()
```



De igual forma que para la prueba anterior, de izquierda a derecha se muestran que variables son más influyentes.

A continuación, se han escogido 3 conjuntos de posibles variables predictoras seleccionadas a partir de las pruebas anteriores. Se usarán para crear modelos de regresión logística y a través de las medidas de bondad de estos modelos nos

quedaremos con el conjunto de variables predictoras que usaremos para el resto del estudio.

- Los conjuntos de posibles variables predictoras son los siguientes:

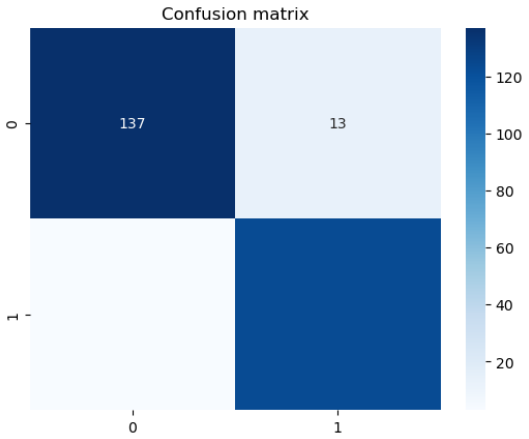
```
# Vamos a estudiar 3 modelos de regresión logistica, cada modelo usará
las siguientes variables:
var_model1 = ['Age', 'Weight', 'Height',
'family_history_with_overweight_yes', 'CAEC_Frequently', 'SCC_yes',
'MTRANS_Walking']
var_model2 = ['Age', 'Weight', 'Height',
'family_history_with_overweight_yes']
var_model3 = ['Weight', 'family_history_with_overweight_yes',
'CAEC_Frequently', 'SCC_yes', ]
```

En el primer conjunto "var_model1" se han seleccionado todas aquellas variables que tuviesen, según las pruebas anteriores, influencia sobre la variable objetivo y en "var_model2" y "var_model3" se han escogido una combinación de estas.

- Se dispone a realizar los modelos de regresión logística con estos conjuntos de variables:

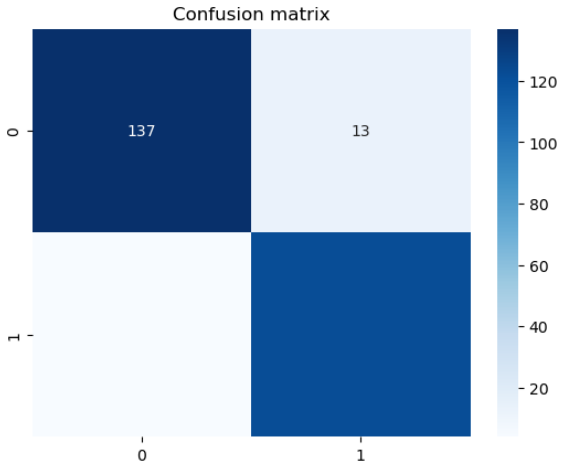
```
# Hacemos el modelo 1

# Separamos las variables predictoras de la variable objetivo
X1 = df2[var_model1]
y1 = df2[target]

# Dividimos el dataset en train y test
X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1,
test_size=0.3, random_state=seed)

# Creamos el modelo
model1 = LogisticRegression(random_state=seed)
model1.fit(X_train1, y_train1)

# Hacemos predicciones
y_pred1 = model1.predict(X_test1)

# Evaluamos el modelo
print('Accuracy:', accuracy_score(y_test1, y_pred1))
print(classification_report(y_test1, y_pred1))
sns.heatmap(confusion_matrix(y_test1, y_pred1), annot=True, cmap='Blues',
fmt='d')
plt.title('Confusion matrix')
plt.show()
```

```
Accuracy: 0.9420289855072463
              precision    recall  f1-score   support

           0       0.98      0.91      0.94       150
           1       0.90      0.98      0.94       126

    accuracy                           0.94       276
   macro avg       0.94      0.94      0.94       276
weighted avg       0.94      0.94      0.94       276
```



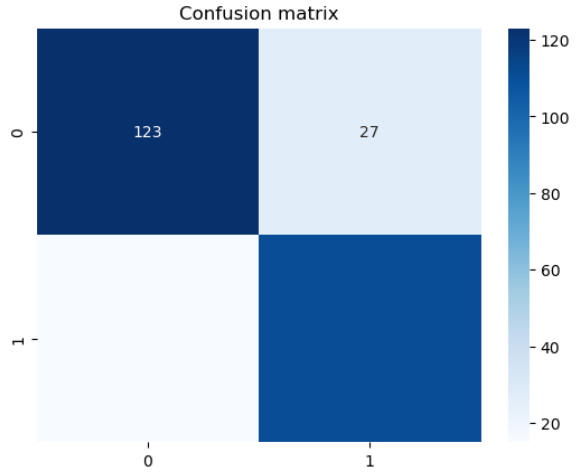

```python
# Hacemos el modelo 2

# Separamos las variables predictoras de la variable objetivo
X2 = df2[var_model2]
y2 = df2[target]

# Dividimos el dataset en train y test
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2, y2,
test_size=0.3, random_state=seed)

# Creamos el modelo
model2 = LogisticRegression(random_state=seed)
model2.fit(X_train2, y_train2)

# Hacemos predicciones
y_pred2 = model2.predict(X_test2)

# Evaluamos el modelo
print('Accuracy:', accuracy_score(y_test2, y_pred2))
print(classification_report(y_test2, y_pred2))
sns.heatmap(confusion_matrix(y_test2, y_pred2), annot=True, cmap='Blues',
fmt='d')
plt.title('Confusion matrix')
plt.show()
```

```
Accuracy: 0.9384057971014492
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       150
           1       0.90      0.97      0.93       126

    accuracy                           0.94       276
   macro avg       0.94      0.94      0.94       276
weighted avg       0.94      0.94      0.94       276
```




```python
# Hacemos el modelo 3

# Separamos las variables predictoras de la variable objetivo
X3 = df2[var_model3]
y3 = df2[target]

# Dividimos el dataset en train y test
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3,
test_size=0.3, random_state=seed)

# Creamos el modelo
model3 = LogisticRegression(random_state=seed)
model3.fit(X_train3, y_train3)

# Hacemos predicciones
y_pred3 = model3.predict(X_test3)

# Evaluamos el modelo
print('Accuracy:', accuracy_score(y_test3, y_pred3))
print(classification_report(y_test3, y_pred3))
sns.heatmap(confusion_matrix(y_test3, y_pred3), annot=True, cmap='Blues',
fmt='d')
plt.title('Confusion matrix')
plt.show()
```

```
Accuracy: 0.8478260869565217
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       150
           1       0.80      0.88      0.84       126

    accuracy                           0.85       276
   macro avg       0.85      0.85      0.85       276
weighted avg       0.85      0.85      0.85       276
```




```
# Se comparan los accuracy de los modelos
accuracy_model1 = accuracy_score(y_test1, y_pred1)
accuracy_model2 = accuracy_score(y_test2, y_pred2)
accuracy_model3 = accuracy_score(y_test3, y_pred3)
# Se muestran los resultados
print('Accuracy modelo 1:', accuracy_model1)
print('Accuracy modelo 2:', accuracy_model2)
print('Accuracy modelo 3:', accuracy_model3)
```

```
Accuracy modelo 1: 0.9420289855072463
Accuracy modelo 2: 0.9384057971014492
Accuracy modelo 3: 0.8478260869565217
```

Por tanto, nos quedaremos con el conjunto de variables predictoras “var_model1” ya que son las que han obtenido una medida de bondad “accuracy” mejor para el modelo 1.

7. Red neuronal a partir de regresión logística, comparación en términos de Accuracy

Habiendo escogido un conjunto de variables predictoras, se dispone a realizar un modelo de red neuronal usando estas variables. Usaremos el método "GridSearchCV" en este caso para encontrar el mejor modelo de red neuronal en términos de Accuracy y para los posteriores casos. "GridSearchCV" es una técnica para encontrar los mejores hiperparámetros usando validación cruzada para un modelo en particular.

Los hiperparámetros se definirán en "param_grid". Uno de los parámetros influyentes en una red neuronal es "hidden_layer_sizes", que representa el numero de nodos de la capa oculta.

- Este valor podemos estimarlo aproximadamente de la siguiente forma:

```
# Segun la teoría podemos calcular el numero de nodos de la capa oculta
de forma aproximada
# h * ( k + 1 ) + h + 1 = num_obs / 20
# siendo h el numero de nodos de la capa oculta y k el numero de
variables predictoras

# Calculamos el numero de nodos de la capa oculta
k = len(var_model1)
num_obs = df2.shape[0]
h = ( (num_obs / 20) - 1) / (k + 2)
h
```

```
5.0
```

Por tanto, tomaremos la referencia inicial de 5 nodos para la capa oculta.

- Buscamos la mejor red neuronal en términos del accuracy:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
red = MLPClassifier(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'hidden_layer_sizes': [5,6,7],
    'activation': ['tanh', 'relu'],
    'max_iter': [2000],
```

```
    'alpha': [0.0001, 0.001],
}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy','roc_auc', 'precision', 'recall', 'f1']

# Aplicamos GridSearchCV
grid = GridSearchCV(estimator=red, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)

 # Obtener el mejor modelo

best_model = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
MLPClassifier(activation='tanh', hidden_layer_sizes=5, max_iter=2000,
              random_state=3314)
Precisión accuracy del modelo: 0.9945652173913043
```

Según "best_estimator" el mejor modelo será aquel con esos hiperparámetros y el correspondiente accuracy.

- Se procede a observar el posible sobreajuste comparando predicciones en train y test. Predicciones significativamente mayores en train que en test puede indicar sobreajuste.

```
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
print('Comprobar que la diferencia no sea muy grande por temas de
sobreajuste')
```

```
Se tiene un accuracy para train de: 0.998641304347826
Se tiene un accuracy para test de: 0.9945652173913043
Comprobar que la diferencia no sea muy grande por temas de sobreajuste
```

Se comprueba que apenas hay diferencia en el accuracy de parte de train respecto a la de prueba.

- Mostramos resultados del modelo:

```
print('Resultados para Modelo')
print(classification_report(y_test, y_test_pred))
```

```
Resultados para Modelo
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       101
           1       0.99      1.00      0.99        83

    accuracy                           0.99       184
   macro avg       0.99      1.00      0.99       184
weighted avg       0.99      0.99      0.99       184
```

- Mostramos si la variable objetivo está igualmente balanceada en la parte train y test.

```
# Mostramos como se distribuye la variable target en la parte train y
test
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

De manera que está balanceada en la parte train y test. Esto es importante para evitar sesgos en el modelo.

- Mostramos los resultados de las mejores 10 redes neuronales en términos de accuracy:

```
results =
pd.DataFrame(grid.cv_results_).sort_values(by='mean_test_accuracy',
ascending=False)
```

```
results[['mean_test_accuracy', 'mean_test_roc_auc',
'mean_test_precision', 'mean_test_recall', 'mean_test_f1',
'params']].head(10)
```

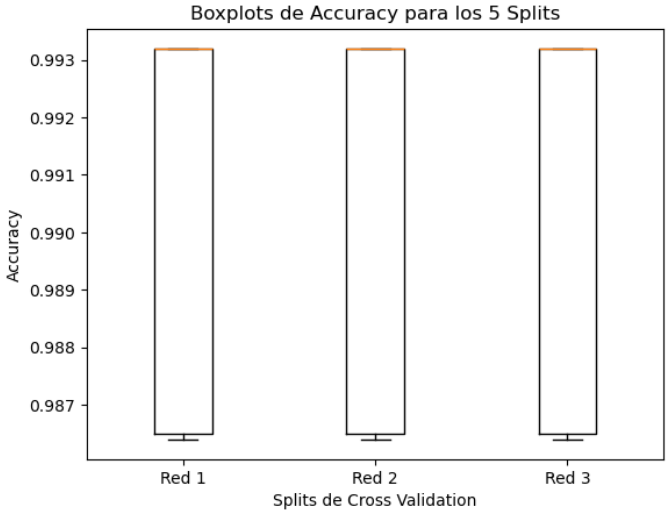
	mean_test_accuracy	mean_test_roc_auc	mean_test_precision	mean_test_recall	mean_test_f1	params
0	0.990495	0.997927	0.994805	0.986486	0.990547	{'activation': 'tanh', 'alpha': 0.0001, 'hidde...
1	0.990495	0.997779	0.994805	0.986486	0.990547	{'activation': 'tanh', 'alpha': 0.0001, 'hidde...
2	0.990495	0.997631	0.994805	0.986486	0.990547	{'activation': 'tanh', 'alpha': 0.0001, 'hidde...
3	0.990495	0.997927	0.994805	0.986486	0.990547	{'activation': 'tanh', 'alpha': 0.001, 'hidden...
4	0.990495	0.997890	0.994805	0.986486	0.990547	{'activation': 'tanh', 'alpha': 0.001, 'hidden...
5	0.990495	0.997631	0.994805	0.986486	0.990547	{'activation': 'tanh', 'alpha': 0.001, 'hidden...
6	0.990495	0.997816	0.994805	0.986486	0.990547	{'activation': 'relu', 'alpha': 0.0001, 'hidde...
8	0.990495	0.997520	0.994805	0.986486	0.990547	{'activation': 'relu', 'alpha': 0.0001, 'hidde...
9	0.990495	0.997816	0.994805	0.986486	0.990547	{'activation': 'relu', 'alpha': 0.001, 'hidden...
11	0.990495	0.997520	0.994805	0.986486	0.990547	{'activation': 'relu', 'alpha': 0.001, 'hidden...

- Se seleccionan las 3 redes candidatas a mejor red y se analiza su robustez a lo largo del cross validation.

```
ac_1 = results[['split0_test_accuracy', 'split1_test_accuracy',
'split2_test_accuracy', 'split3_test_accuracy',
'split4_test_accuracy']].iloc[0]
ac_2 = results[['split0_test_accuracy', 'split1_test_accuracy',
'split2_test_accuracy', 'split3_test_accuracy',
'split4_test_accuracy']].iloc[1]
ac_3 = results[['split0_test_accuracy', 'split1_test_accuracy',
'split2_test_accuracy', 'split3_test_accuracy',
'split4_test_accuracy']].iloc[2]
```

- Se crea un boxplot para los valores de accuracy de las 3 redes candidatas:

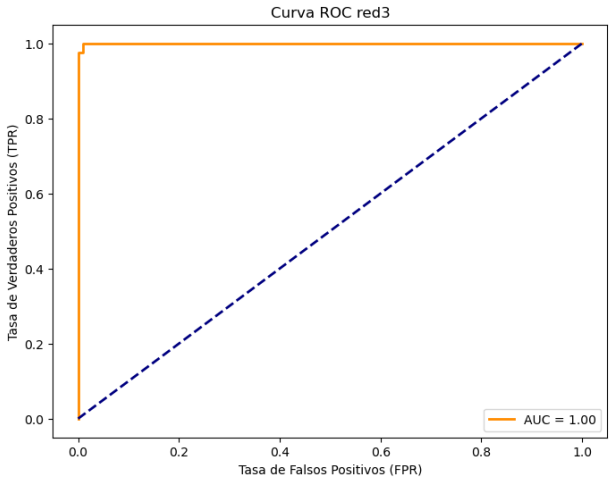
```
plt.boxplot([ac_1.values, ac_2.values, ac_3.values], labels = ['Red 1',
'Red 2', 'Red 3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



Por tanto, en términos de accuracy no nos importa tanto la red que seleccionemos ya que todas tienen un accuracy y una varianza similar. En este caso nos quedamos con la red 3.

- Representamos la curva ROC de la red 3:

```
red_log = MLPClassifier(**results.iloc[2].params)
red_log.fit(X_train, y_train)
y_pred_log = red_log.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_log)
roc_auc = auc(fpr, tpr)
print(f"\nÁrea bajo la curva ROC (AUC) para la red en test:
{roc_auc:.2f}")
# Graficar la curva ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC red3')
plt.legend(loc="lower right")
plt.show()
```



A través de los altos valores de accuracy para train y test, podemos observar como la mejor red neuronal puede estar altamente sobre ajustada. Por ejemplo, observando la curva ROC, vemos que deja por debajo toda el área posible, esto es señal de que nuestro modelo está altamente sobre ajustado, de hecho, sería lo más sobre ajustado que podría estar aunque habría que tener en cuenta otras métricas de evaluación para confirmar esto.

- Seleccionamos el mejor modelo y lo guardamos para compararlo posteriormente con otras redes. "_log" hará referencia a que las variables predictoras han sido seleccionadas a través de un modelo de regresión logística:

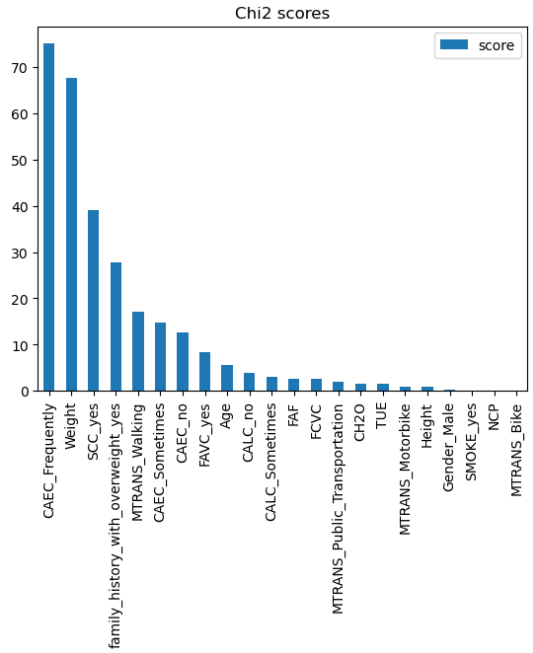
```
ac_log = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy',
'split4_test_accuracy']].iloc[2]
auc_log = results[['split0_test_roc_auc',
'split1_test_roc_auc','split2_test_roc_auc', 'split3_test_roc_auc',
'split4_test_roc_auc']].iloc[2]
```

8. Red neuronal a través de "*select k best*", comparación en términos de AUC

En este apartado se aplicará un proceso de selección de variables "select k best" tomando *k*=4. Con las variables que obtendremos, encontraremos el mejor modelo de red neuronal pero esta vez será en términos de AUC.

- Usamos "select k best":

```
# Usamos chi2 para estudiar la relación entre las variables y la variable
objetivo

# Separamos las variables predictoras de la variable objetivo
X = df2.drop(columns=target)
y = df2[target]

# Aplicamos el método de selección de variables
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)

fit.transform(X)

# Creamos un dataframe con los resultados
scores = pd.DataFrame(fit.scores_, index=X.columns,
columns=['score']).sort_values(by='score', ascending=False)
scores

# Visualizamos los resultados
plt.figure(figsize=(10, 5))
scores.plot(kind='bar')
plt.title('Chi2 scores')
plt.show()
```



- Tomando las cuatro mejores variables, encontraremos la mejor red neuronal en términos de AUC:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[scores.index[:4]]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
red = MLPClassifier(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'hidden_layer_sizes': [3,5,6,7],
    'activation': ['tanh', 'relu'],
    'max_iter': [2000],
    'alpha': [0.0001, 0.001]
}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro', ]

# Usamos GridSearchCV para encontrar la mejor red neuronal
grid = GridSearchCV(estimator=red, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='roc_auc')
grid.fit(X_train, y_train)
```

```
# Obtener el mejor modelo
best_model = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
MLPClassifier(activation='tanh', hidden_layer_sizes=5, max_iter=2000,
              random_state=3314)
Precisión del modelo: 0.8315217391304348
```

- Se procede a observar el posible sobreajuste comparando predicciones en train y test:

```
# Predicciones en conjunto de entrenamiento y prueba
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
print('Comprobar que la diferencia no sea muy grande por temas de
sobreajuste')
```

```
Se tiene un accuracy para train de: 0.8980978260869565
Se tiene un accuracy para test de: 0.8315217391304348
Comprobar que la diferencia no sea muy grande por temas de sobreajuste
```

Se comprueba que puede haber un posible sobreajuste al haber una diferencia entre ambos valores de accuracy.

- Los resultados del modelo son:

```
Resultados para Modelo
              precision    recall  f1-score   support

           0       0.87      0.81      0.84       101
           1       0.79      0.86      0.82        83

    accuracy                           0.83       184
   macro avg       0.83      0.83      0.83       184
weighted avg       0.83      0.83      0.83       184
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
 1    0.504076
 0    0.495924
 Name: proportion, dtype: float64,
 Obesity
 0    0.548913
 1    0.451087
 Name: proportion, dtype: float64)
```

Por tanto, se encuentra correctamente balanceada.

- Se muestran los resultados ordenando de mayor a menor en términos de AUC:

```
results =
pd.DataFrame(grid.cv_results_).sort_values(by='mean_test_roc_auc',
ascending=False)
results[['param_hidden_layer_sizes', 'param_activation', 'param_alpha',
'mean_test_roc_auc', 'mean_test_accuracy', 'mean_test_precision_macro',
'mean_test_recall_macro', 'mean_test_f1_macro']].head(10)
```

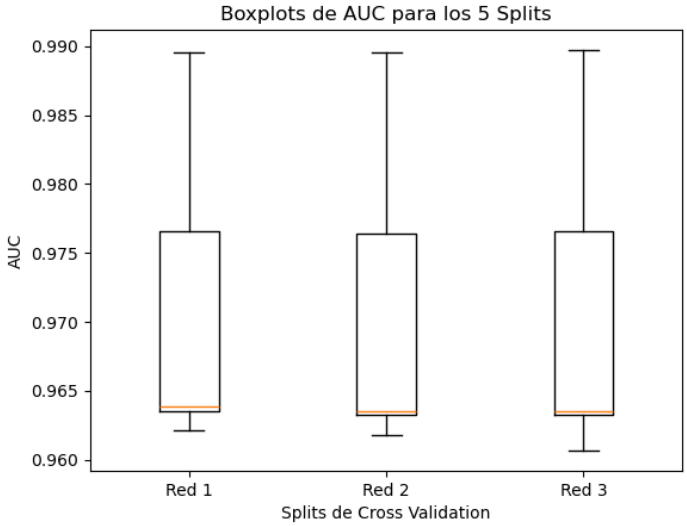
	param_hidden_layer_sizes	param_activation	param_alpha	mean_test_roc_auc	mean_test_accuracy	mean_test_precision_macro	mean_test_recall_macro	mean_test_f1_macro
1	5	tanh	0.0001	0.971127	0.894052	0.895744	0.894201	0.893920
5	5	tanh	0.001	0.971127	0.894052	0.895744	0.894201	0.893920
2	6	tanh	0.0001	0.970906	0.894052	0.896000	0.894219	0.893904
6	6	tanh	0.001	0.970906	0.894052	0.896000	0.894219	0.893904
3	7	tanh	0.0001	0.970758	0.895413	0.897412	0.895571	0.895261
7	7	tanh	0.001	0.970758	0.895413	0.897412	0.895571	0.895261
10	6	relu	0.0001	0.970721	0.898134	0.900958	0.898421	0.897953
14	6	relu	0.001	0.970647	0.898134	0.900958	0.898421	0.897953
0	3	tanh	0.0001	0.970350	0.892692	0.894487	0.892850	0.892556
4	3	tanh	0.001	0.970350	0.892692	0.894487	0.892850	0.892556

- Se seleccionan las 3 redes candidatas y se analiza su robustez del AUC a lo largo de cross validation:

```
auc_1 = results[['split0_test_roc_auc',
'split1_test_roc_auc','split2_test_roc_auc', 'split3_test_roc_auc',
'split4_test_roc_auc']].iloc[0]
auc_2 = results[['split0_test_roc_auc',
'split1_test_roc_auc','split2_test_roc_auc', 'split3_test_roc_auc',
'split4_test_roc_auc']].iloc[2]
auc_3 = results[['split0_test_roc_auc',
'split1_test_roc_auc','split2_test_roc_auc', 'split3_test_roc_auc',
'split4_test_roc_auc']].iloc[4]
```

- Se crea un boxplot para los valores de AUC de las 3 redes candidatas:

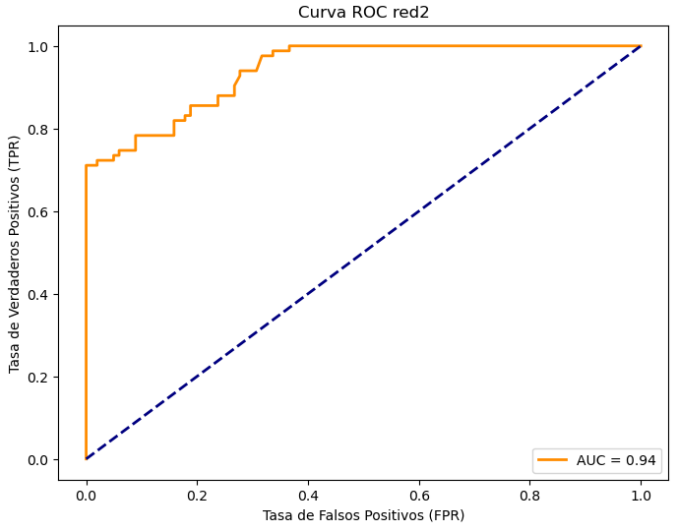
```
plt.boxplot([auc_1.values, auc_2.values, auc_3.values], labels = ['Red
1', 'Red 2', 'Red 3'])
plt.title('Boxplots de AUC para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('AUC')
plt.show()
```



Apenas hay diferencia entre las 3 redes en términos de AUC, por tanto, nos quedaremos con la red 3 ya que al menos tiene mejor accuracy que las otras dos.

- Representamos la curva ROC para la red 3:

```
red_k = MLPClassifier(**results.iloc[4].params)
red_k.fit(X_train, y_train)

y_pred_k = red_k.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_k)
roc_auc = auc(fpr, tpr)

print(f"\nÁrea bajo la curva ROC (AUC) para la red 2 en test:
{roc_auc:.2f}")

# Graficar la curva ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC red2')
plt.legend(loc="lower right")
plt.show()
```



- Seleccionamos el mejor modelo y lo guardamos para compararlo posteriormente con otras redes. "_k" hará referencia a que las variables predictoras han sido seleccionadas a través de un modelo de "select k best":

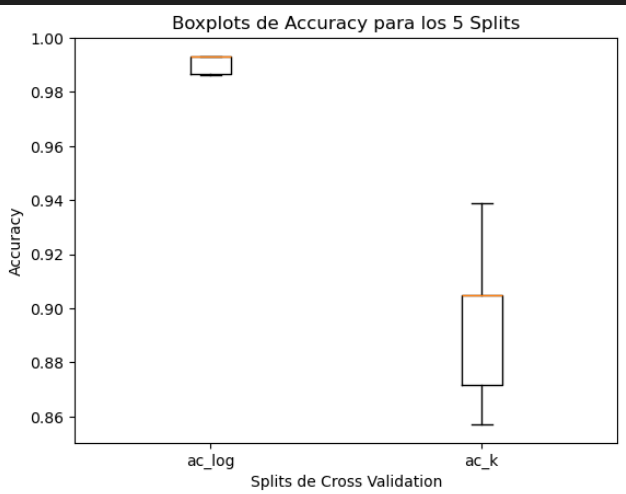
```
ac_k = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy', 'split3_test_accuracy',
'split4_test_accuracy']].iloc[4]
auc_k = results[['split0_test_roc_auc',
'split1_test_roc_auc','split2_test_roc_auc', 'split3_test_roc_auc',
'split4_test_roc_auc']].iloc[4]
```

9. Comparación de redes neuronales

Habiendo creado las dos redes neuronales de los apartados 7 y 8 disponemos de compararlas en términos de accuracy y AUC.

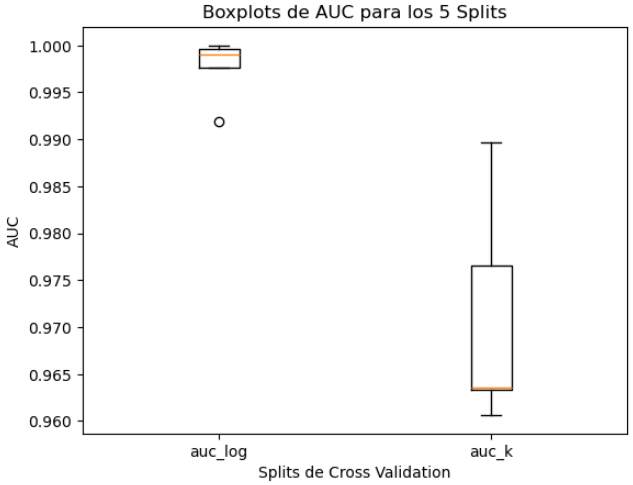
- Comparación de accuracy:

```
# Comparamos el accuracy mediante boxplot de ac_log y ac_k
plt.boxplot([ac_log.values, ac_k.values], labels = ['ac_log', 'ac_k'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



- Comparación de AUC:

```
# Comparamos el AUC mediante boxplot de auc_log y auc_k
plt.boxplot([auc_log.values, auc_k.values], labels = ['auc_log',
'auc_k'])
plt.title('Boxplots de AUC para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('AUC')
plt.show()
```



Conclusión: Seleccionaremos la red obtenida a través de las variables que escogimos primeramente a través de un modelo de regresión logística. Esta red tiene, como muestran las gráficas, un mejor valor de accuracy y AUC, además de tener menor varianza que la red con las 4 variables obtenidas a través de “Select K Best”.

10. Árbol de decisión (Decision Tree Classifier)

A continuación, se realizará la mejor búsqueda paramétrica para lograr el mejor árbol según cuatro métodos diferentes de validación de la bondad de la clasificación. Para el árbol ganador lo representaremos gráficamente, mostraremos sus reglas en formato texto y graficaremos la importancia de sus variables.

- Realizamos la búsqueda del mejor árbol de decisión:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
tree = DecisionTreeClassifier(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'max_depth': [1,2,3,4],
    'criterion': ['gini', 'entropy'],
    'min_samples_split': [2, 5, 7],
}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro']

# Usamos GridSearchCV para encontrar el mejor árbol de decisión
grid = GridSearchCV(estimator=tree, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)
```

- Obtenemos el mejor modelo:

```
best_model = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=3314)
Precisión del modelo: 0.967391304347826
```

- Vemos el accuracy de la parte train y test:

```
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 0.9836956521739131
Se tiene un accuracy para test de: 0.967391304347826
```

No hay mucha diferencia entre ambos valores de accuracy.

- Mostramos los resultados del modelo:

```
print('Resultados para Modelo')
print(classification_report(y_test, y_test_pred))
```

```
Resultados para Modelo
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       101
           1       0.93      1.00      0.97        83

    accuracy                           0.97       184
   macro avg       0.97      0.97      0.97       184
weighted avg       0.97      0.97      0.97       184
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

- Mostramos los resultados de los modelos obtenidos ordenando de mayor a menor accuracy:

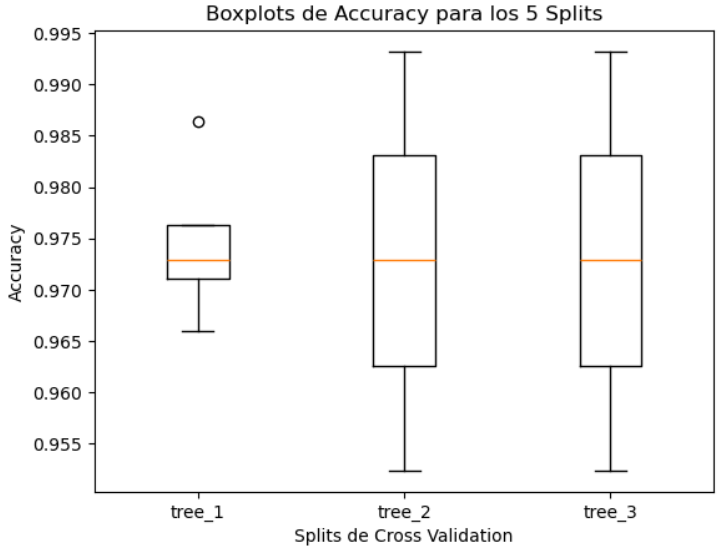
```
results =
pd.DataFrame(grid.cv_results_).sort_values(by='mean_test_accuracy',
ascending=False)
```

```
results[['param_max_depth', 'param_criterion', 'param_min_samples_split',
'mean_test_accuracy', 'mean_test_roc_auc', 'mean_test_precision_macro',
'mean_test_recall_macro', 'mean_test_f1_macro']].head(10)
```

	param_max_depth	param_criterion	param_min_samples_split	mean_test_accuracy	mean_test_roc_auc	mean_test_precision_macro	mean_test_recall_macro	mean_test_f1_macro
18	3	entropy	2	0.974186	0.986796	0.974794	0.974065	0.974170
20	3	entropy	7	0.974186	0.986796	0.974794	0.974065	0.974170
19	3	entropy	5	0.974186	0.986796	0.974794	0.974065	0.974170
9	4	gini	2	0.972817	0.983791	0.973167	0.972806	0.972808
10	4	gini	5	0.972817	0.983791	0.973167	0.972806	0.972808
6	3	gini	2	0.971456	0.983373	0.971831	0.971436	0.971446

- Se seleccionan los 3 árboles candidatos:

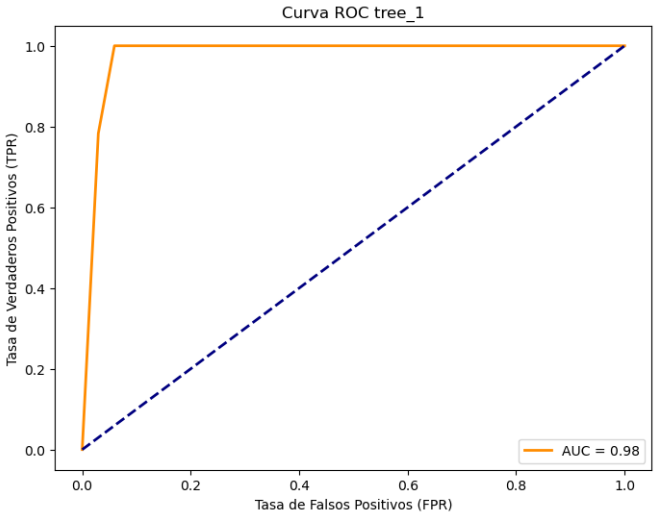
```
tree_1 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[0]
tree_2 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[3]
tree_3 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[5]

# Crear un boxplot para los 3 valores de accuracy
plt.boxplot([tree_1.values,tree_2.values,tree_3.values], labels =
['tree_1','tree_2','tree_3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



Por tanto, seleccionamos el tree_1 ya que tiene una mayor precisión y menor varianza.

- Hacemos predicciones con el mejor árbol de decisión y mostramos la curva ROC para el tree_1:

```
tree = DecisionTreeClassifier(**results.iloc[0].params)
tree.fit(X_train, y_train)
y_pred_tree = tree.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_tree)
roc_auc = auc(fpr, tpr)
print(f"\nÁrea bajo la curva ROC (AUC) para el árbol 1 en test:
{roc_auc:.2f}")

# Graficar la curva ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC tree_1')
plt.legend(loc="lower right")
plt.show()
```



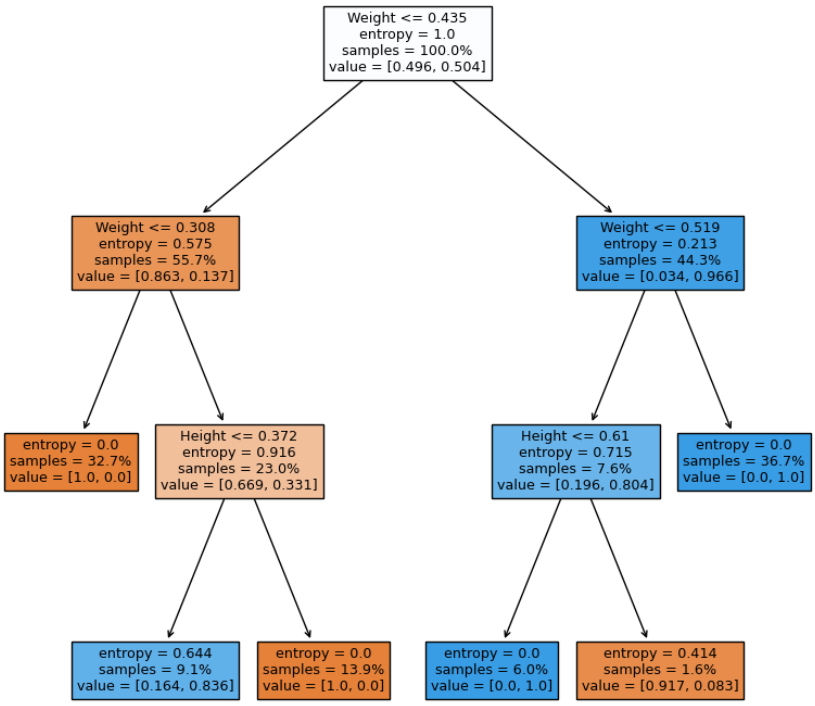
- Mostramos además la matriz de confusión de este modelo:

```
# Hacemos predicciones
tree = DecisionTreeClassifier(**results.iloc[0].params)
predictor = tree.fit(X_train, y_train).predict(X_test)

# Matriz de confusión
print(confusion_matrix(y_test, predictor))
```

```
[[95  6]
 [ 0 83]]
```

- Representamos gráficamente el árbol de decisión tree_1:

```
plt.figure(figsize=(10,10))
plot_tree(tree, feature_names=X.columns.tolist(), filled=True,
proportion=True)
plt.show()
```



- Mostramos las reglas en formato texto del árbol de decisión tree_1:

```
from sklearn.tree import export_text
r = export_text(tree, feature_names=X.columns.tolist())
print(r)
```

```
|--- Weight <= 0.44
|   |--- Weight <= 0.31
|   |   |--- class: 0
|   |--- Weight >  0.31
|   |   |--- Height <= 0.37
|   |   |   |--- class: 1
|   |   |--- Height >  0.37
|   |   |   |--- class: 0
|--- Weight >  0.44
|   |--- Weight <= 0.52
|   |   |--- Height <= 0.61
|   |   |   |--- class: 1
|   |   |--- Height >  0.61
|   |   |   |--- class: 0
|   |--- Weight >  0.52
|   |   |--- class: 1
```

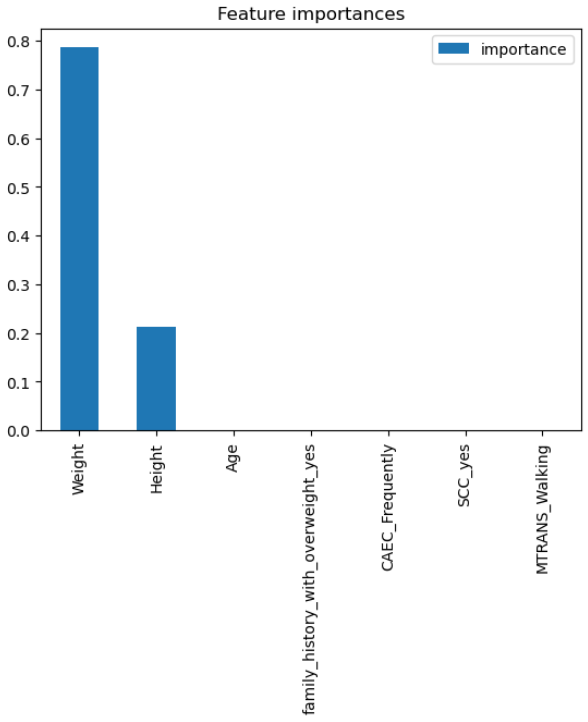
- Se estudia la importancia o valor predictivo de cada variable en el modelo:

```
importances = grid.best_estimator_.feature_importances_
importances = pd.DataFrame(importances, index=X.columns,
columns=['importance']).sort_values(by='importance', ascending=False)
importances
```

	importance
Weight	0.786645
Height	0.213355
Age	0.000000
family_history_with_overweight_yes	0.000000
CAEC_Frequently	0.000000
SCC_yes	0.000000
MTRANS_Walking	0.000000

- Graficamos la importancia de las variables del árbol:

```
plt.figure(figsize=(10, 5))
importances.plot(kind='bar')
plt.title('Feature importances')
plt.show()
```



Por consiguiente, hemos realizado un árbol de decisión con un valor considerable de accuracy, donde sus variables más importantes son el peso "Weight" y la altura "Height". A través de estas variables es capaz de predecir fácilmente si una persona es obesa o no.

11. Bagging

En este apartado se realizará la mejor búsqueda paramétrica para lograr el mejor modelo con Bagging en términos de Accuracy.

- Hacemos la búsqueda del mejor modelo bagging usando GridSearchCV:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
bagging = BaggingClassifier(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'n_estimators': [10, 50, 100,250],
    'max_samples': [1,75,150,300],
    'max_features': [1,4,7,9],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False]
}
# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro']

# Usamos GridSearchCV para encontrar el mejor modelo de bagging
grid = GridSearchCV(estimator=bagging, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)
```

- Obtenemos el mejor modelo:

```
best_model = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)
# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
BaggingClassifier(bootstrap=False, max_features=7, max_samples=300,
                  n_estimators=50, random_state=3314)
Precisión del modelo: 0.967391304347826
```

- Comparamos el accuracy para la parte train y para la parte test:

```
y_train_pred = best_model.predict(X_train)
y_test_pred = best_model.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 1.0
Se tiene un accuracy para test de: 0.967391304347826
```

- Se muestran los resultados para el modelo:

```
print('Resultados para Modelo')
print(classification_report(y_test, y_test_pred))
```

```
Resultados para Modelo
              precision    recall  f1-score   support

           0       1.00      0.94      0.97       101
           1       0.93      1.00      0.97        83

    accuracy                           0.97       184
   macro avg       0.97      0.97      0.97       184
weighted avg       0.97      0.97      0.97       184
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

La variable target está bien distribuida en test y en train.

- Mostramos los resultados de los modelos obtenidos ordenando de mayor a menor accuracy:

```
results =
pd.DataFrame(grid.cv_results_).sort_values(by='mean_test_accuracy',
ascending=False)
results[['param_n_estimators', 'param_max_samples', 'param_max_features',
'param_bootstrap', 'param_bootstrap_features', 'mean_test_accuracy',
'mean_test_roc_auc', 'mean_test_precision_macro',
'mean_test_recall_macro', 'mean_test_f1_macro']].head(10)
```

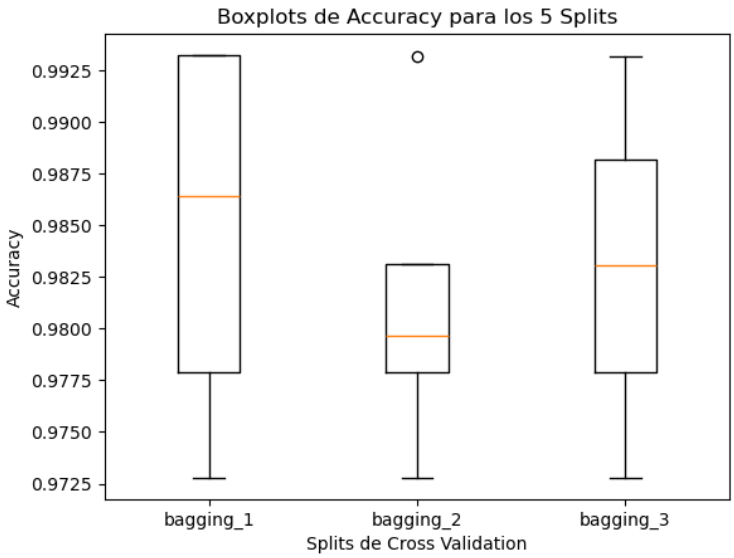
param_n_estimators	param_max_samples	param_max_features	param_bootstrap	param_bootstrap_features	mean_test_accuracy	mean_test_roc_auc
250	300	7	False	False	0.987764	0.999335
100	300	7	False	False	0.987764	0.999409
50	300	7	False	False	0.987764	0.999223
50	300	7	True	False	0.986422	0.999335
100	300	7	True	False	0.985062	0.999373

- Se seleccionan los 3 modelos candidatos:

```
bagging_1 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[0]
bagging_2 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[4]
bagging_3 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[6]
```

- Se representa un boxplot de accuracy para los 5 splits de cada modelo:

```
plt.boxplot([bagging_1.values,bagging_2.values,bagging_3.values], labels
= ['bagging_1','bagging_2','bagging_3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



De estos tres modelos de bagging seleccionaremos el segundo modelo "bagging_2" ya que tiene una mayor precisión y menor varianza. El accuracy del mejor modelo es considerablemente alto, pero tampoco existe gran diferencia entre el accuracy de la parte train y test por tanto puede que hayamos obtenido un buen modelo de predicción usando el método bagging pero habría que estudiarlo con otras métricas de evaluación para confirmar que nuestro modelo no está sobre ajustado.

12. Random Forest

En este apartado se realizará la mejor búsqueda paramétrica para lograr el mejor modelo con Random Forest en términos de Accuracy.

- Hacemos la búsqueda del mejor modelo con Random Forest usando GridSearchCV:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
forest = RandomForestClassifier(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'n_estimators': [50, 100,250],
    'max_depth': [6,7,8],
    'criterion': ['gini', 'entropy'],
    'min_samples_leaf': [1,2,3],
    'min_samples_split': [2, 5, 7],
    'bootstrap': [True, False]

}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro']

# Usamos GridSearchCV para encontrar el mejor modelo de random forest
grid = GridSearchCV(estimator=forest, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)
```

En cuanto a la elección de hiperparámetros, en un primer entrenamiento del modelo, los mejores modelos tenían “param_max_depth” = 4, indicador de que aumentando el rango de este hiperparámetro se podría mejorar el modelo. Por tanto, lo ampliamos hasta 8. Igual pasaba con “param_min_samples leaf”. Idealmente, esto se podría hacer infinitamente hasta encontrar el valor del parámetro ideal.

- Obtenemos el mejor modelo:

```
best_model_RF = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model_RF.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
RandomForestClassifier(max_depth=8, min_samples_split=7, n_estimators=50,
                       random_state=3314)
Precisión del modelo: 0.9619565217391305
```

- Comparamos el accuracy para la parte train y para la parte test:

```
y_train_pred = best_model_RF.predict(X_train)
y_test_pred = best_model_RF.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 0.998641304347826
Se tiene un accuracy para test de: 0.9619565217391305
```

- Los resultados del modelo son:

```
Resultados para Modelo
              precision    recall  f1-score   support

           0       1.00      0.93      0.96       101
           1       0.92      1.00      0.96        83

    accuracy                           0.96       184
   macro avg       0.96      0.97      0.96       184
weighted avg       0.96      0.96      0.96       184
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

Esta correctamente distribuida la variable target en la parte train y test.

- Mostramos los resultados de los modelos obtenidos ordenando de mayor a menor accuracy:

```
results =
pd.DataFrame(grid.cv_results_).sort_values(by='mean_test_accuracy',
ascending=False)
results[['param_n_estimators', 'param_max_depth', 'param_criterion',
'param_min_samples_leaf', 'param_min_samples_split', 'param_bootstrap',
'mean_test_accuracy', 'mean_test_roc_auc', 'mean_test_precision_macro',
'mean_test_recall_macro', 'mean_test_f1_macro']].head(10)
```

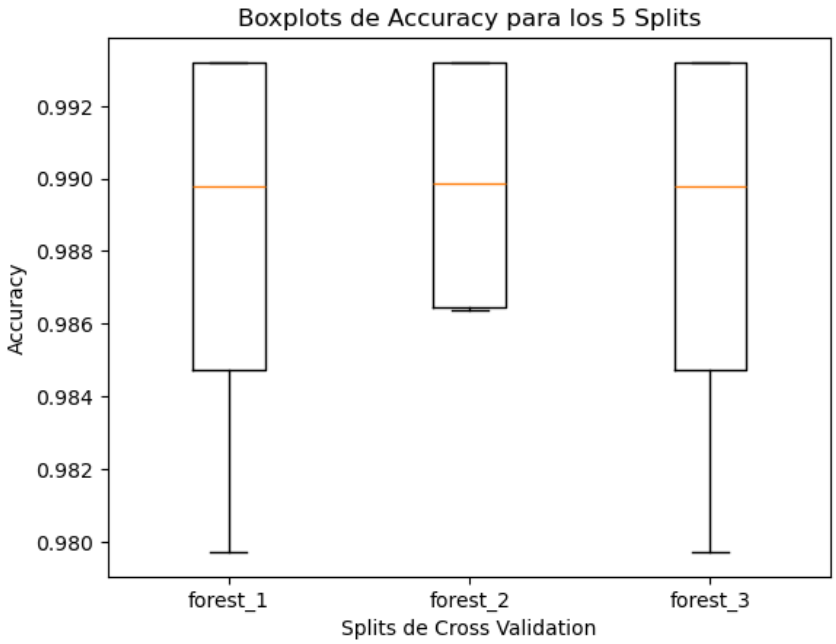
	param_n_estimators	param_max_depth	param_criterion	param_min_samples_leaf	param_min_samples_split	param_bootstrap	mean_test_accuracy
141	50	8	entropy	1	7	True	0.989143
60	50	8	gini	1	7	True	0.989143
135	50	8	entropy	1	2	True	0.989134
112	100	7	entropy	1	5	True	0.989134
136	100	8	entropy	1	2	True	0.989134
138	50	8	entropy	1	5	True	0.987783

- Se seleccionan los tres modelos candidatos:

```
forest_1 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[0]
forest_2 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[2]
forest_3 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[5]
```

- Se representa un boxplot de accuracy para los 5 splits de cada modelo:

```
plt.boxplot([forest_1.values,forest_2.values,forest_3.values], labels =
['forest_1','forest_2','forest_3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



Elegimos el modelo 2 “forest_2” ya que tiene una mayor precisión y menor varianza. El accuracy del mejor modelo es considerablemente alto, pero tampoco existe gran diferencia entre el accuracy de la parte train y test por tanto puede que hayamos obtenido un buen modelo de predicción usando el método Random Forest, pero habría que estudiarlo con otras métricas de evaluación para confirmar que nuestro modelo no está sobre ajustado.

13. Gradient Boosting

En este apartado se realizará la mejor búsqueda paramétrica para lograr el mejor modelo con Gradient Boosting en términos de Accuracy.

- Hacemos la búsqueda del mejor modelo con Gradient Boosting usando GridSearchCV:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
gradient = GradientBoostingClassifier(random_state=seed)

# Definimos los hiperparámetros
params = {
    'n_estimators': [400,600],
    'n_iter_no_change': [None,5,10],
    'max_depth': [5, 10],
    'min_samples_leaf' : [30],
    'min_samples_split': [5, 10, 50],
}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro']

# Usamos GridSearchCV para encontrar el mejor modelo de Gradient Boosting
grid = GridSearchCV(estimator=gradient, param_grid=params, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)
```

- Obtenemos el mejor modelo:

```
best_model_GB = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model_GB.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
GradientBoostingClassifier(max_depth=10, min_samples_leaf=30,
                           min_samples_split=5, n_estimators=400,
                           n_iter_no_change=10, random_state=3314)
Precisión del modelo: 0.967391304347826
```

- Comparamos el accuracy para la parte train y para la parte test:

```
y_train_pred = best_model_GB.predict(X_train)
y_test_pred = best_model_GB.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 0.9972826086956522
Se tiene un accuracy para test de: 0.967391304347826
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

- Mostramos los resultados de los modelos obtenidos de mayor a menor accuracy:

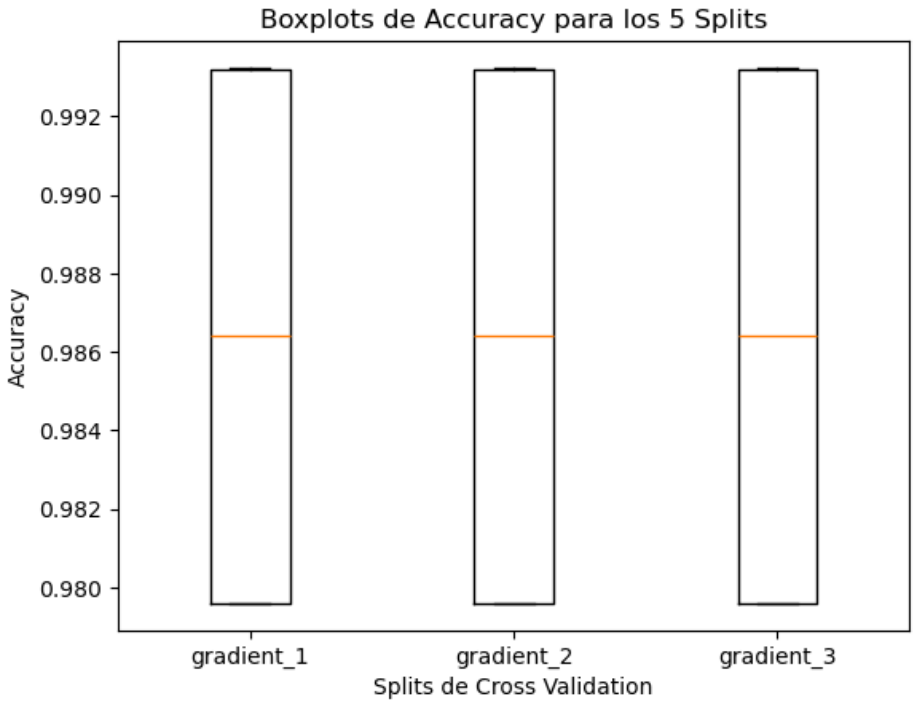
param_n_estimators	param_n_iter_no_change	param_max_depth	param_min_samples_split	param_min_samples_leaf	mean_test_accuracy
600	10	10	50	30	0.989125
400	10	10	50	30	0.989125
600	10	10	5	30	0.989125
600	10	10	10	30	0.989125
400	10	10	5	30	0.989125

- Se seleccionan los 3 modelos candidatos:

```
gradient_1 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[0]
gradient_2 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[3]
gradient_3 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[9]
```

- Se representa un boxplot de accuracy para los 5 splits de cada modelo:

```
plt.boxplot([gradient_1.values,gradient_2.values,gradient_3.values],
labels = ['gradient_1','gradient_2','gradient_3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



Cualquier de los 3 candidatos que escojamos tiene prácticamente el mismo accuracy, por tanto, da igual el que escojamos. En este caso hemos tomado el segundo modelo "gradient_2". De igual forma que para los algoritmos anteriores, el accuracy del mejor modelo es considerablemente alto, pero tampoco existe gran diferencia entre el accuracy de la parte train y test por tanto puede que hayamos obtenido un buen modelo de predicción usando el método Gradient Boosting, pero habría que estudiarlo con otras métricas de evaluación para confirmar que nuestro modelo no está sobreajustado.

14.XGBoost

En este apartado se realizará la mejor búsqueda paramétrica para lograr el mejor modelo con XGBoost en términos de Accuracy.

- Hacemos la búsqueda del mejor modelo con XGBoost usando GridSearchCV:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
xgb = XGBClassifier(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'n_estimators': [100,200,300],
    'eta' : [0.1,0.4,0.7],
    'gamma' : [0.1,0.5,1],
    'max_depth': [5, 10]
}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro']

# Usamos GridSearchCV para encontrar el mejor modelo de XGBoost
grid = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)
```

- Se obtiene el mejor modelo:

```
# Obtener el mejor modelo
best_model_XGB = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model_XGB.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eta=0.4, eval_metric=None,
              feature_types=None, gamma=1, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=5,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=100,
              n_jobs=None, num_parallel_tree=None, ...)
Precisión del modelo: 0.9619565217391305
```

- Comparamos el accuracy para la parte train y para la parte test:

```
y_train_pred = best_model_XGB.predict(X_train)
y_test_pred = best_model_XGB.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 1.0
Se tiene un accuracy para test de: 0.9619565217391305
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
 1    0.504076
 0    0.495924
 Name: proportion, dtype: float64,
 Obesity
 0    0.548913
 1    0.451087
 Name: proportion, dtype: float64)
```

- Mostramos los resultados de los modelos obtenidos de mayor a menor accuracy:

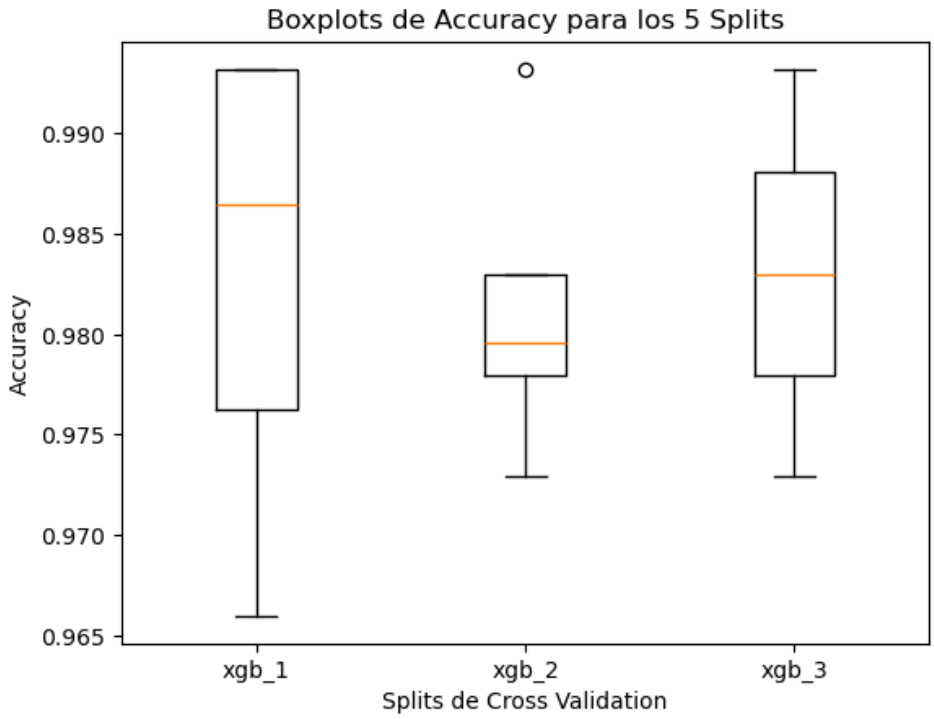
param_n_estimators	param_eta	param_gamma	param_max_depth	mean_test_accuracy
200	0.4	1	10	0.986422
300	0.4	1	10	0.986422
100	0.4	1	5	0.986422
200	0.4	1	5	0.986422
300	0.4	1	5	0.986422

- Se seleccionan los tres modelos candidatos:

```
xgb_1 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[0]
xgb_2 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[6]
xgb_3 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[27]
```

- Se representa un boxplot de accuracy para los 5 splits de cada modelo:

```
plt.boxplot([xgb_1.values,xgb_2.values,xgb_3.values], labels =
['xgb_1','xgb_2','xgb_3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



Por tanto, seleccionamos el modelo "xgb_2" ya que tiene una mayor precisión y menor varianza. De igual forma que para los algoritmos anteriores, el accuracy del mejor modelo es considerablemente alto, aunque en este caso sí existe diferencia apreciable entre el accuracy de la parte train y test por tanto puede que nuestro modelo esté sobre ajustado, aunque también puede que hayamos obtenido un buen modelo de predicción usando el método XGBoost.

15. SVM

En este apartado se realizará la mejor búsqueda paramétrica para lograr el mejor modelo con SVM en términos de Accuracy.

- Hacemos la búsqueda del mejor modelo con SVM usando GridSearchCV:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo
svc = SVC(random_state=seed)

# Definimos los hiperparámetros
param_grid = {
    'C': [10,11,12],
    'kernel': ['linear', 'rbf', 'poly'],
    'degree': [2,3,4],
    'gamma': ['scale', 'auto']
}

# Definimos las métricas de evaluación
scoring_metrics = ['accuracy', 'roc_auc', 'precision_macro',
'recall_macro', 'f1_macro']

# Usamos GridSearchCV para encontrar el mejor modelo de SVM
grid = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5,
scoring=scoring_metrics, refit='accuracy')
grid.fit(X_train, y_train)
```

- Obtenemos el mejor modelo:

```
best_model_SVC = grid.best_estimator_
print(grid.best_estimator_)
y_pred_rf = best_model_SVC.predict(X_test)
accuracy_rf_gs = accuracy_score(y_test, y_pred_rf)

# Evaluar el rendimiento del modelo
print(f'Precisión del modelo: {accuracy_rf_gs}')
```

```
SVC(C=10, degree=2, kernel='linear', random_state=3314)
Precisión del modelo: 0.9891304347826086
```

- Comparamos el accuracy para la parte train y para la parte test:

```
y_train_pred = best_model_SVC.predict(X_train)
y_test_pred = best_model_SVC.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 0.9972826086956522
Se tiene un accuracy para test de: 0.9891304347826086
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

- Mostramos los resultados de los modelos obtenidos de mayor a menor accuracy:

```
results =
pd.DataFrame(grid.cv_results_).sort_values(by='mean_test_accuracy',
ascending=False)
results[['param_C', 'param_kernel', 'param_degree', 'param_gamma',
'mean_test_accuracy', 'mean_test_roc_auc', 'mean_test_precision_macro',
'mean_test_recall_macro', 'mean_test_f1_macro']].head(10)
```

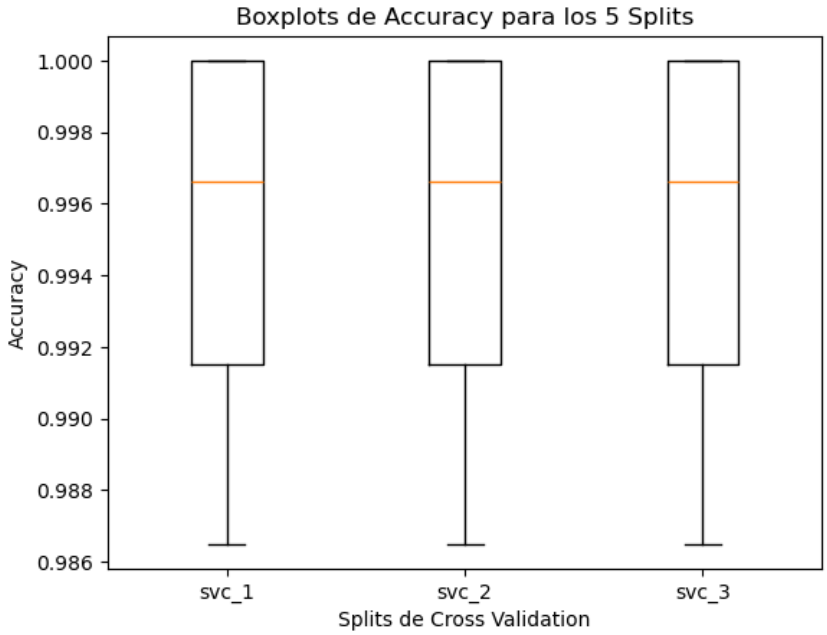
param_C	param_kernel	param_degree	param_gamma	mean_test_accuracy
10	linear	2	scale	0.993216
10	linear	4	auto	0.993216
12	linear	4	auto	0.993216
12	linear	4	scale	0.993216
12	linear	3	auto	0.993216

- Se seleccionan los 3 modelos candidatos:

```
svc_1 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[0]
svc_2 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[6]
svc_3 = results[['split0_test_accuracy',
'split1_test_accuracy','split2_test_accuracy',
'split3_test_accuracy']].iloc[7]
```

- Se representa un boxplot de accuracy para los 5 splits de cada modelo:

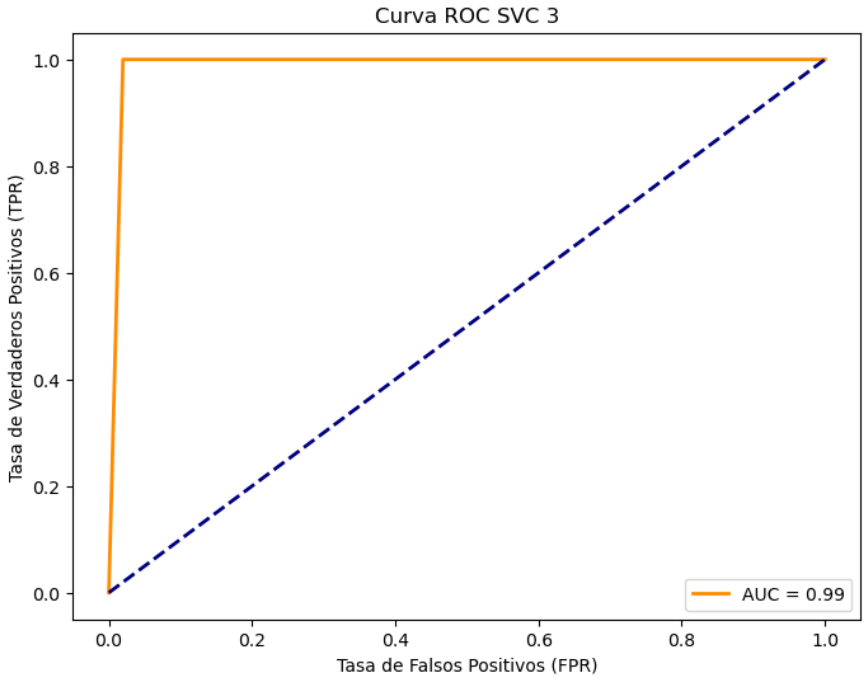
```
plt.boxplot([svc_1.values,svc_2.values,svc_3.values], labels =
['svc_1','svc_2','svc_3'])
plt.title('Boxplots de Accuracy para los 5 Splits')
plt.xlabel('Splits de Cross Validation')
plt.ylabel('Accuracy')
plt.show()
```



- Representamos la curva ROC:

```
y_pred_svc = svc.predict(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_svc)
roc_auc = auc(fpr, tpr)
print(f"\nÁrea bajo la curva ROC (AUC) para el SVC 3 en test:
{roc_auc:.2f}")

# Graficar la curva ROC
plt.figure(figsize=(8, 6))
```

```
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC SVC 3')
plt.legend(loc="lower right")
plt.show()
```



Da igual el modelo que escojamos ya que los tres son prácticamente iguales en términos de accuracy. De igual forma que para los algoritmos anteriores, el accuracy del mejor modelo es considerablemente alto, pero tampoco existe gran diferencia entre el accuracy de la parte train y test por tanto puede que hayamos obtenido un buen modelo de predicción usando el método Gradient Boosting, pero habría que estudiarlo con otras métricas de evaluación para confirmar que nuestro modelo no está sobre ajustado.

16. Ensamblado de Bagging con un mismo clasificador base que no sea un árbol.

En este apartado se realiza un método de Ensamblado de Bagging con un mismo clasificador base que no sea un árbol. En este caso hemos usado sin ningún motivo especial una regresión logística como el modelo base.

- Realizamos en modelo:

```
# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos el modelo base
logistic = LogisticRegression(random_state=seed)

# Creamos el modelo de Bagging
bagging = BaggingClassifier(estimator=logistic, n_estimators=10,
random_state=seed)

# Entrenamos el modelo
bagging.fit(X_train, y_train)

# Hacemos predicciones
y_pred = bagging.predict(X_test)
```

- Evaluamos el rendimiento del modelo y se muestra el accuracy para la parte train y test:

```
# Evaluamos el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)

# Se procede a observar el posible sobreajuste comparando predicciones en
train y test.
# predicciones significativamente mayores en train que en test puede
indicar sobreajuste.
# Predicciones en conjunto de entrenamiento y prueba
y_train_pred = bagging.predict(X_train)
y_test_pred = bagging.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 0.96875
Se tiene un accuracy para test de: 0.9456521739130435
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
y_train.value_counts(normalize=True), y_test.value_counts(normalize=True)
```

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

Se distribuye de forma balanceada tanto en train como en test.

- Mostramos los resultados del modelo:

```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.92      0.95       101
           1       0.91      0.98      0.94        83

    accuracy                           0.95       184
   macro avg       0.94      0.95      0.95       184
weighted avg       0.95      0.95      0.95       184
```

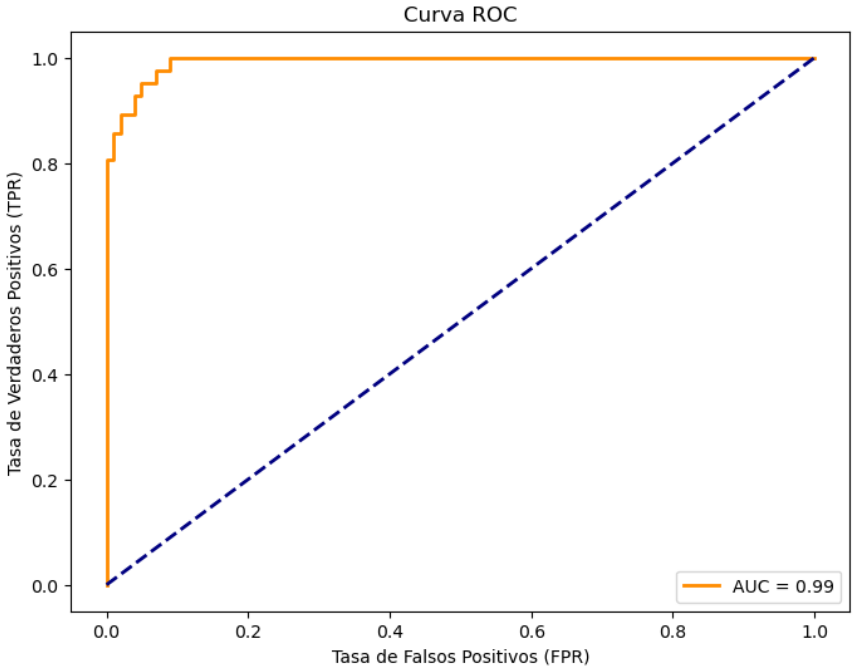
- Mostramos la matriz de confusión donde indica los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos:

```
confusion_matrix(y_test, y_test_pred)
```

```
array([[93,  8],
       [ 2, 81]], dtype=int64)
```

- Representamos la curva ROC:

```
y_pred_proba = bagging.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()
```



Finalmente hemos obtenido un modelo a través de un Ensamblado de Bagging usando una regresión logística como modelo el modelo base. Es importante destacar que la regresión logística, al ser un modelo lineal, puede no beneficiarse tanto del bagging como otros modelos más complejos, como árboles de decisión o modelos de ensamblado como Random Forest o Gradient Boosting Machines. Sin embargo, aun así, puede proporcionar mejoras en la precisión y la estabilidad del modelo, especialmente en conjuntos de datos ruidosos o con alta varianza.

De igual forma que para los algoritmos anteriores, el accuracy del mejor modelo es considerablemente alto, pero tampoco existe gran diferencia entre el accuracy de la parte train y test por tanto puede que hayamos obtenido un buen modelo de predicción usando este método, pero habría que estudiarlo con otras métricas de evaluación para confirmar que nuestro modelo no está sobre ajustado.

17. Stacking

En este apartado se ha realizado un modelo usando un método de Stacking escogiendo varios algoritmos de entrada y un modelo de ensamblaje o Stacking.

Los modelos base ("estimators") serán:

- Random Forest
- Gradient Boosting
- Decision Tree

El modelo de Stacking ("final_estimator") será:

- Random Forest

- Creamos el modelo:

```
# Stacking escogiendo varios algoritmos de entrada y el modelo que se
prefiera como modelo de ensamblaje

# Separamos las variables predictoras de la variable objetivo
X = df2[var_model1]
y = df2[target]

# Dividimos el dataset en train y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=seed)

# Creamos los modelos base
rf = RandomForestClassifier(random_state=seed)
gb = GradientBoostingClassifier(random_state=seed)
dt = DecisionTreeClassifier(random_state=seed)

# Creamos el modelo de Stacking con un random forest como modelo final
stacking = StackingClassifier(estimators=[('rf', rf), ('gb', gb), ('dt',
dt)], final_estimator=rf)

# Entrenamos el modelo
stacking.fit(X_train, y_train)

# Hacemos predicciones
y_pred = stacking.predict(X_test)
```

- Evaluamos el rendimiento del modelo y mostramos el accuracy para la parte train y test:

```
# Evaluamos el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)

# Se procede a observar el posible sobreajuste comparando predicciones en
train y test.
# predicciones significativamente mayores en train que en test puede
indicar sobreajuste.
# Predicciones en conjunto de entrenamiento y prueba
y_train_pred = stacking.predict(X_train)
y_test_pred = stacking.predict(X_test)
print(f'Se tiene un accuracy para train de:
{accuracy_score(y_train,y_train_pred)}')
print(f'Se tiene un accuracy para test de:
{accuracy_score(y_test,y_test_pred)}')
```

```
Se tiene un accuracy para train de: 0.998641304347826
Se tiene un accuracy para test de: 0.9619565217391305
```

- Mostramos como se distribuye la variable target en la parte train y test:

```
(Obesity
1    0.504076
0    0.495924
Name: proportion, dtype: float64,
Obesity
0    0.548913
1    0.451087
Name: proportion, dtype: float64)
```

- Mostramos los resultados del modelo:

```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       101
           1       0.93      0.99      0.96        83

    accuracy                           0.96       184
   macro avg       0.96      0.96      0.96       184
weighted avg       0.96      0.96      0.96       184
```

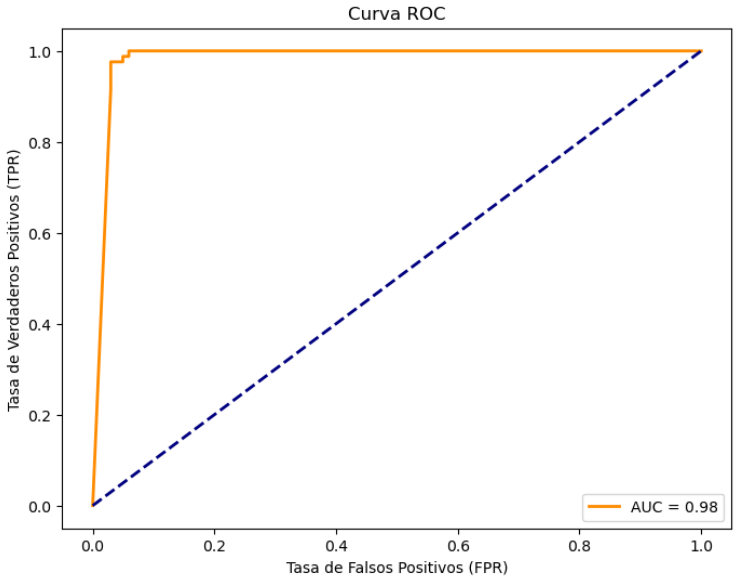
- Mostramos la matriz de confusión:

```
confusion_matrix(y_test, y_test_pred)
```

```
array([[95,  6],
       [ 1, 82]], dtype=int64)
```

- Representamos la curva ROC:

```
y_pred_proba = stacking.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC =
{roc_auc:.2f}')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()
```



En este modelo se han obtenido las predicciones de varios modelos y a través de otro modelo se han combinado sus predicciones. Usar un Random Forest como modelo de stacking puede ser beneficioso porque los Random Forests suelen ser modelos robustos y tener buen rendimiento en una variedad de conjuntos de datos. Además, al utilizar un Random Forest como modelo de stacking, se pueden capturar relaciones más complejas entre las características generadas por los modelos base.

En este caso hemos obtenido un modelo final con un accuracy considerablemente bueno, aunque al igual que en los anteriores casos tampoco existe gran diferencia entre el accuracy de la parte train y test por tanto puede que hayamos obtenido un buen modelo de predicción usando este método, pero habría que estudiarlo con otras métricas de evaluación para confirmar que nuestro modelo no está sobreajustado.

18. Discusión y conclusión

En este apartado discutiremos y presentaremos algunos puntos a destacar.

- Por lo general, las redes neuronales necesitan por su naturaleza, un gran volumen de datos para que digamos de alguna forma, las hagamos "inteligentes". Si en nuestro caso tenemos una muestra de instancias bastante pequeña como es 920, la red neuronal no podrá entrenarse correctamente y tenderá a fallar, sobre ajustándose, por ejemplo.
- Los modelos obtenidos en la mayoría de los apartados tienen un accuracy demasiado alto, señal de que el modelo puede estar sobre ajustado, pero cuando se compara el accuracy de la parte train y test no suele haber señal de que haya sobreajuste ya que no hay una diferencia suficientemente apreciable. Por tanto, dejamos como posibilidad el sobreajuste de los modelos con posibilidad de comprobarlo a través de distintas metodologías que en este estudio no se ha realizado para que no se extendiese. Algunas metodologías que podrían usarse para comprobar si los modelos están sobre ajustados son estudiar las curvas de aprendizaje o el uso de Early Stopping que permite detener el proceso de entrenamiento en el momento óptimo, evitando así que el modelo se sobreajuste a los datos de entrenamiento.
- Hubiese sido interesando haber realizado el estudio eliminando las variables peso "Weight" o la altura "Height" debidas a su alta y lógica influencia sobre la variable objetivo. Esto nos hubiese permitido haber descubierto otras variables que pudiesen ser predictoras y no se identificasen tan fácilmente como puede ser el peso.
- Se podría haber hecho el estudio como un problema de clasificación no binario donde la variable objetivo fuese "NObeyesdad" y se predijese si el individuo tiene 'Insufficient_Weight', 'Overweight_Level_I', 'Obesity_Type_II', 'Overweight_Level_II', 'Normal_Weight', 'Obesity_Type_III' o 'Obesity_Type_I'. Pero se ha decidido por simplicidad hacer el estudio como un problema de clasificación binaria.

19. Bibliografía

- Hastie, T., Tibshirani, R., Friedman, J. (2009): The Elements of Statistical Learning: data mining, inference and prediction.
- Michie, D., Spiegelhalter, D.J., Taylor, C.C. (1994) Machine Learning, Neural and Statistical Classification.
- Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984.Classification and Regression Trees. 1nd ed. Boca Raton, Florida: Chapman;
- https://scikit-learn.org/stable/