UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA
DIVISIÓN DE INVESTIGACIÓN Y POSGRADO

Machine Learning: Práctica 3. Normalización de datos y balance de clases.

Alumno:
Ing. Enrique Mena Camilo

Profesor:
Dr. Marco Antonio Aceves Fernández

Marzo 2023

Índice

1. Objetivos

- Implementar normalización de atributos: Comprobar el correcto funcionamiento del algoritmo mediante la comparación de las distribuciones de los datos previo y posterior a normalizar.
- Implementar por lo menos 1 método de normalización de los siguientes:
  - Normalización min-max
  - Normalización mean
  - Normalización z-score
  - Normalización unit-vector
- Implementar por lo menos 1 método de normalización de los siguientes:
  - Normalización de frecuencia
  - Normalización de logaritmo
  - Normalización por probabilidad acumulada
  - Normalización por categoría
  - Normalización L1, L2
  - Normalización por unidad compleja
  - Normalización por sigmoide
- Implementar balance de clases mediante algoritmos de generación de datos sintéticos

2. Introducción

La normalización de datos es una técnica esencial en el aprendizaje automático que se utiliza para preparar y pre-procesar los datos antes de entrenar un modelo. La normalización implica escalar los datos de manera que todas las características tengan el mismo rango de valores, lo que ayuda a evitar que ciertas características dominen el proceso de aprendizaje y mejora la precisión y estabilidad del modelo.

La normalización se puede realizar de varias formas, como la normalización min-max, la normalización z-score y la normalización de unidades vectoriales. Cada método tiene sus propias ventajas y desventajas y es importante elegir el método adecuado en función de los requisitos específicos del modelo.

Algunos ejemplos de algoritmos de aprendizaje automático que se benefician de la normalización de datos son:

- **Redes neuronales**: La normalización de datos es esencial para entrenar redes neuronales eficientes y estables. La normalización z-score y la normalización de unidades vectoriales son las técnicas de normalización más comunes en redes neuronales.
- **SVM (máquinas de soporte vectorial)**: La normalización de datos es importante para SVM porque el algoritmo busca el hiperplano que mejor separa las clases. La normalización min-max es una técnica de normalización común para SVM.
- **KNN (k-vecinos más cercanos)**: La normalización de datos es importante para KNN porque el algoritmo utiliza la distancia euclidiana para medir la similitud entre instancias. La normalización z-score es una técnica de normalización común para KNN.
- **Regresión lineal**: La normalización de datos es importante para la regresión lineal porque ayuda a evitar que una característica tenga un peso dominante en el modelo. La normalización min-max y la normalización z-score son técnicas de normalización comunes para la regresión lineal.

El desbalance de clases en aprendizaje automático se produce cuando una clase tiene muchas más muestras que otras. Esto puede causar problemas durante el entrenamiento del modelo, ya que el modelo puede aprender a predecir siempre la clase mayoritaria, lo que resulta en una precisión engañosa. También puede haber problemas en la validación del modelo, ya que el modelo puede tener una precisión alta simplemente porque la clase minoritaria se predice con poca frecuencia, lo que puede ser un problema en aplicaciones del mundo real.

Existen varios algoritmos que pueden ayudar a lidiar con el problema de desbalance de clases en aprendizaje automático, de los cuales, para fines de este trabajo, destacan los algoritmos de sobremuestreo sintético:

- **SMOTE (Synthetic Minority Over-sampling Technique)**: es uno de los algoritmos más utilizados para el sobremuestreo sintético. SMOTE genera nuevas instancias de la clase minoritaria interpolando las características de las instancias existentes de esta clase. Básicamente, SMOTE crea una instancia sintética entre dos instancias de la clase minoritaria cercanas en el espacio de características.
- **ADASYN (Adaptive Synthetic Sampling)**: es otro algoritmo de sobremuestreo sintético que también utiliza una técnica de interpolación. ADASYN tiene en cuenta la densidad de las instancias de la clase minoritaria y genera nuevas instancias en regiones más densas para reducir la posibilidad de sobremuestreo en regiones menos densas.
- **Borderline-SMOTE**: es una variante de SMOTE que solo genera instancias sintéticas en la frontera entre las clases minoritaria y mayoritaria. Borderline-SMOTE intenta enfocarse en las regiones de decisión más difíciles y evitar el sobremuestreo en las regiones de la clase minoritaria que están bien separadas de la clase mayoritaria.
- **MDO (Minority Data Oversampling)**: es un algoritmo de sobremuestreo sintético que utiliza una estrategia de vecinos más cercanos para generar nuevas instancias. MDO encuentra los vecinos más cercanos de las instancias de la clase minoritaria y crea nuevas instancias basadas en la media y la desviación estándar de los valores de características de los vecinos cercanos.

3. Marco teórico

3.1. Métodos de normalización de datos

3.1.1. Normalización min-max

La normalización min-max es un método común de normalización de datos utilizado en aprendizaje automático para ajustar los valores de un conjunto de datos dentro de un rango específico. El método se basa en la escala de los datos de manera proporcional al rango completo de los datos.

El proceso de normalización min-max implica la transformación de cada valor de la característica en una escala de 0 a 1, en relación con el valor mínimo y máximo de esa característica en el conjunto de datos. La fórmula para calcular el valor normalizado de cada valor de característica es la siguiente:

$$x_{normalizado} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Donde x representa la observación que se desea transformar, x_{min} representa el valor mínimo del conjunto de observaciones, y x_{max} representa el valor máximo del conjunto de observaciones.

Dicho valor normalizado puede ser escalado posteriormente al rango deseado mediante la aplicación de la ecuación siguiente:

$$x_{escalado} = x_{normalizado} * (r_{max} - r_{min}) + r_{min}$$

Donde $x_{normalizado}$ representa el resultado de normalizar la observación en el rango típico de 0 a 1, r_{max} representa el valor máximo al que se desea escalar los datos, y r_{min} representa el valor mínimo al que se desea escalar los datos.x

La normalización min-max es útil cuando se quiere comparar características con diferentes rangos de valores y para preparar los datos para algoritmos que requieren que los datos estén en un rango específico. Sin embargo, es importante tener en cuenta que la normalización min-max puede ser sensible a los valores atípicos, por lo que puede ser necesario utilizar otras técnicas de normalización si hay valores atípicos presentes en los datos.

3.1.2. Normalización z-score

La normalización z-score, también conocida como estandarización, es un método común de normalización de datos utilizado en aprendizaje automático para ajustar los valores de un conjunto de datos para que tengan una media de cero y una desviación estándar de uno.

El proceso de normalización z-score implica la transformación de cada valor de la característica en una escala basada en la media y la desviación estándar de esa característica

en el conjunto de datos. La fórmula para calcular el valor normalizado de cada valor de característica es la siguiente:

$$x_{normalizado} = \frac{x - \bar{x}}{\sigma}$$

Donde x es el valor sin escalar de la característica, $\bar{x}$ es la media de esa característica en el conjunto de datos, y σ es la desviación estándar de esa característica en el conjunto de datos.

La normalización z-score es útil cuando se desea comparar características con diferentes unidades de medida y para preparar los datos para algoritmos que requieren que los datos tengan una distribución normal. La normalización z-score también puede ayudar a reducir la influencia de los valores atípicos en los datos.

Sin embargo, es importante tener en cuenta que la normalización z-score puede no ser adecuada para conjuntos de datos que no tienen una distribución normal o que tienen valores atípicos extremos.

3.1.3. Normalización L1

La normalización L1 es un método de normalización de datos utilizado en aprendizaje automático para ajustar los valores de un conjunto de datos de manera que la suma de los valores absolutos de cada característica sea igual a 1.

El proceso de normalización L1 implica la transformación de cada valor de la característica en una escala basada en la suma de los valores absolutos de esa característica en el conjunto de datos. La fórmula para calcular el valor normalizado de cada valor de característica es la siguiente:

$$x_{normalizados} = \frac{x}{\sum \mid x \mid}$$

Donde x es el valor sin escalar de la característica.

La normalización L1 es útil cuando se desea que los datos sean robustos frente a valores extremos y cuando se desea que los valores de las características estén en la misma escala. La normalización L1 también puede ayudar a reducir la influencia de los valores atípicos en los datos.

Sin embargo, es importante tener en cuenta que la normalización L1 puede no ser adecuada para conjuntos de datos con características altamente correlacionadas, ya que puede conducir a la reducción de la información útil. En esos casos, se pueden utilizar otras técnicas de normalización.

3.2. Métodos para creación de datos sintéticos

3.2.1. SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) es un algoritmo de sobremuestreo sintético utilizado en aprendizaje automático para abordar el problema del desequilibrio de clases en los datos. Este algoritmo genera datos sintéticos de la clase minoritaria mediante la interpolación de los datos existentes de la clase minoritaria. SMOTE funciona mediante la selección de un punto de datos de la clase minoritaria y la creación de nuevos puntos de datos sintéticos que se encuentran a lo largo de las líneas entre el punto seleccionado y sus vecinos más cercanos.

El proceso de SMOTE se realiza en los siguientes pasos:

1. Se selecciona un punto de datos de la clase minoritaria.
2. Selecciona los k vecinos más cercanos del punto seleccionado.
3. Selecciona un vecino aleatorio de los k vecinos más cercanos y calcula la diferencia entre el punto seleccionado y el vecino seleccionado.
4. Multiplica la diferencia por un número aleatorio entre 0 y 1 y agrega el resultado al punto seleccionado para crear un nuevo punto de datos sintético.

El método SMOTE es útil porque ayuda a abordar el problema del desequilibrio de clases al generar datos sintéticos de la clase minoritaria, lo que puede aumentar el tamaño del conjunto de datos de la clase minoritaria y mejorar el rendimiento del modelo. Sin embargo, es importante tener en cuenta que la generación de datos sintéticos puede aumentar el riesgo de sobreajuste del modelo. Además, la selección de un valor óptimo para el parámetro k puede ser crítica para el rendimiento del modelo.

4. Materiales y métodos

Para el desarrollo de esta práctica se utilizó el lenguaje de programación Python en su versión 3.10, con el que se diseño un script para cumplir los objetivos de la práctica.

4.1. Conjunto de datos utilizado

El conjunto de datos utilizado para esta práctica consta de datos tabulares con información clínica para la predicción no invasiva de coledocolitiasis (CBDS, por sus siglas en inglés). Dicho conjunto de datos consta de 19 atributos y 1 variable objetivo, teniendo un total de 293 instancias. El conjunto de datos se encuentra completo, es decir, no hay presencia de datos faltantes.

Los atributos (y variable objetivo) del conjunto de datos se pueden agrupar de la siguiente manera:

- Dicotómicos
  - gender
  - dm
  - ibd
  - cirrhosis
  - stones_on_bd
  - stone_sludge_ercp (variable objetivo)
  - pyobilia_ercp
- Categóricos
  - age_at_ercp
  - race
  - parity
  - intraductal_filling
  - cystic_duct_filling
  - stone_shape_ercp
  - stone_color_ercp
  - gallbladder
- Continuos

- bmi
- peak_bili
- cbd_diameter_us
- cbd_diameter_mrcp
- cbd_diameter_ercp

4.2. Normalización de datos

Para el proceso de normalización de datos se diseñaron diversas funciones en el lenguaje de programación Python, la cuales implementan normalización mediante los métodos min-max, z-score y L1.

Los atributos dicotómicos no se sometieron a proceso de normalización ni codificación, ya que estos se presentaban como 0 y 1 correctamente. Todos los atributos continuos se sometieron a un proceso de normalización mediante el método z-score, debido a que es un método ampliamente utilizado para normalizar datos que serán utilizados dentro de una red neuronal artificial. El resto de atributos, los categóricos, se sometieron a normalización mediante min-max y L1, esto principalmente por fines educativos.

Para evaluar el correcto funcionamiento de los métodos de normalización, se diseñaron funciones que agilizan la obtención y almacenamiento de histogramas. Primero se generaron y almacenaron los histogramas de los datos sin normalizar, posteriormente se aplicaron los métodos de normalización y se volvió a generar y almacenar los histogramas, pero ahora usando los datos normalizados.

4.3. Balance de clases

Para el balance de clases mediante generación de datos sintéticos se diseñaron funciones que permiten implementar 2 métodos de balance de datos: el método de ruleta, y el método SMOTE.

Para evaluar el funcionamiento de estos algoritmos se realizaron 3 pruebas:

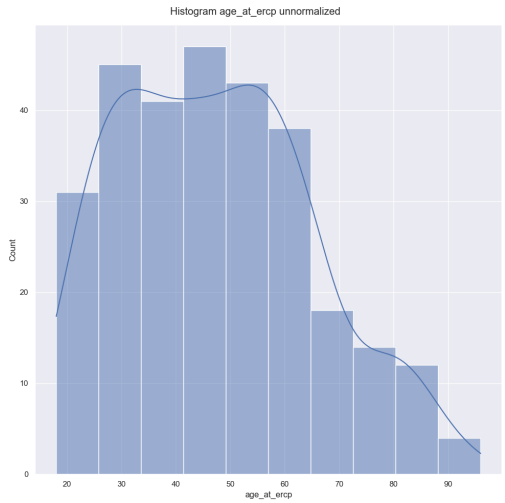
- Se generó una gráfica de distribución de clase: Esto para corroborar que se generaron nuevas instancias de las clase con menor instancias.
- Se obtuvieron los valores de desviación estándar de los datos originales y los datos sintéticos: Esto para tener una idea de las variaciones que presentaban ambos conjuntos de datos.
- Se generaron histogramas de los datos sintéticos: Para comparar la distribución de los nuevos datos respecto a los datos originales.

5. Resultados

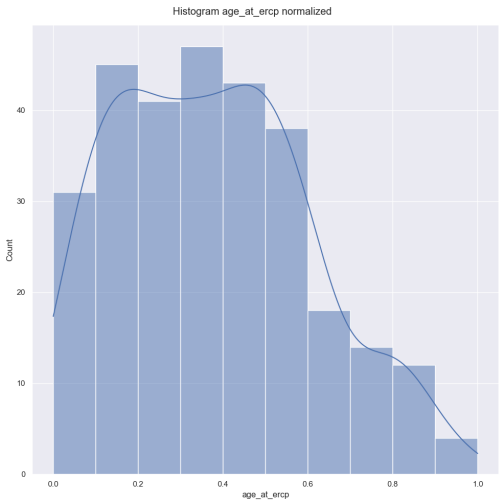
5.1. Normalización de datos

5.1.1. Normalización min-max

La Figuras 1 - 6 muestran los resultados de normalización de los atributos elegidos para el método min-max.

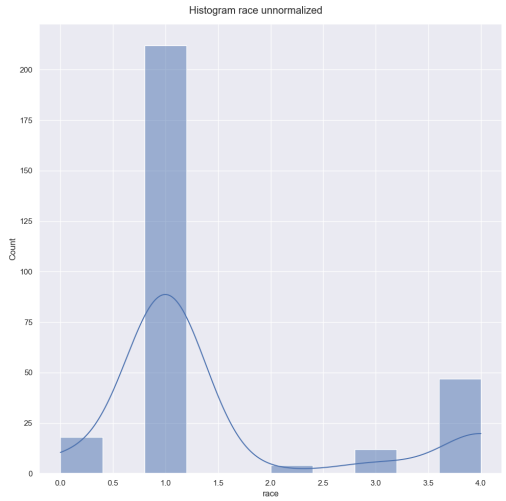


(a) Histograma de atributo *age_at_ercp* sin normalizar.

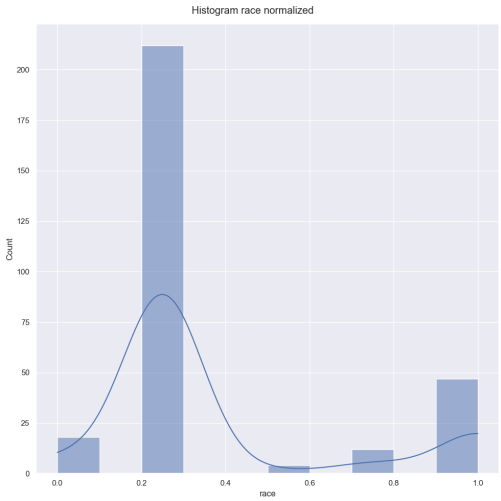


(b) Histograma de atributo *age_at_ercp* posterior a normalización.

Figura 1: Resultados de normalización de atributo *age_at_ercp*

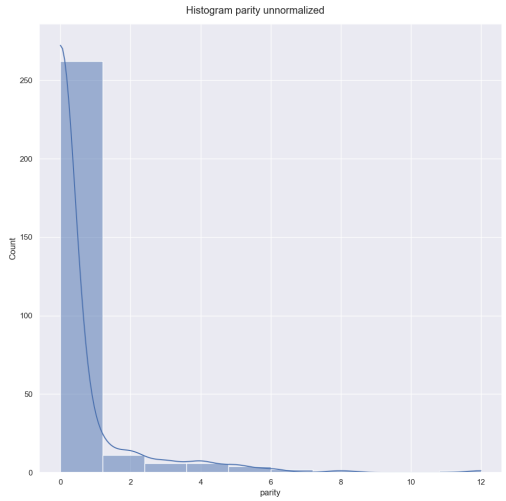


(a) Histograma de atributo *race* sin normalizar.

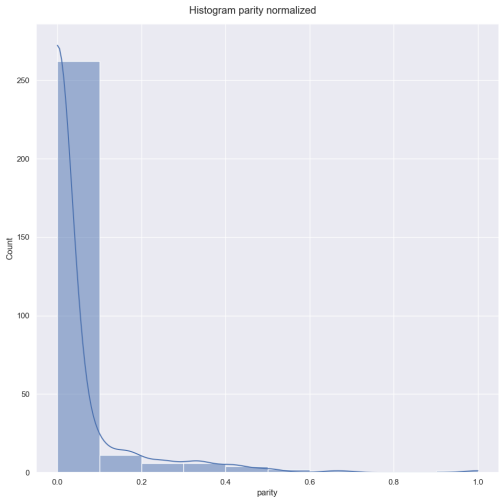


(b) Histograma de atributo *race* posterior a normalización.

Figura 2: Resultados de normalización de atributo *race*

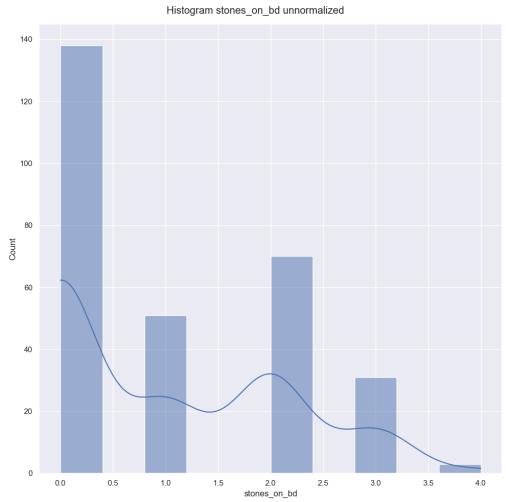


(a) Histograma de atributo *parity* sin normalizar.

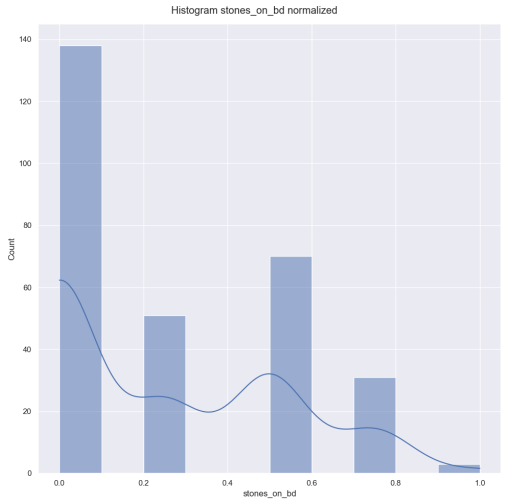


(b) Histograma de atributo *parity* posterior a normalización.

Figura 3: Resultados de normalización de atributo *parity*

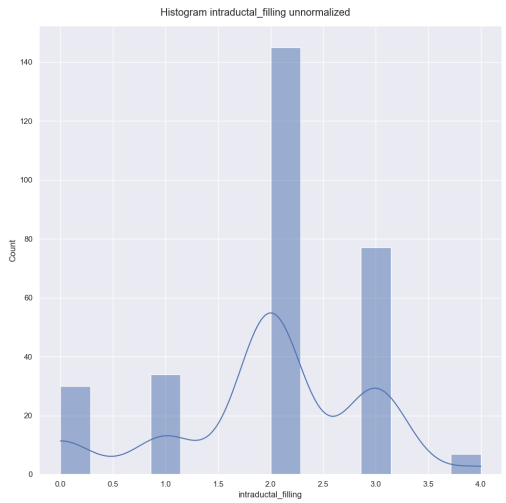


(a) Histograma de atributo *stones_on_bd* sin normalizar.

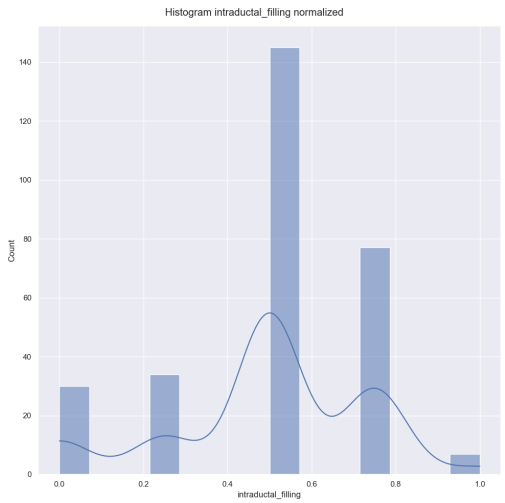


(b) Histograma de atributo *stones_on_bd* posterior a normalización.

Figura 4: Resultados de normalización de atributo *stones_on_bd*

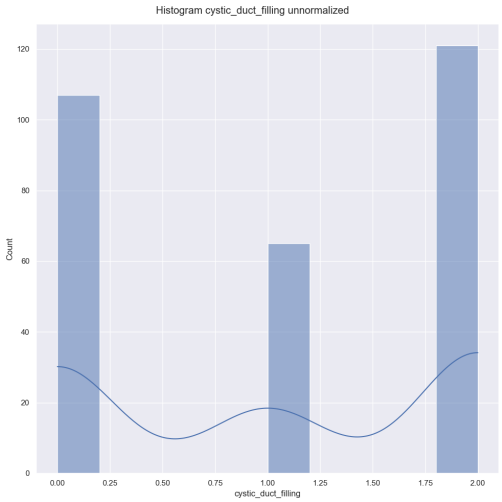


(a) Histograma de atributo *intraductal_filling* sin normalizar.

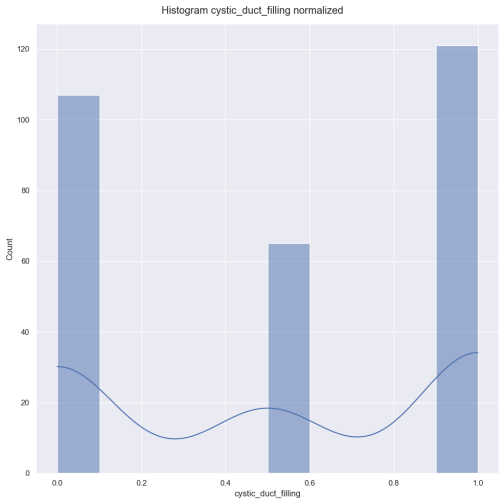


(b) Histograma de atributo *intraductal_filling* posterior a normalización.

Figura 5: Resultados de normalización de atributo *intraductal_filling*



(a) Histograma de atributo *cystic_duct_filling* sin normalizar.

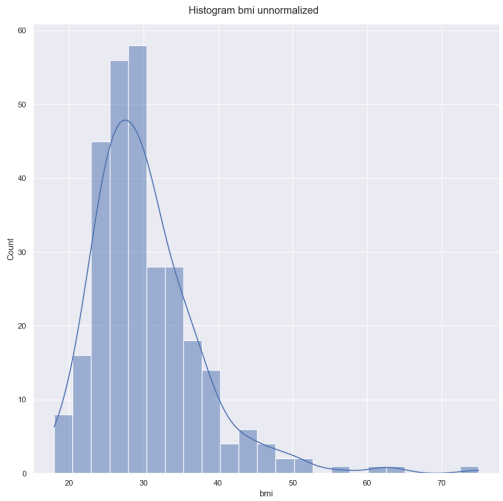


(b) Histograma de atributo *cystic_duct_filling* posterior a normalización.

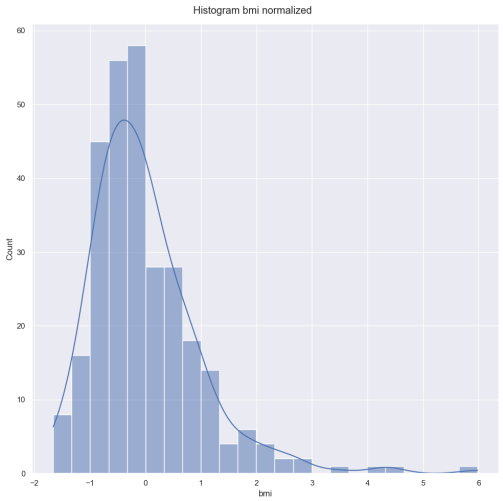
Figura 6: Resultados de normalización de atributo *cystic_duct_filling*

5.1.2. Normalización z-score

La Figuras 7 - 11 muestran los resultados de normalización de los atributos elegidos para el método z-score.

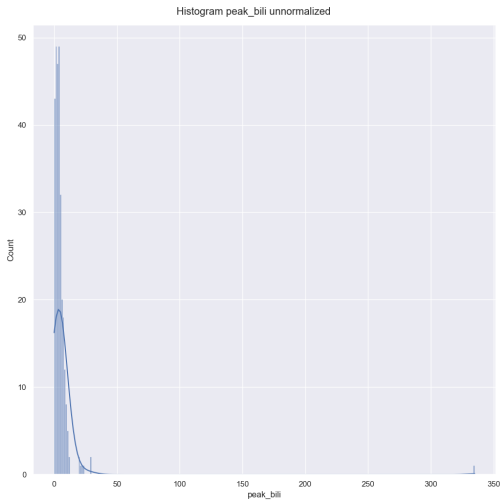


(a) Histograma de atributo *bmi* sin normalizar.

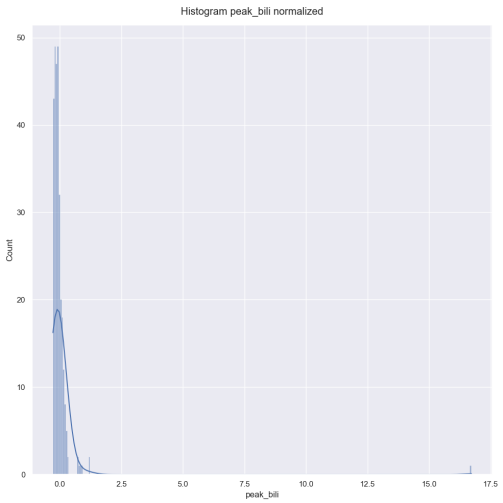


(b) Histograma de atributo *bmi* posterior a normalización.

Figura 7: Resultados de normalización de atributo *bmi*

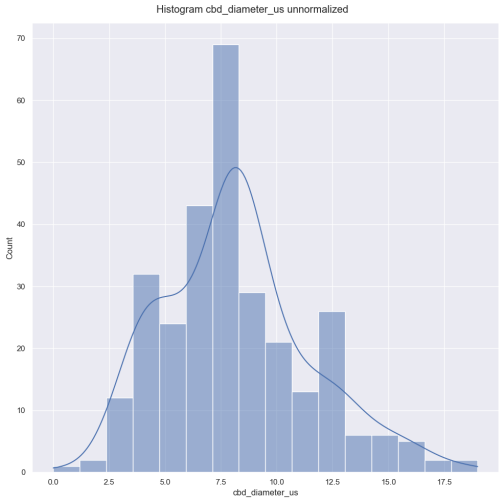


(a) Histograma de atributo *peak_bili* sin normalizar.

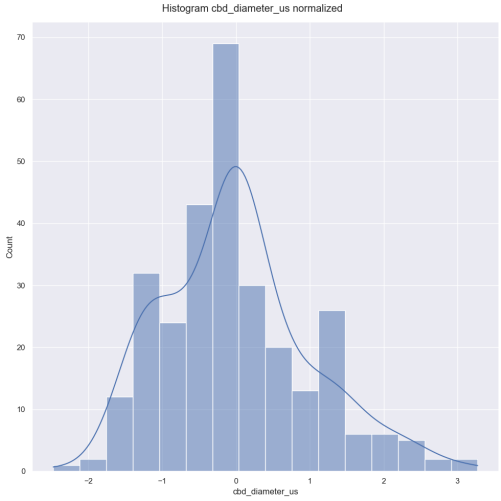


(b) Histograma de atributo *peak_bili* posterior a normalización.

Figura 8: Resultados de normalización de atributo *peak_bili*

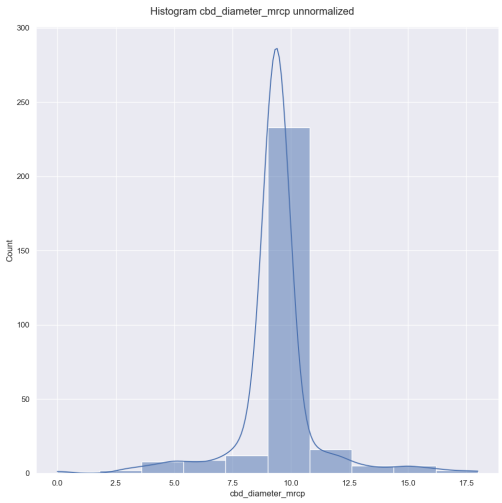


(a) Histograma de atributo *cbd_diameter_us* sin normalizar.

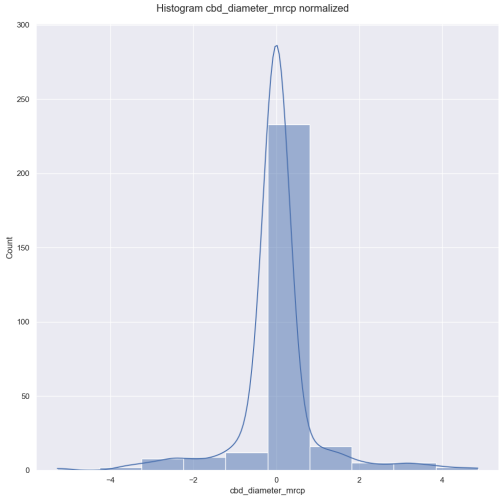


(b) Histograma de atributo *cbd_diameter_us* posterior a normalización.

Figura 9: Resultados de normalización de atributo *cbd_diameter_us*

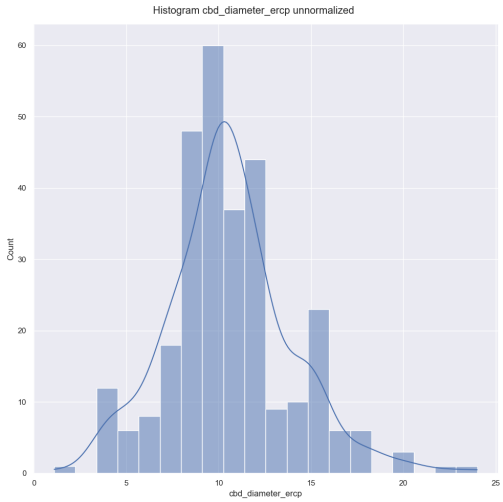


(a) Histograma de atributo *cbd_diameter_mrcp* sin normalizar.

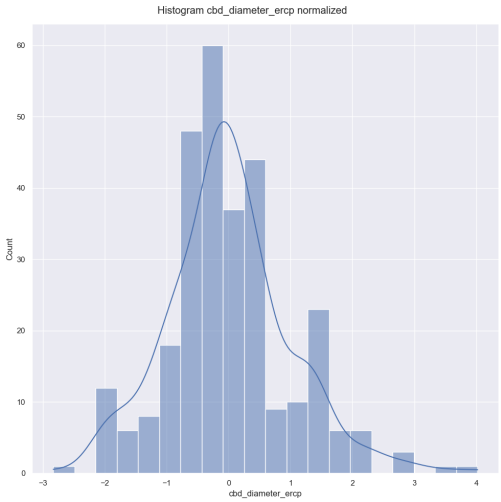


(b) Histograma de atributo *cbd_diameter_mrcp* posterior a normalización.

Figura 10: Resultados de normalización de atributo *cbd_diameter_mrcp*



(a) Histograma de atributo *cbd_diameter_ercp* sin normalizar.

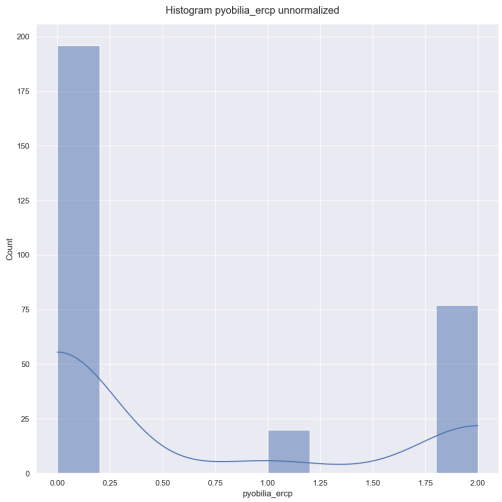


(b) Histograma de atributo *cbd_diameter_ercp* posterior a normalización.

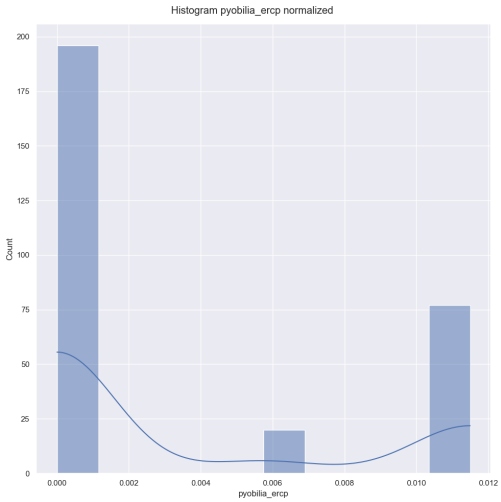
Figura 11: Resultados de normalización de atributo *cbd_diameter_mrcp*

5.1.3. Normalización L1

La Figuras 12 - 14 muestran los resultados de normalización de los atributos elegidos para el método L1.

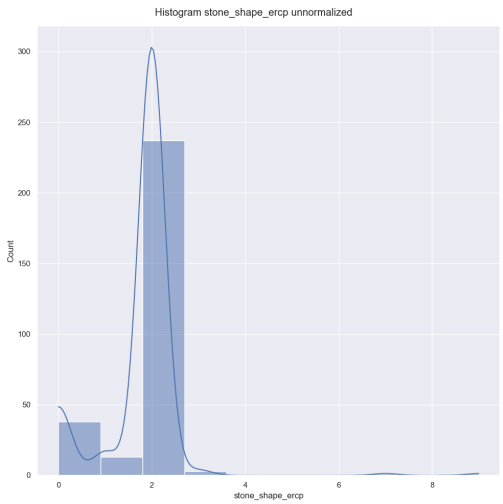


(a) Histograma de atributo *pyobilia_ercp* sin normalizar.

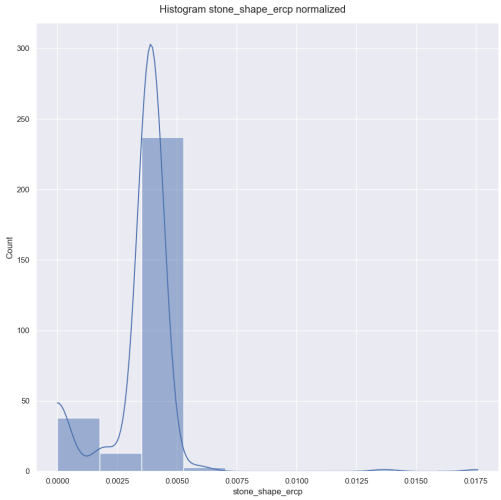


(b) Histograma de atributo *pyobilia_ercp* posterior a normalización.

Figura 12: Resultados de normalización de atributo *pyobilia_ercp*

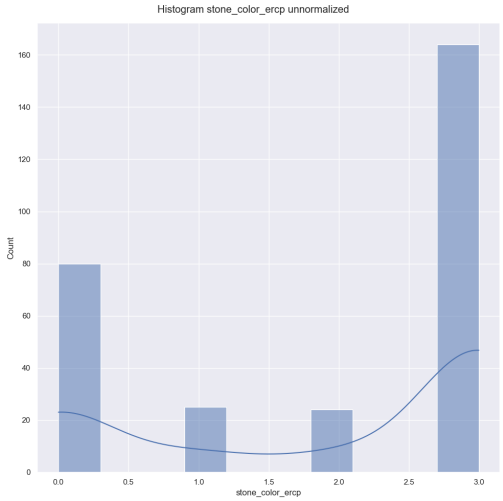


(a) Histograma de atributo *stone_shape_ercp* sin normalizar.

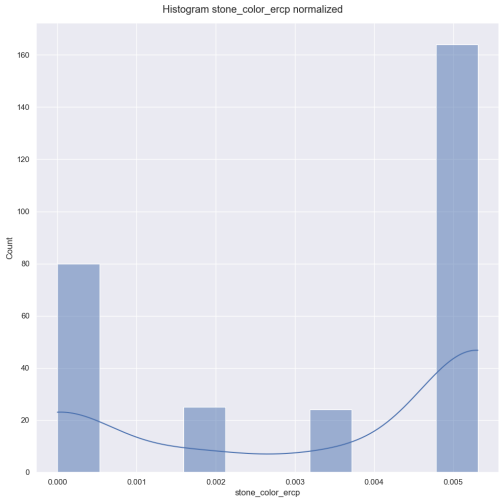


(b) Histograma de atributo *stone_shape_ercp* posterior a normalización.

Figura 13: Resultados de normalización de atributo *stone_shape_ercp*



(a) Histograma de atributo *stone_color_ercp* sin normalizar.



(b) Histograma de atributo *stone_color_ercp* posterior a normalización.

Figura 14: Resultados de normalización de atributo *stone_shape_ercp*

5.2. Balance de clases con datos sintéticos

La Figura 15 muestra la distribución de clases original del conjunto de datos.

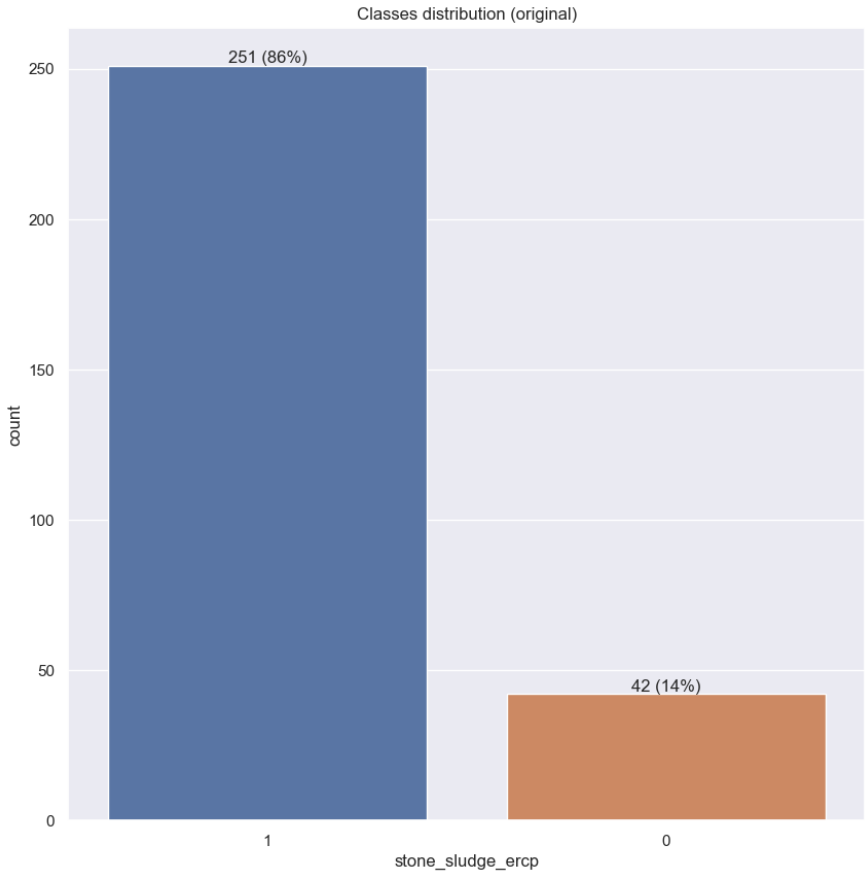


Figura 15: Distribución de clases en el conjunto de datos original

5.2.1. Método de ruleta

La Figura 16 muestra la distribución de clases resultante de aplicar el método de ruleta. La Tabla 1 muestra una comparación entre la desviación estándar de los atributos de la base datos original y los datos obtenidos tras aplicar el método ruleta. Por otro lado, la Figura 17 muestra un ejemplo de la distribución de los datos resultantes del método ruleta.

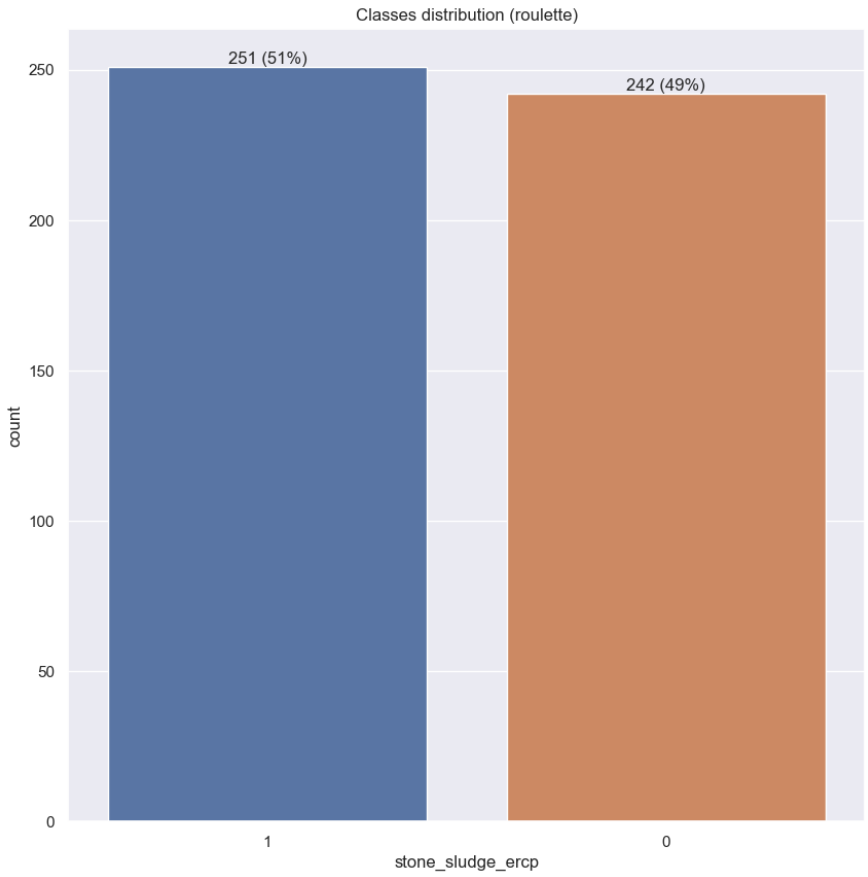
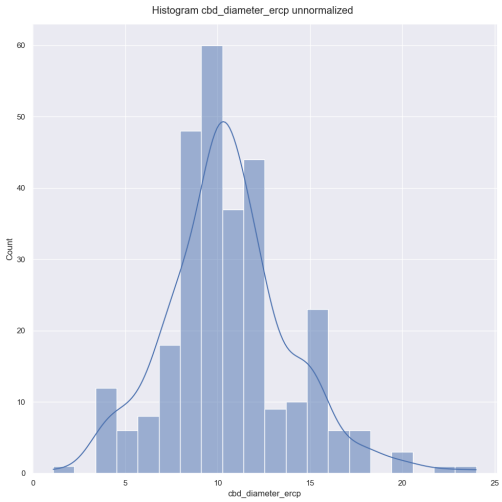


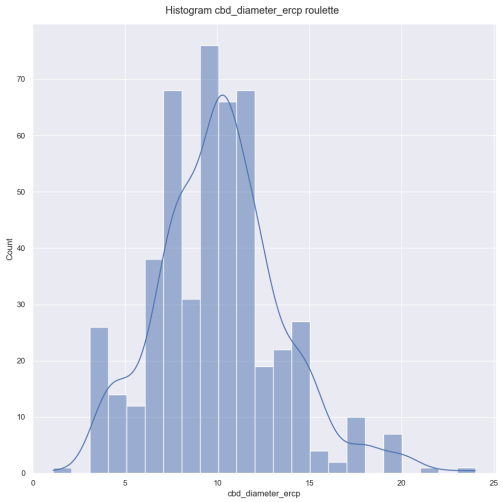
Figura 16: Distribución de clases en el conjunto de datos resultante del método de ruleta

Tabla 1: Comparación de los valores de desviación estándar del conjunto de datos original y posterior al balance de clases por le método de la ruleta.

Atributo	σ	
	Original	**Ruleta**
age_at_ercp	17.81	17.49
gender	0.48	0.49
race	1.19	1.25
bmi	7.47	1.15
parity	1.34	0.46
dm	0.36	0.41
ibd	0.05	0.45
cirrhosis	0.18	0.21
peak_bili	19.73	29.91
stones_on_bd	1.10	1.14
cbd_diameter_us	3.31	3.27
cbd_diameter_mrcp	1.77	1.67
cbd_diameter_ercp	3.34	3.37
intraductal_filling	0.94	1.18
cystic_duct_filling	0.88	0.92
stone_shape_ercp	0.87	0.85
stone_color_ercp	1.31	1.17
pyobilia_ercp	0.87	0.87
gallbladder	0.50	0.53
stone_sludge_ercp	0.35	0.50



(a) Histograma de atributo *cbd_diameter_ercp* original.



(b) Histograma de atributo *cbd_diameter_ercp* posterior a método de ruleta.

Figura 17: Resultados de datos sintéticos mediante el método ruleta

5.2.2. SMOTE

La Figura 18 muestra la distribución de clases resultante de aplicar el método SMOTE. La Tabla 2 muestra una comparación entre la desviación estándar de los atributos de la base datos original y los datos obtenidos tras aplicar el método SMOTE. Por otro lado, la Figura 19 muestra un ejemplo de la distribución de los datos resultantes del método SMOTE.

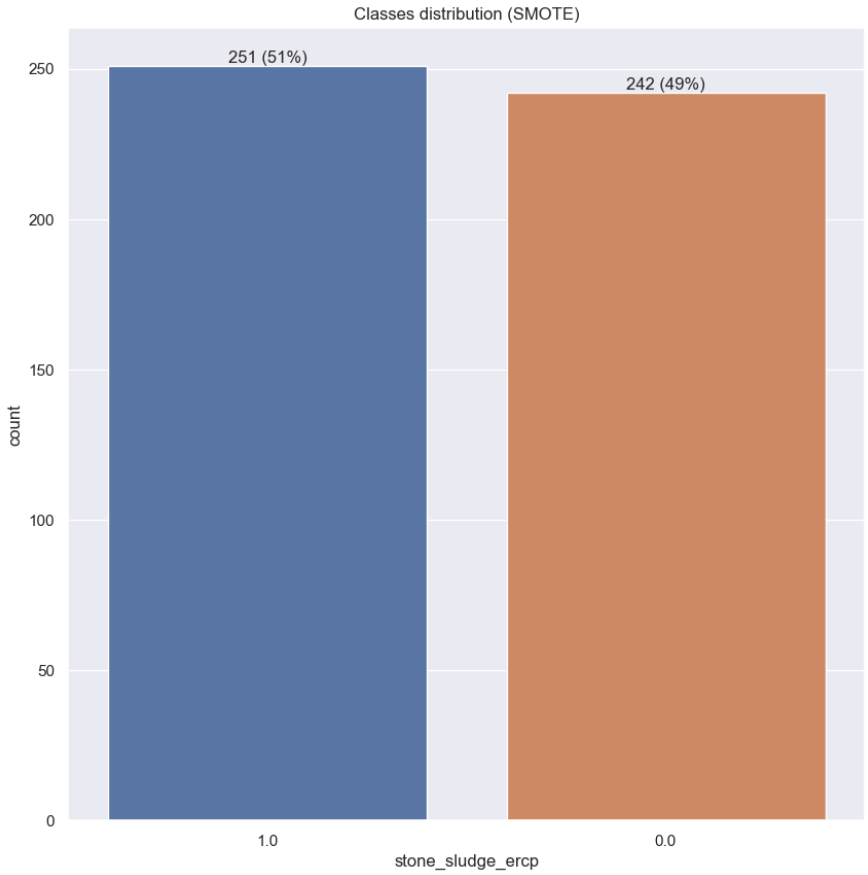
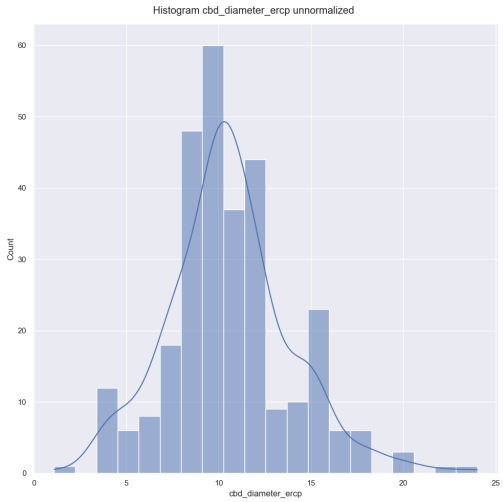


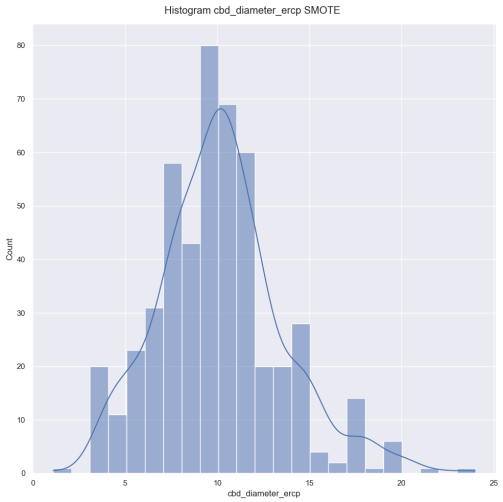
Figura 18: Distribución de clases en el conjunto de datos resultante del método SMOTE

Tabla 2: Comparación de los valores de desviación estándar del conjunto de datos original y posterior al balance de clases por el método SMOTE.

Atributo	σ	
	Original	**SMOTE**
age_at_ercp	17.81	17.07
gender	0.48	0.46
race	1.19	1.18
bmi	7.47	7.50
parity	1.34	1.15
dm	0.36	0.39
ibd	0.05	0.04
cirrhosis	0.18	0.17
peak_bili	19.73	25.85
stones_on_bd	1.10	1.09
cbd_diameter_us	3.31	3.14
cbd_diameter_mrcp	1.77	1.54
cbd_diameter_ercp	3.34	3.37
intraductal_filling	0.94	1.11
cystic_duct_filling	0.88	0.87
stone_shape_ercp	0.87	0.76
stone_color_ercp	1.31	1.15
pyobilia_ercp	0.87	0.84
gallbladder	0.50	0.48
stone_sludge_ercp	0.35	0.50



(a) Histograma de atributo *cbd_diameter_ercp* original.



(b) Histograma de atributo *cbd_diameter_ercp* posterior a método SMOTE.

Figura 19: Resultados de datos sintéticos mediante el método SMOTE

6. Conclusiones

Relacionado a los algoritmos implementados para normalización de datos. Se pudo observar que los 3 algoritmos implementados lograron conservar la distribución de los datos originales, lo cual indica que se han implementado de forma exitosa. Hablando en específico de los datos normalizados mediante z-score, cabe destacar que al aplicarlo a atributos continuos cuyo histograma se asemejaba a una distribución normal, se logró enfatizar este comportamiento, lo cual se espera sea de ayuda cuando se desee trabajar con este conjunto de datos en una red neuronal artificial.

Relacionado a los algoritmos de balance de clases con datos sintéticos. En primer lugar, para ambos algoritmos se logró el objetivo de crear más instancias de la clase con menor, logrando pasar de una relación 85-15 a una relación 51-49, lo cuál es una característica ideal de un conjunto de datos. Al revisar las similitudes entre los valores de desviación estándar de cada atributo en cada conjunto de datos generado, fue posible observar que sus valores eran muy similares, lo cuál nos da buenos indicios de que el proceso de balance de clases se realizó correctamente. Por último, al comparar las gráficas de distribución de los conjuntos de datos generados con el conjunto original se logra observar que presentan diferencias considerables, tanto en forma como en magnitud, lo cuál es un indicio de que podría haberse realizado mal el proceso de balanceo de clases. Cabe destacar que para el proceso de balance de clases solamente se realizó la síntesis de datos para la clase menor, generando un total de 200 instancias completamente nuevas.

En conclusión, las técnicas de normalización de datos son de gran importancia ya que permite adecuar nuestros datos a los requerimientos establecidos por nuestros algoritmos de interés. En esta ocasión se implementaron métodos de normalización que conservan la forma de la distribución original, sin embargo, existen algoritmos que buscan lo contrario, como lo es generar una distribución de tipo uniforme. Es parte esencial del especialista de datos el elegir el método apropiado para su caso de uso. Hablando de los algoritmos de balance de clases mediante datos sintéticos, se logró el objetivo principal del balancear, sin embargo, debido a los resultados obtenidos al comparar las distribuciones, se llegan a presentar duda sobre si el proceso fue realizado correctamente, habrá que investigar más a fondo sobre las técnicas y sobre sus métodos de validación.

Referencias bibliográficas

[1] D. Pyle, *Data Preparation for Data Mining (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1999, ISBN: 9781558605299.

[2] S. Raschka, "About feature scaling and normalization, and the effect of standardization for machine learning algorithms," Sebastian Raschka, 2014, Visitado: Marzo 2023. [En linea]. Disponible: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

[3] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009. [En linea]. Disponible: https://doi.org/10.1109/TKDE.2008.239

[4] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 2002, pp. 321–357, 2002. [En linea]. Disponible: https://doi.org/10.1613/jair.953

[5] M. A. Aceves Fernández, *Inteligencia artificial para programadores con prisa*. Universo de Letras, 2021, ISBN: 9788418854613.

A. Código documentado

El código completo y funcional se puede encontrar anexo en el archivo *zip* compartido en conjunto con este reporte, así como en las secciones anexas.

A.1. Practica3_EnriqueMenaCamilo.py

Script implementado para el desarrollo de la práctica.

```python
#!/usr/bin/env python3
"""Machine learning: Practice 3.
Implement normalization methods and class balancing.
By: Enrique Mena Camilo.
"""

import random
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from typing import Union, Tuple

from collections import Counter

sns.set_theme(style="darkgrid")
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

__author__ = "Enrique Mena Camilo"
__email__ = "enriquemece97@gmail.com"


def get_histogram(feature: Union[np.ndarray, pd.DataFrame, pd.Series], title: str):
    """_summary_

    :param Union[np.ndarray, pd.DataFrame, pd.Series] feature:
    :param str title:
    """
    fig, ax = plt.subplots(1, 1)
    fig.suptitle(f"Histogram {title}")
    fig.set_size_inches(10, 10)

    sns.histplot(feature, ax=ax, kde=True)
    ax.grid(True)
    plt.tight_layout()
    plt.savefig(f"./figures/histogram_{'_'.join(title.lower().split())}.png", bbox_inches="tight")
    plt.show()


def get_classes_distribution(dataset: pd.DataFrame, target_name: str, title: str):
    """

    :param dataset:
    :param target_name:
    :param title:
    :return:
```

```python
    """
    plt.figure(figsize=(10, 10))
    ax = sns.countplot(data=dataset, x=target_name,
                       order=dataset[target_name].value_counts(ascending=False).index)
    abs_values = dataset[target_name].value_counts(ascending=False)
    rel_values = dataset[target_name].value_counts(ascending=False, normalize=True).values * 100
    lbls = [f"{p[0]} ({p[1]:.0f}%)" for p in zip(abs_values, rel_values)]
    _ = ax.bar_label(container=ax.containers[0], labels=lbls)
    _ = ax.set_title(f"Classes distribution ({title})", fontsize=12)
    plt.savefig(f"./figures/classes_distribution_{title.lower()}.png", bbox_inches="tight")
    plt.show()


def min_max_normalization(feature: Union[np.ndarray, pd.DataFrame, pd.Series], desired_range: tuple = (0, 1)) -> np.ndarray:
    """_summary_

    :param Union[np.ndarray, pd.DataFrame, pd.Series] feature: _description_
    :param tuple desired_range: _description_, defaults to (0, 1)
    :return np.ndarray: _description_
    """
    feature_ = feature.copy()
    if type(feature) in [pd.DataFrame, pd.Series]:
        feature_ = feature.to_numpy()

    feature_ = (feature_ - feature_.min(axis=0)) / (feature_.max(axis=0) - feature_.min(axis=0))
    return feature_ * (desired_range[1] - desired_range[0]) + desired_range[0]


def z_score_normalization(feature: Union[np.ndarray, pd.DataFrame, pd.Series]) -> np.ndarray:
    """_summary_

    :param Union[np.ndarray, pd.DataFrame, pd.Series] feature: _description_
    :return np.ndarray: _description_
    """
    feature_ = feature.copy()
    if type(feature) in [pd.DataFrame, pd.Series]:
        feature_ = feature_.to_numpy()

    return (feature_ - feature_.mean(axis=0)) / feature_.std(axis=0)


def l1_normalization(feature: Union[np.ndarray, pd.DataFrame, pd.Series]) -> np.ndarray:
    """_summary_

    :param Union[np.ndarray, pd.DataFrame, pd.Series] feature: _description_
    :return np.ndarray: _description_
    """
    feature_ = feature.copy()
    if type(feature) in [pd.DataFrame, pd.Series]:
        feature_ = feature_.to_numpy()

    norm = np.sum(np.abs(feature_))
    return feature_ / norm


def roulette_class_balancing(dataset: pd.DataFrame, N: int, target_name: str, minor_class: int) -> pd.DataFrame:
    """Generate synthetic data from roulette method.

    :param pd.DataFrame dataset: Dataset to which synthetic data will be added
    :param int N: Number of synthetic instances to generate.
    :param str target_name: Column name of target variable.
```

```
    :param int minor_class: Label of the class with the least amount of data.
    :return pd.DataFrame: Balanced dataset
    """
    ds = []
    data_minor = dataset.copy()
    data_minor = data_minor[data_minor[target_name] == minor_class]
    columns = data_minor.columns.to_list()
    for j in range(N):
        row = []
        for i in range(len(columns)):
            # Unique value
            item_counts_key = data_minor[columns[i]].value_counts().index.tolist()

            # Probability that this value appears in the database
            item_counts = data_minor[columns[i]].value_counts(normalize=True)

            # Cumulative sum of probability to prioritize values
            cumsum_p = item_counts.cumsum()
            w = cumsum_p.to_numpy()

            # Generate a random percentage
            r = random.random()

            # Find the closest percentage achieved by unique values
            magic = np.argwhere(r <= w)

            # Get the index of each value
            x = int(magic[0])

            # Get the value to generate synthetic data
            dato = item_counts_key[x]
            row.append(dato)

        # Add the synthetically created value to the dataset
        ds.append(row)

    # Build new dataset
    d_syn = pd.DataFrame(ds, columns=columns)
    return pd.concat([dataset, d_syn], ignore_index=True)


def smote_class_balancing(
        features: Union[np.ndarray, pd.DataFrame, pd.Series],
        target: Union[np.ndarray, pd.DataFrame, pd.Series],
        minor_class: int, k: int, N: int
) -> Tuple[np.ndarray, np.ndarray]:
    """Generates new synthetic instances of the minority class using the SMOTE algorithm.

    :param Union[np.ndarray, pd.DataFrame, pd.Series] features: Features array with shape (n_samples, n_features).
    :param Union[np.ndarray, pd.DataFrame, pd.Series] target: Target array with shape (n_samples,).
    :param int minor_class: Label of the class with the least amount of data.
    :param int k: Number of nearest neighbors to consider to generate new synthetic instances.
    :param int N: Number of synthetic instances to generate.
    :return Tuple[np.ndarray, np.ndarray]: (Balanced features array, balanced target array).
    """
    features_ = features.copy()
    if type(features) in [pd.DataFrame, pd.Series]:
        features_ = features_.to_numpy()

    # Splits majority and minority class instances
    minority_class = features_[target == minor_class]
```

```
    majority_class = features_[target != minor_class]

    # Initializes the feature array and label vector for the new synthetic instances
    synthetic_features = np.zeros((N, minority_class.shape[1]))
    synthetic_labels = np.ones(N)

    # SMOTE algorithm
    for i in range(N):
        # Select a random instance of the minority class
        j = np.random.choice(minority_class.shape[0])

        # Find the k nearest neighbors of the selected instance in the feature space
        knn_indices = np.argsort(np.sum((minority_class - minority_class[j])**2, axis=1))[:k]

        # Randomly select one of the k nearest neighbors
        m = np.random.choice(knn_indices)

        # Generates a new synthetic instance by interpolating between the selected instance and the selected neighbor
        synthetic_features[i, :] = minority_class[j, :] + np.random.rand() * (minority_class[m, :] - minority_class[j, :])

    # Concatenates the new synthetic instances with the instances of the original minority class
    x_resampled = np.vstack((minority_class, synthetic_features))
    y_resampled = np.concatenate((np.zeros(minority_class.shape[0]), np.zeros(N)))

    # Concatenates the instances of the original majority class
    x_resampled = np.vstack((x_resampled, majority_class))
    y_resampled = np.concatenate((y_resampled, np.ones(majority_class.shape[0])))

    return x_resampled, y_resampled


if __name__ == "__main__":
    data = pd.read_csv("../data/cbds.csv")

    # ==== DATA PREPROCESSING ====
    # Does the dataset have missing data?
    print(data.info())
    # R: No, the dataset is complete

    # Get histograms with unnormalized data
    columns = data.columns.to_list()
    for column in columns:
        get_histogram(data[column], f"{column} unnormalized")

    # ==== DATA NORMALIZATION ====
    data_norm = data.copy()
    # Min-Max normalizations
    features = ["age_at_ercp", "race", "parity", "stones_on_bd", "intraductal_filling", "cystic_duct_filling"]
    for feature in features:
        # The selected range is made in this way because there are no negative observations.
        data_norm[feature] = min_max_normalization(data_norm[[feature]], desired_range=(0, 1))

    # L1 normalizations
    features = ["pyobilia_ercp", "stone_shape_ercp", "stone_color_ercp"]
    for feature in features:
        data_norm[feature] = l1_normalization((data_norm[[feature]]))

    # Z-score normalizations
    features = ["bmi", "peak_bili", "cbd_diameter_us", "cbd_diameter_mrcp", "cbd_diameter_ercp"]
    for feature in features:
        data_norm[feature] = z_score_normalization(data_norm[[feature]])
```

```
232
233     # Get histograms with normalized data
234     columns = data.columns.to_list()
235     for column in columns:
236         get_histogram(data_norm[column], f"{column} normalized")
237
238     # ==== CLASS BALANCING ====
239     target = "stone_sludge_ercp"
240
241     # What do we know about the classes? are they balanced?
242     get_classes_distribution(data, target, "original")
243     # R: We don't have a class balance, we have a ratio close to 85-15
244
245     print("Original standard deviation")
246     print(data.std())
247
248     # Roulette class balancing
249     data_roulette = roulette_class_balancing(data, 200, target, 0)
250     get_classes_distribution(data_roulette, target, "roulette")
251
252     # Get histograms with roulette data
253     columns = data_roulette.columns.to_list()
254     for column in columns:
255         get_histogram(data_roulette[column], f"{column} roulette")
256
257     print("Roulette standard deviation")
258     print(data_roulette.std())
259
260     # SMOTE class balancing
261     x_balanced, y_balanced = smote_class_balancing(data.drop(columns=[target]), data[target], minor_class=0, k=3, N=200)
262     features = data.drop(columns=target).columns.to_list()
263     data_smote = pd.DataFrame(data=x_balanced, columns=features)
264     data_smote[target] = y_balanced
265     get_classes_distribution(data_smote, target, "SMOTE")
266
267     # Get histograms with SMOTE data
268     columns = data_smote.columns.to_list()
269     for column in columns:
270         get_histogram(data_smote[column], f"{column} SMOTE")
271
272     print("SMOTE standard deviation")
273     print(data_smote.std())
```