UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

GENERACIÓN DE DATA SINTÉTICA EN APLICACIONES DE MACHINE LEARNING PARA TELECOMUNICACIONES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

JOAQUÍN ENRIQUE PÉREZ OLIVARES

PROFESOR GUÍA:
ÓSCAR PEREDO ANDRADE

MIEMBROS DE LA COMISIÓN:
ANDRÉS CABA RUTTE
MIGUEL ESPINOZA PEREIRA

SANTIAGO DE CHILE
2024

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: JOAQUÍN ENRIQUE PÉREZ OLIVARES
FECHA: 2024
PROF. GUÍA: ÓSCAR PEREDO ANDRADE

GENERACIÓN DE DATA SINTÉTICA EN APLICACIONES DE MACHINE LEARNING PARA TELECOMUNICACIONES

En el contexto de aprendizaje automático y ciencia de datos, la disponibilidad de grandes volúmenes de datos etiquetados es fundamental para el entrenamiento eficaz de modelos predictivos. Sin embargo, la obtención de datos que sean suficientes y con la calidad adecuada, puede ser costosa y además puede tomar un gran tiempo. Para abordar este problema, se presenta el desarrollo e implementación de un motor de aumento de datos que utiliza una variedad de técnicas avanzadas de machine learning y deep learning, incluidas Redes Generativas Adversarias (GANs).

El motor de aumento de datos propuesto está diseñado para generar datos sintéticos que complementen los conjuntos de datos existentes, mejorando la capacidad de generalización de modelos de aprendizaje automático, incluídos los presentes en ClaroVTR. Este motor se estructura en varios módulos clave: engine.py, que maneja el flujo principal de datos y las operaciones de aumento; gan_wrapper.py, que proporciona una capa de abstracción sobre los modelos GAN; y utils.py, que ofrece funciones auxiliares para la manipulación de datos.

El desarrollo del motor se realizó en varias fases: definición del problema y requisitos, análisis de datos, diseño de la arquitectura del sistema, implementación de los módulos, y validación del motor mediante pruebas. La solución fue evaluada en diversas bases de datos que varían en tamaño y complejidad para determinar su eficacia en diferentes escenarios. Los resultados demuestran que el motor puede generar datos sintéticos que mejoran el rendimiento de los modelos de aprendizaje automático en múltiples casos de uso, aunque con variaciones en la efectividad dependiendo del tipo y tamaño del conjunto de datos. En situaciones en que la data presenta una dimensión muy alta, es posible que al reducir espacios de búsqueda para optimizar tiempos de ejecución, los resultados no mejoren considerablemente, presentando incluso disminuciones del rendimiento.

La implementación se llevó a cabo utilizando bibliotecas de código abierto como numpy, pandas, pytorch y SDV, asegurando la integridad y reproducibilidad del sistema.

A toda mi familia y amigas/os,

Muchas gracias

Agradecimientos

En primer lugar, me gustaría agradecer a mi núcleo familiar, en especial a mis padres, que son los que me ayudaron desde la casa a que todo el proceso universitario lo pudiera llevar a cabo, dando apoyo, recursos, ánimos y preocupación en seguir adelante y cumplir mis objetivos. Siempre recordaré los desayunos y onces especiales que mi mamá me llevaba cuando me veía en el PC muy ocupado y muchos otros gestos que siempre se valorarán. También a mis hermanos mayores, la Alon, siempre preocupada de nuestro bienestar y dando muchos regalitos, y Conra, apoyando desde la distancia allá en el sur.

A toda la familia Olivares, a la *Truda* que aún está con nosotros, a las y los primos, a todos los tíos y tías, al tío Cristian, al tío Guille y la tía Kathy que siempre estuvieron apoyando de alguna u otra manera, muchísimas gracias. Todas las juntas familiares, todo el *lote* como decía el *Trudo*, también sirvieron como apoyo e inspiración a todo el proceso.

No puedo dejar de mencionar a todos los amigos y amigas de la U, que estuvimos juntos desde plan común hasta ahora, que estamos casi todos listos con nuestro proceso. Todos los almuerzos, carretes y juntas que siguen hasta el día de hoy, se agradecen, espero que nunca acaben. Pensar que en esos tiempos llegamos a una final nacional de taca taca con mi partner Joaquín, algo que nunca olvidaré.

A mis amigos del colegio, que estamos juntos desde media hasta hoy, juntos casi todas las semanas a través de discord, jugando valorant y otros juegos, dándome muchos momentos de alegría, a ellos, muchas gracias también.

A mi jefe y profesor guía del proceso de práctica, Óscar, nunca imaginé que iba a encontrar a alguien tan humano y empático, pese a haber sido el primer estudiante en práctica de todo ClaroVTR, cosa que tuvo sus pro y sus contras, el equipo humano siempre estuvo preocupado y ayudaba en las situaciones que yo requería.

Por último, una mención especial a ti, ya sabes, a pesar de que ya no estamos juntos, ese año fue especial, pues estuviste en mi proceso de práctica y fuiste de mucha ayuda, pasé muy buenos momentos y no los olvidaré, gracias.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

En la época digital actual, las telecomunicaciones tienen un rol central en la conectividad global, permitiendo la interconexión instantánea de personas y dispositivos en todo el mundo. Desde el uso de servicios de mensajería hasta servicios de transmisión en alta definición, estas complejas redes logran mantener a toda una sociedad conectada y a los negocios en funcionamiento. Sin embargo, este sistema también presenta grandes desafíos en términos de gestión de datos [1], puesto que el rápido avance tecnológico ha hecho que el almacenamiento, análisis y generación de datos, sean pilares fundamentales para el funcionamiento eficiente de las empresas. En este sentido, los datos se han vuelto un recurso sumamente importante y la tarea actual es saber aprovecharlos para poder tomar decisiones de manera respaldada y que mejoren las operaciones.

Empresas como ClaroVTR, que ofrecen un amplia variedad de servicios de telecomunicaciones en Chile, dependen en gran medida de la información para poder optimizar la calidad de sus servicios. Este tipo de empresas se desenvuelven en un campo en donde existe una elevada cantidad de datos, que pueden ir desde registros de llamadas hasta métricas de rendimientos de red. De esta forma, poder recopilar y analizar datos que sean representativos se ha vuelto una tarea fundamental a la hora de optimizar los servicios.

La gestión de datos de las empresas de telecomunicaciones, se relaciona con la aplicación de inteligencia artificial (IA) utilizando modelos de Machine Learning (ML) para transformar datos en información útil. Estos modelos pueden ser aplicables en varios ámbitos dentro de la estructura y los servicios que brinda la empresa, por ejemplo, optimizar la red al predecir patrones de tráfico, anticipar el abandono de clientes, personalizar ofertas o incluso detectar fraudes. En este sentido, los modelos provenientes del campo de la IA le otorgan a las empresas la posibilidad de tomar decisiones informadas, mejorar la calidad de los servicios y mantenerse competitivas en un entorno que está en constante cambio.

1.2. Descripción del problema

El problema que presentan este tipo de modelos, es que la efectividad depende en gran medida de los datos con los que se les alimenta. En muchas ocasiones, la falta de datos suficientes y variados puede limitar el rendimiento y la capacidad para generalizar a situaciones del mundo real.

ClaroVTR opera en un entorno dinámico, donde el entendimiento de los patrones de consumo, las tendencias de la red y las preferencias de los clientes son cruciales a la hora de ofrecer servicios de buena calidad. Sin embargo, la recopilación de datos que sean representativos y completos, puede resultar desafiante debido al acceso limitado a ciertos tipos de datos o las limitaciones operativas. Estas brechas en los datos pueden dificultar la creación u operación de modelos de ML que sean efectivos y capaces de adaptarse a las cambiantes condiciones de la red y el mercado.

Recopilar y utilizar datos que sean suficientes y diversos para entrenar o mantener modelos de ML robustos, puede resultar en un obstáculo, pues los modelos requieren conjuntos de datos amplios y representativos para que puedan tener precisión y eficacia en los resultados.

1.3. Objetivos

Con el fin de abordar las limitaciones de disponibilidad y diversidad de datos que puede tener ClaroVTR, las técnicas de generación sintética de datos puede ser una solución innovadora. Estas técnicas implican la creación de datos artificiales que siguen patrones y distribuciones similares a los datos reales. Así, al integrar estos datos sintéticos con los datos reales existentes, es posible expandir y mejorar los conjuntos de datos de entrenamiento de los modelos de ML existentes.

1.3.1. Objetivo general

- Diseñar e implementar una aplicación que sea capaz de recibir datasets tabulares multivariados y retorne un dataset aumentado con un mayor número de registros y las mismas columnas. Los nuevos registros deben ser estadísticamente similares a los registros originales.

1.3.2. Objetivos específicos

- Realizar una revisión bibliográfica de técnicas de generación de data aumentada para data tabular. Seleccionar las técnicas más relevantes según la literatura y al menos 1 que pertenezca al mundo de la inteligencia artificial generativa (Generative AI).
- Realizar experimentos que comparen performance entre las técnicas tradicionales y las técnicas del campo de Generative AI.
- Desarrollar una metodología de comparación entre las distintas técnicas de generación de data sintética.
- Desarrollar un programa (Python) que permita utilizar las técnicas de mejor performance.

Capítulo 2

Marco Teórico y Estado del Arte

2.1. Marco Teórico

2.1.1. Inteligencia Artificial

La inteligencia artificial (IA) se puede pensar como la ciencia que incorpora conocimiento a los procesos para que funcionen más eficazmente [2]. Los modelos de IA por su parte, son sistemas computacionales que imitan la capacidad humana de aprender y tomar decisiones. En este tipo de modelos se busca aprender a realizar una tarea específica a partir de datos proporcionados.

Existen diversas áreas en las que la IA se utiliza en mayor o menor medida. Por ejemplo, en el tratamiento de lenguaje natural, en robótica, en visión computacional, procesamiento de audio, sistemas expertos, etc. Dentro de estas áreas, un aspecto fundamental de muchas tareas es la clasificación, donde la entrada de datos tiene asignada una clase o etiqueta.

2.1.1.1. Machine learning

El aprendizaje automático es un subcampo de la inteligencia artificial que se encarga de desarrollar algoritmos y modelos que permiten aprender patrones a partir de datos. En lugar de que una persona programe de manera explícita todas las reglas y lógica para realizar una tarea específica, el aprendizaje automático permite que las máquinas aprendan de manera automática y mejoren su rendimiento con la experiencia. Los algoritmos construyen un modelo matemático a partir de los datos de entrada, con el propósito de poder tomar una decisión según la tarea que se desea resolver. Es común que estos algoritmos se dividan en dos categorías principales: aprendizaje supervisado y aprendizaje no supervisado [3].

El aprendizaje supervisado utiliza un conjunto de datos etiquetados, en donde cada registro tiene una salida objetivo deseada para que el modelo aprenda a tomar decisiones. El modelo adquiere conocimiento al entender las conexiones o patrones que existen entre los datos de entrada y la salida que se busca obtener. El aprendizaje no supervisado no cuenta con etiquetas en los registros, por lo que debe encontrar patrones en los datos y lograr estructurarlos por sí mismo.

Para resolver ciertas tareas que requieren el uso de machine learning, ClaroVTR utiliza mo-

delos basados en *Boosting*, que es una técnica de ensamblado que combina varios modelos de aprendizaje débil (rendimiento ligeramente superior al alzar) para formar un modelo fuerte [4]. En términos simples, el *Boosting* se traduce en mejorar gradualmente la combinación de modelos más débiles, dando énfasis a instancias que fueron mal clasificadas por modelos anteriores, para finalmente formar un modelo fuerte que es una combinación ponderada de la secuencia de modelos débiles. Algunos de los algoritmos de *Boosting* más conocidos incluyen AdaBoost, XGBoost, LightGBM y CatBoost.

2.1.1.1.1. AdaBoost

Este algoritmo se basa en la idea de asignar pesos a las instancias de datos y construir modelos secuenciales que se centran en corregir los errores cometidos por los modelos anteriores. Se da mayor énfasis a las instancias que han sido clasificadas incorrectamente. Tiene la ventaja de que es adaptable a diferentes tipos de modelos débiles y no requiere ajuste manual de hiperparámetros.

2.1.1.1.2. XGBoost

Este algoritmo es muy usado en conjuntos de datos grandes, ya que es altamente eficiente y escalable. Usa técnicas como el early stopping y la regularización (estrategias efectivas para mejorar la capacidad de generalización de modelos de machine learning) de tal forma de evitar el sobreajuste y mejorar el rendimiento. Una de las ventajas que tiene es que puede manejar de manera automática los valores faltantes en los datos, por lo que facilita el preprocesamiento.

2.1.1.1.3. LightBoost

Este algoritmo acelera el proceso de entrenamiento a través de la técnica de *binning* que divide los datos de manera eficiente. También es altamente efectivo en conjuntos de datos grandes.

2.1.1.1.4. CatBoost

Catboost destaca por su capacidad para manejar de manera automática variables categóricas (no numéricas), eliminado la necesidad de la codificación manual y por ende trabajo de preprocesamiento. Es robusto frente al sobreajuste y tiene buen rendimiento sin la necesidad de realizar ajustes profundos de hiperparámetros.

2.1.1.2. Deep learning

El aprendizaje profundo o deep learning, es un subcampo de la inteligencia artificial que a través de la construcción y entrenamiento de modelos basados en redes neuronales, logra aprender tareas específicas a partir de datos [5]. Los modelos son capaces de aprender características abstractas y de alto nivel, sin la necesidad de una programación o diseño explícito para abordar la tarea. El deep learning ha sido aplicado en varios campos, como el procesamiento de lenguaje natural, el procesamiento de la voz, la visión por computadora, etc., logrando resultados comparables o incluso mejores a lo que obtendrían los expertos en cada campo.

La unidad básica de una red neuronal es el perceptrón, que es un modelo matemático inspirado en la estructura y las funciones de una neurona biológica. Al realizar conexiones entre perceptrones se forma una red neuronal, que a su vez, se logra conectar a otras redes neuronales formando una estructura de capas que dan origen a una red neuronal profunda.

Existen varios tipos de redes neuronales profundas (Deep Neural Network, DNN), cada una diseñada para abordar diferentes tipos de problemas y estructuras de datos. En general, las redes tienen múltiples capas ocultas entre la capa de entrada y la capa de salida [6], lo que les permite aprender representaciones jerárquicas y características complejas de los datos de entrada.

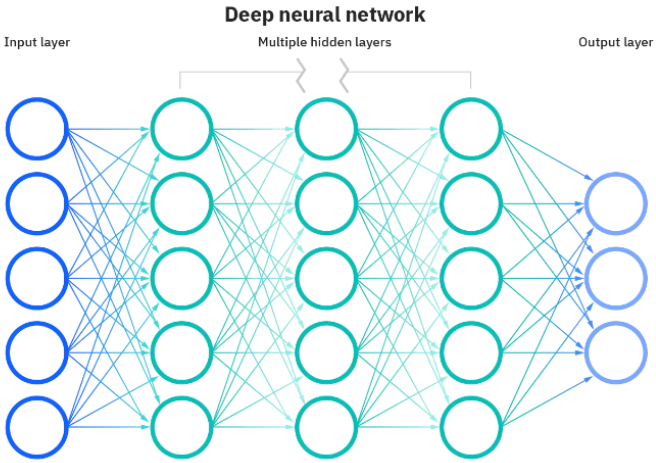

Figura 2.1: Diagrama general de una DNN [7].

La imagen de la figura 2.1 muestra la manera más general de representar un DNN, en donde cada nodo de una capa de salida se conecta directamente a un nodo de la capa anterior, formando capas completamente conectadas o fully-connected. Cuando en una red la información fluye directamente desde la capa de entrada hacia la capa de salida, tal como se muestra en en la imagen, se le denomina como red feed-forward.

2.1.1.3. Clases en un Modelo de Inteligencia Artificial

En clasificación los modelos de ML aprenden a realizar las tareas a partir de un conjunto de datos de entrenamiento que contiene ejemplos previamente etiquetados. Cada ejemplo en el conjunto de datos pertenece a una de las clases. La función del modelo entonces, es poder asignar una clase a un ejemplo que no contiene una etiqueta previa. Las clases se refieren a las categorías en las que se clasifican los datos. Por ejemplo, en un problema de clasificación de animales, las clases podrían ser León o Tigre.

En muchos conjuntos de datos del mundo real, existe un desequilibrio en la distribución de las clases. Este desequilibrio se produce cuando las clases no están representadas de manera equitativa, es decir, existe una gran diferencia en el número de ejemplos entre la clase mayoritaria y la clase minoritaria. Esto puede ser problemático, ya que los modelos de inteligencia artificial tienden a aprender mejor las clases mayoritarias y pueden tener dificultades para clasificar correctamente las clases minoritarias [8], resultando en un modelo sesgado y que toma decisiones inexactas. Esto podría ser crítico en el contexto en que la clase minoritaria sea importante, como por ejemplo, en diagnósticos médicos.

Actualmente, muchos de los problemas de la IA surgen a raíz de tener datos insuficientes, que estén sin etiquetar o que sean de mala calidad. Dado esto, existe un alto interés en la

utilidad que tienen los datos sintéticos y los problemas que se pueden superar. En el caso específico de desequilibrio de clases, se utilizan técnicas de preprocesamiento de datos, como el sobremuestreo de la clase minoritaria o el submuestreo de la clase mayoritaria. También se pueden ajustar los pesos de las clases durante el entrenamiento del modelo, o aplicar técnicas de generación de características para mejorar la discriminación entre las clases. Estas estrategias buscan mejorar la capacidad del modelo para clasificar con precisión todas las clases, independientemente de su distribución en el conjunto de datos. Para entrar en más detalle, a continuación se presentan los distintos métodos de generación sintética de datos.

2.1.2. Métodos de generación de muestras sintéticas

Los datos sintéticos se generan artificialmente mediante algoritmos informáticos y tienen las mismas propiedades estadísticas que los datos reales. Estos datos se pueden generar a partir de los mismos datos reales, de modelos existentes, utilizando conocimiento experto del dominio del problema o a partir de una combinación de ambas opciones. Para obtener muestras sintéticas se utilizan modelos que capturan las propiedades de los datos reales (por ejemplo, la distribución), de tal forma que cuando el modelo esté creado, se puedan muestrear datos sintéticos.

Las muestras sintéticas a partir de modelos existentes son generadas por modelos estadísticos, que son modelos matemáticos que tienen suposiciones estadísticas acerca de cómo se generan los datos, según se describe en *Survey on Synthetic Data Generation, Evaluation Methods and GANs* de Alvaro Figueira and Bruno Vaz [9]. En adelante, esta obra será referida como *Estudio de Síntesis*. Como se mencionó, el conocimiento experto también se puede utilizar para generar datos sintéticos, el problema es que se requiere un amplio conocimiento del dominio del problema para poder generar datos similares a los reales.

En [10] se analizan tres direcciones para el uso de datos sintéticos en aprendizaje automático. Primero se habla sobre el uso de datos sintéticos para entrenar modelos de machine learning y usarlos para hacer predicciones en datos del mundo real. También se habla de aumentar conjuntos de datos reales existentes, para poder cubrir partes subrepresentadas de los datos. Y finalmente, se tocan temas sobre problemas de privacidad y legalidad en la generación de datos anonimizados. En este trabajo, el enfoque está en utilizar datos sintéticos tabulares para mejorar el rendimiento de modelos de aprendizaje automático.

En las siguientes subsecciones, se expondrán las técnicas más comunes en la generación sintética de datos. Existen métodos estándar y métodos que utilizan el aprendizaje profundo. Esta división sigue un enfoque similar al presentado en el *Estudio de Síntesis*, pues se considera que ayudará a organizar y entender mejor cada una de las técnicas.

2.1.2.1. Métodos estándar

Estos métodos eran los más comunes antes del surgimiento de los modelos generativos de aprendizaje profundo. El orden está definido a partir de la complejidad del algoritmo utilizado.

2.1.2.1.1. Random Oversampling

El objetivo principal es aumentar el número de ejemplos de la clase minoritaria al replicar instancias de esa clase de manera aleatoria. En otras palabras, el Random Oversampling crea copias adicionales de las instancias de la clase minoritaria para igualar o acercar la proporción de ejemplos entre la clase minoritaria y la clase mayoritaria. Esto se hace de manera aleatoria y puede resultar en un conjunto de datos balanceado donde ambas clases tienen una representación similar.

Es el método más simple para aumentar un conjunto de datos, el problema está en que este enfoque puede cambiar la distribución de los datos, provocando por ejemplo, que un modelo aprenda una distribución incorrecta. Además, este modelo no crea nuevas muestras sintéticas, sino que simplemente replica las existentes.

2.1.2.1.2. SMOTE

Es un enfoque de sobremuestreo, pero en este caso no se duplican las muestras como en la técnica anterior, sino que se contrarresta el desequilibrio de clases generando observaciones sintéticas a partir de las clases minoritarias [11]. El proceso de SMOTE se inicia seleccionando aleatoriamente una instancia de la clase minoritaria en el conjunto de datos. Luego, se elige un número específico de vecinos cercanos de la misma clase minoritaria. La cantidad de vecinos se controla mediante un hiperparámetro llamado "k".

Una vez que se han seleccionado los vecinos, SMOTE crea ejemplos sintéticos al combinar características de la instancia original con las de sus vecinos. La idea es generar ejemplos que estén situados a lo largo de las líneas que conectan la instancia original con los vecinos. Para conseguir esto, el algoritmo calcula la diferencia entre las características de la instancia original y sus vecinos, multiplica esta diferencia por un número aleatorio entre 0 y 1, y suma el resultado a la instancia original. Este proceso de generación de ejemplos sintéticos se repite para múltiples instancias de la clase minoritaria. La repetición puede detenerse cuando se ha alcanzado un nivel deseado de equilibrio entre las clases o cuando se ha generado un número específico de ejemplos sintéticos.

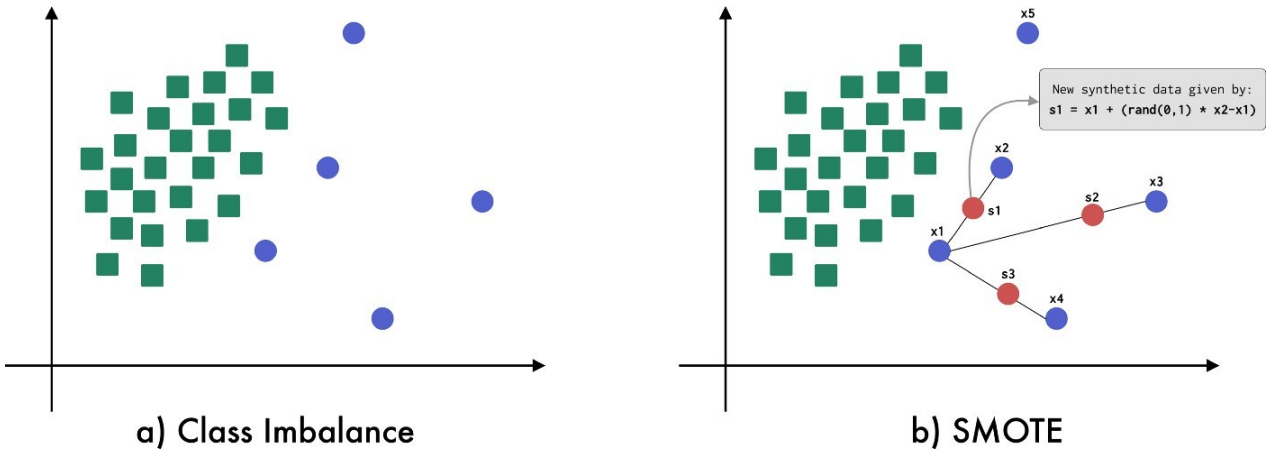

Figura 2.2: Descripción visual SMOTE [12].

Si bien es cierto, SMOTE presenta un avance en comparación a random oversampling, aún

tiene algunos problemas. El algoritmo ignora la clase mayoritaria y se centra únicamente en la clase minoritaria, y además, altera la verdadera distribución de los datos.

2.1.2.1.3. Borderline-SMOTE

Existen dos algoritmos que se han propuesto en [13] como una variación de SMOTE. En el primero, BorderlineSMOTE1, se sobremuestrean puntos de la clase minoritaria que se encuentran cerca de los límites o frontera de decisión, vale decir, se toman puntos o ejemplos de la clase minoritaria que tienen un número de vecinos de clase mayoritaria. Esos son los puntos que tienden a clasificarse erróneamente, pues son los puntos límites dentro de toda la clase minoritaria. Una vez que se detectan esas observaciones, se aplica SMOTE (generar muestras sintéticas a partir de los ejemplos cerca de la frontera).

En BorderlineSMOTE2, se tiene un proceso muy similar, pero se considera a los vecinos de la clase mayoritaria. Primero se identifican esos puntos cerca de la frontera de decisión, al igual que con el primer algoritmo, pero las muestras sintéticas son generadas no sólo por puntos de la clase minoritaria sino que también se toman en cuenta puntos de la clase mayoritaria.

2.1.2.1.4. Safe-Level-SMOTE

Lo que hace SMOTE es muestrear la clase minoritaria a lo largo de una línea que conecta la clase minoritaria con sus vecinos más cercanos, ignorando la clase mayoritaria. En [14], se definen regiones seguras de tal manera de evitar el sobremuestreo en regiones superpuestas, lo que proporciona un mejor rendimiento de precisión que el SMOTE original.

En el algoritmo, para cada ejemplo de la clase minoritaria se calcula un nivel de seguridad definido por la cantidad de vecinos cercanos que también pertenecen a la clase minoritaria. Ese número se calcula usando un valor k, que determina cuántos vecinos se toman en cuenta, y lo define el usuario.

Luego, al generar nuevas instancias sintéticas, el algoritmo se asegura de que esas muestras se coloquen en las zonas más seguras. Es decir, las nuevas muestras se colocan más cerca de los ejemplos minoritarios que tienen un mayor nivel de seguridad. Por ejemplo, si se tiene un punto p de la clase minoritaria, y su vecino más cercano, n (que también pertenece a la clase minoritaria), el algoritmo creará una nueva muestra más cerca de p si p tiene más vecinos minoritarios que n. En caso contrario, la nueva muestra se colocará más cerca de n.

2.1.2.1.5. ADASYN

ADASYN [15] al igual que las técnicas anteriores, es un algoritmo de sobremuestreo que mejora el rendimiento de los clasificadores. La particularidad está en que utiliza una distribución ponderada para diferentes instancias de clases minoritarias, teniendo en cuenta su nivel de dificultad para que un modelo clasificador las aprenda, esto es porque las muestras de clases minoritarias son más difíciles de aprender que aquellas que tienen más vecinos de la misma clase [9]. Lo anterior provoca que se generen más muestras sintéticas para los ejemplos de las clases minoritarias que son más difíciles de aprender y menos para los ejemplos de clases minoritarias que son más fáciles de aprender.

Esta técnica es similar a SMOTE en cuanto a la generación de muestras sintéticas en los segmentos de línea entre dos puntos de datos de la clase minoritaria, la diferencia radica en que se utiliza una distribución de densidad como criterio para determinar automáticamente

la cantidad de muestras sintéticas que se van a generar para cada instancia de la clase minoritaria.

2.1.2.1.6. K-Means SMOTE

Un camino distinto a las técnicas anteriores, es no modificar el algoritmo SMOTE, sino que utilizar un algoritmo no supervisado antes de aplicar SMOTE. En [16] se propone la combinación de K-Means, que es un algoritmo de agrupamiento, con SMOTE, evitando así la generación de ruido y superando los desequilibrios dentro y entre las clases.

En este algoritmo, primero se aplica K-Means para agrupar los datos, identificando clusters donde las instancias de la clase minoritaria están concentradas. Después, sigue un paso de filtrado en donde se descartan los clusters que tienen una muy baja proporción de instancias de la clase minoritaria, de tal forma de evitar generar ruido en áreas donde la clase minoritaria es escasa. Como último paso, se aplica SMOTE a cada uno de los clusters restantes, asignando un mayor número de instancias a los clusters que tienen una menor proporción de muestras de la clase minoritaria.

2.1.2.1.7. Cluster-Based Oversampling

En [17] se propone un algoritmo de sobremuestreo basado en clusters, en donde a cada clase por separado se le aplica algún algoritmo de clustering, y luego se aplica random oversampling a cada clúster. Para los clusters de la clase mayoritaria, todos excepto el clúster más grande, se sobremuestran aleatoriamente hasta que todos tengan el mismo número de instancias que el clúster mayoritario con más datos. Los clusters de las clases minoritarias se sobremuestran aleatoriamente hasta que cada clúster tenga un número determinado de instancias igual a $\frac{m}{N}$, donde m es el número total de instancias de la clase mayoritaria (después del sobremuestreo) y N es el número de clusters de la clase minoritaria.

Este enfoque es muy similar a K-Means, pero se diferencian en que el algoritmo de clustering es de elección libre, y el proceso de clustering se realiza a cada clase por separado. Además, la técnica de sobremuestreo utilizada es random oversampling y no SMOTE.

2.1.2.1.8. Modelo de Mezcla Gaussiana

La mayoría de las técnicas anteriores tienden a descuidar la distribución de los datos originales, por esto surge la necesidad de modelar la distribución de los datos subyacentes y extraer una muestra de ella. El problema es que estimar esa distribución es un problema muy complejo. En este sentido, surge el modelo de mezcla gaussiana (GMM), el cual es un modelo probabilístico que supone que los datos pueden modelarse mediante una suma ponderada de un número finito de distribuciones gaussianas [18].

$$p(x) = \pi_1 p_1(x) + \pi_2 p_2(x) + ... + \pi_n p_n(x) \tag{2.1}$$

La ecuación 2.1 muestra el modelo resultante, en donde $p_i(x)$ es la función de densidad de probabilidad de una distribución normal univariada de media μ_i y desviación estándar σ_i, con π_i el peso asignado a cada $p_i(x)$ y n el número de componentes.

Un ejemplo de uso de las GMM puede ser encontrado en [19], donde los autores abordan el problema de falta de datos en entornos virtuales (IVEs). Primero se realiza una etapa de ajuste del GMM a los datos reales, determinando el número óptimo de distribuciones gaussianas que mejor se ajustan a los datos y estimando los parámetros de cada distribución

gaussiana que componen al GMM. Luego, sigue la etapa de generación de muestras en donde se selecciona aleatoriamente una de las distribuciones gaussianas del GMM, de acuerdo con sus pesos, y se genera un nuevo punto de datos a partir de la distribución gaussiana seleccionada. Esto implica muestrear de una distribución normal multivariada con la media y matriz de covarianza correspondientes a la componente seleccionada. Se repiten los pasos anteriores tantas veces como muestras sintéticas se deseen generar.

2.1.2.2. Métodos de aprendizaje profundo

A continuación, se presentan los tipos más comunes de redes neuronales profundas que se utilizan en los métodos de generación sintética de datos.

2.1.2.2.1. Redes Neuronales Convolucionales

Las CNN o Redes Neuronales Convolucionales, son un tipo de redes neuronales especialmente diseñadas para el procesamiento de datos estructurados en forma de cuadrícula. Tal como se explica en [20], las CNN se basan en el concepto de convolución, que es una operación matemática que combina una ventana deslizante con una matriz de entrada para extraer características locales. Esa ventana deslizante se lo conoce comúnmente como núcleo o kernel, que es una matriz con determinados operadores. Estas redes utilizan capas convolucionales para aplicar los núcleos o filtros convolucionales a los datos de entrada y extraer características espaciales, como los bordes, las texturas y patrones relevantes. Cada núcleo que aplica la convolución tiene como salida un mapa de características.

Generalmente, después de un determinado número de capas convolucionales hay una capa de agrupamiento o pooling, que se encarga de sub-mapear la dimensionalidad, teniendo como efecto la reducción de la dimensionalidad y el resalte de las características más relevantes. Existen distintas formas de aplicar el pooling, las más comunes son el max-pooling y el average-pooling. En max-pooling se toma el máximo de los elementos de un determinado mapa de características y en average-pooling se toma el promedio de los elementos.

Para terminar el flujo de la red neuronal convolucional, después de las etapas de capas convolucionales y pooling, el mapa de características resultante se ingresa a una red neuronal formada por múltiples capas completamente conectadas, o fully-connected.

2.1.2.2.2. Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNN, Recurrent Neuroral Network) son un tipo de redes neuronales que se especializan en tratar con datos secuenciales. El adjetivo recurrente, viene del hecho de que presentan un loop de retroalimentación, en donde se realiza la misma tarea para cada elemento de una secuencia, lo que les permite tener memoria y procesar secuencias de datos. Esto hace que estas redes tengan la capacidad de capturar dependencias temporales y modelar relaciones contextuales a lo largo del tiempo [21].

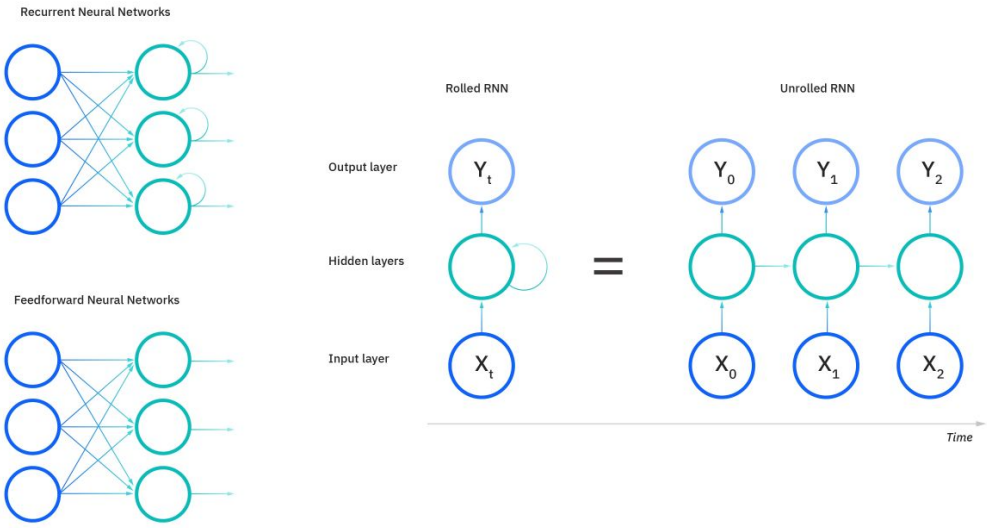

Figura 2.3: Comparación entre redes neuronales [22]

De la figura 2.3, se puede observar la diferencia entre una red neuronal recurrente y una red en la cual la información fluye en una sola dirección y no tiene memoria. Las redes feed foward solo ven la entrada actual, en las RNN en cambio, cada neurona toma como entrada tanto la entrada actual como la salida de la neurona anterior, repitiendo este proceso en forma de ciclos recursivos, provocando que la información fluya a través de la red en forma de bucles.

Un problema que tienen las RNN, es el desvanecimiento del gradiente. Cuando una red neuronal se entrena utilizando una retro-propagación del error, los gradientes se calculan a partir del error entre la salida de la red y el valor objetivo. Esos gradientes se usan para ajustar los pesos de la red, actualizándolos en la dirección que reduce el error. El problema está en que a medida que los gradientes se propagan hacia atrás a través de múltiples capas, pueden disminuir de manera exponencial, lo que resulta en gradientes muy pequeños en las capas iniciales de la red [23]. Una de las soluciones a este problema, fue crear nuevos tipos de arquitecturas de redes recurrentes, como las LSTM (Long short-term memory), que permiten que los gradientes fluyan a través del tiempo sin desvanecerse exponencialmente. Las redes LSTM, usan mecanismos de puertas que regulan el flujo de información y gradientes en la red, lo que les permite capturar dependencias a largo plazo de manera más efectiva [24].

El método de retro-propagación mencionado, fue formalizado en la década de 1980 por Rumelhart, Hinton, y Williams en el artículo [25]. A lo largo del tiempo, se han desarrollado y mejorado diversos métodos para mitigar problemas asociados con el algoritmo original. La retro-propagación moderna y sus técnicas asociadas, como la normalización por lotes (batch normalization) y las funciones de activación mejoradas, han sido discutidas en literatura extensa para abordar estos desafíos [26].

2.1.2.2.3. Redes Neuronales Bayesianas

Una red bayesiana (BN) es un tipo de modelo gráfico probabilístico que integra conceptos de redes neuronales con inferencia bayesiana. La diferencia que tiene con las redes neuronales comunes que producen predicciones puntuales, es que en este caso la red bayesiana asigna distribuciones de probabilidad a los pesos y sesgos de la red. Esto quiere decir que en vez de obtener un solo resultado, la red proporciona una gama de posibles resultados junto con sus

respectivas probabilidades. Al utilizar la inferencia bayesiana, las redes pueden capturar la incertidumbre inherente en los datos y en los parámetros del modelo.

La inferencia bayesiana se utiliza para actualizar las distribuciones de probabilidad a medida que se observan nuevos datos, lo que les permite adaptarse y generalizar de manera más efectiva, especialmente en situaciones con datos limitados. Además, estas redes son útiles para proporcionar estimaciones de la incertidumbre asociada con las predicciones del modelo, lo que es crucial en aplicaciones como la toma de decisiones autónoma.

2.1.2.2.4. Transformers

La arquitectura transformer presentada en [27], aprovecha un mecanismo único llamado autoatención, que le permite capturar dependencias y relaciones de largo alcance dentro de la entrada datos de manera más efectiva. Esta arquitectura consta de un stack de capas idénticas, cada una de las cuales contiene una autoatención de múltiples cabezales seguido de redes feedforward fully-connected por posición. Al evitar capas recurrentes o convolucionales, el modelo Transformer es paralelizable y computacionalmente más eficiente, lo que lleva a tiempos de entrenamiento más rápidos y rendimiento mejorado en varias tareas.

2.1.2.3. Generative AI

Los modelos de IA generativa se refieren a una variedad de métodos de IA que pueden aprender la distribución de datos a partir de ejemplos existentes y generar nuevos objetos de datos estructurados [28]. Aunque estos modelos también son métodos de aprendizaje profundo, se hace una distinción para destacar que se centran en la generación de datos. A diferencia de otros enfoques de aprendizaje profundo que se enfocan en la clasificación o la regresión, los modelos generativos aprenden la distribución subyacente de los datos y generan nuevos ejemplos muestreando a partir de esta distribución aprendida.

2.1.2.3.1. Redes Neuronales Generativas Adversarias

En el *Estudio de Síntesis*, se explica que las Redes Generativas Adversarias (GANs, Generative Adversarial Networks), son una arquitectura compuesta por dos módulos o modelos de redes neuronales profundas, uno es el generador G y el otro es el discriminador D. El modelo generador intenta generar muestras que sigan la distribución subyacente de los datos. Cabe destacar, que estas observaciones son lo suficientemente diferentes al conjunto de datos original, de tal modo que no es una simple replicación de observaciones que ya ocurren. El modelo discriminador D, que dada una observación del conjunto de datos original o sintetizadas por el generador, la clasifica como falsa si viene producida por el generador.

Estas dos redes son adversarias, es decir, G y D compiten o juegan un juego de suma cero, en donde si una de las redes gana la otra pierde. Mientras que G genera puntos de datos similares a los datos originales, con el propósito de engañar al discriminador, D intenta distinguir las observaciones generadas de las reales. El proceso de aprendizaje se realiza mediante la creación de muestras sintéticas del generador a partir de una distribución de entrada, y lo que tiene que hacer el discriminador es intentar distinguir o filtrar entre las muestras reales y las generadas por el generador. Las dos redes se entrenan en competencia mutua, donde el generador busca mejorar su habilidad para engañar al discriminador y el discriminador busca mejorar su capacidad para poder filtrar entre lo real y lo sintético proveniente del generador. A través de este proceso de competencia, la red GAN aprende a generar muestras cada vez más realistas y de alta calidad [29].

La figura 2.4 muestra el diagrama de una GAN, en donde a G se le entrega un ruido aleatorio, generalmente proveniente de una distribución normal multivariada para generar un conjunto de datos X_{fake}. D recibe tanto los datos originales X_{real} como los datos generados. D genera una etiqueta y que indica si una observación es falsa o real. Si es falsa, significa que proviene del generador G, en caso contrario, significa que proviene de los datos originales.

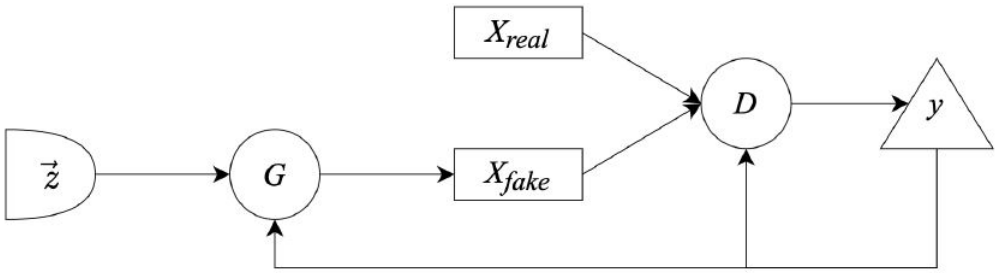

Figura 2.4: Diagrama de una GAN [9]

2.1.2.3.2. Redes Neuronales Autoencoders

Las redes del tipo autoencoder, son redes neuronales que tienen como objetivo aprender representaciones latentes (de dimensión reducida) de los datos de entrada. De manera más específica, una red autoencoder aprende a reconstruir los datos de entrada a partir de una versión comprimida de sí misma, teniendo como salida una predicción para la entrada. Como se expone en [5], para conseguir lo anterior la red tiene dos componentes, un codificador y un decodificador. Lo que hace el codificador, es transformar los datos de entrada mediante una compresión, obteniendo una representación de menor dimensión y capturando las características más relevantes de los datos. El decodificador toma esa representación comprimida y se encarga de reconstruir los datos de entrada a partir de ella.

La imagen de la figura 2.5 muestra la arquitectura general de un autoencoder. Se observa que el input entra a la componente del encoder, para luego transformarse en una representación latente en forma de cuello de botella. Esa representación entra a la componente del decoder, de tal manera que el output contiene la reconstrucción de esta etapa. La idea de esta arquitectura es minimizar la diferencia entre el input y el output reconstruido, aprendiendo una representación interna que captura las características más importantes del input.

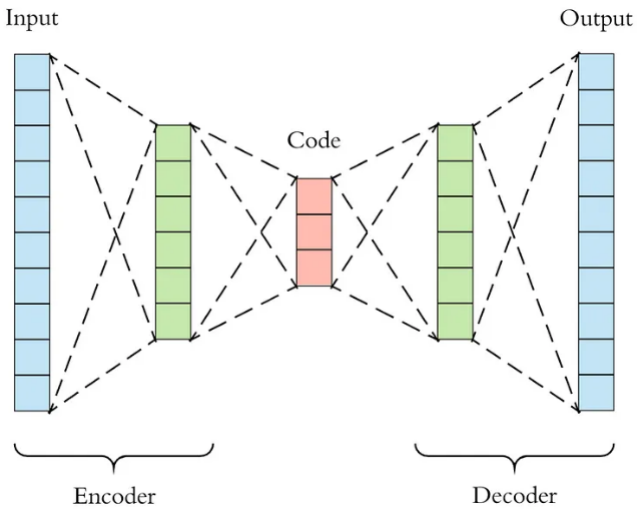

Figura 2.5: Esquema de la arquitectura de un autoencoder [30].

El codificador puede verse como una función $c = E(x)$, que produce una representación de baja dimensión de los datos, y el decodificador como una función $r = D(c)$, que produce una reconstrucción del código latente. El objetivo es aprender a copiar los datos originales sólo de manera aproximada y sólo las entradas que se parezcan a los datos originales [9]. Al restringirlo y obligarlo a aprender qué aspectos de los datos deben priorizarse, los codificadores automáticos pueden aprender propiedades útiles sobre los datos.

Al hablar de generación de muestras sintéticas, los codificadores automáticos presentan algunos problemas. La distribución aprendida de puntos en el espacio latente no está definida, lo que implica que la muestra decodificada no necesariamente es una muestra plausible. También, existe una falta de diversidad en las muestras generadas y los puntos generados que pertenecen a una misma clase pueden tener grandes espacios dentro del espacio latente.

Para solucionar lo anterior, se introdujeron los codificadores automáticos variacionales (VAE), que son una extensión de los codificadores automáticos básicos al introducir algunos cambios en el codificador y una función de pérdida [31]. El codificador de un VAE asigna cada punto de los datos originales a una distribución normal multivariada en el espacio latente (representada por los vectores de media y varianza). Los VAEs suponen que no hay correlación entre las dimensiones en el espacio latente, lo que significa que la matriz de covarianza es diagonal. Es decir, solo se consideran las varianzas de las dimensiones individuales y se asume que las covarianzas entre diferentes dimensiones son cero ($\sigma_{12} = \sigma_{21} = 0$, etc). Con este cambio se asegura que un punto determinado a, muestreado en la vecindad de otro punto b en el espacio latente, sea similar a b [9]. Por ende, un punto en el espacio latente que sea nuevo para el decodificador, probablemente producirá una muestra plausible.

El diagrama de elaboración propia mostrado en la figura 2.6, ilustra cómo los VAEs operan en un espacio latente 2D. Muestra una distribución normal multivariada con covarianza diagonal, destacando que las dimensiones z_1 y z_2 son independientes. Un codificador convierte puntos del espacio original (x) en el espacio latente (z), mientras que un decodificador transforma puntos del espacio latente de vuelta al espacio original. Se resalta que puntos cercanos en el espacio latente (a y b) tienden a ser similares, lo que asegura que muestras nuevas generadas

en el espacio latente sean plausibles.

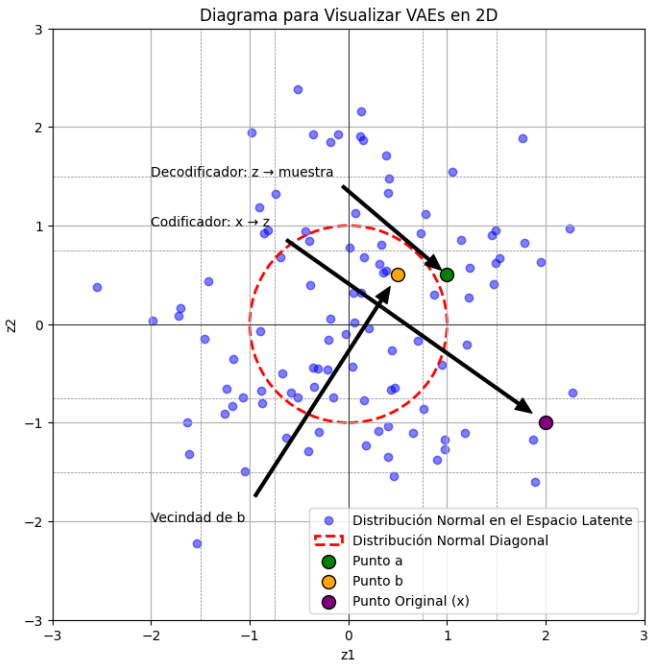

Figura 2.6: Diagrama 2D VAE

2.1.3. Evaluación de las muestras sintéticas

Con el objetivo de evaluar los métodos utilizados es necesario evaluar la calidad de las muestras generadas. Hay varias técnicas de evaluación, y en este trabajo se expondrán las técnicas más utilizadas en el contexto de datos tabulares.

2.1.3.1. Comparación estadística

La forma más simple de evaluar los datos sintéticos, es compararlos estadísticamente con los datos reales. Esa comparación estadística básica se puede hacer, por ejemplo, mediante la media, la mediana, desviación estándar, etc. En este caso, si los valores son similares, es posible que los datos sintéticos también sean similares. El problema con esta comparación, es que puede inducir a un error al no tomar en cuenta la distribución de los conjuntos.

2.1.3.2. Representaciones gráficas

Las representaciones gráficas como histogramas o diagramas de caja, pueden complementar la comparación estadística anterior. La comparación de las gráficas proporciona una evaluación visual de los datos sintéticos, permitiendo en este caso encontrar similitudes entre las distribuciones de los conjuntos. Según el *Estudio de Síntesis*, a pesar de que estas representaciones entregan una gran utilidad, este método puede ser muy costoso a la hora de tener altos volúmenes de datos con una gran cantidad de variables, por lo que es necesario explorar más técnicas.

2.1.3.3. Eficacia del aprendizaje automático

Otra opción para evaluar la calidad de las muestras generadas es utilizar la eficacia del aprendizaje automático que se expone en el *Estudio de Síntesis*. Como lo que se busca es mejorar el rendimiento de ciertos modelos de machine learning, la eficacia es usada para evaluar la calidad de los datos con respecto a los mismos modelos que se pretenden mejorar. Concretamente, si se tiene un conjunto de datos D que está dividido en conjunto de entrenamiento D_{train} y conjunto de test D_{test}, lo que se busca es comparar el rendimiento de un modelo M entre D_{train} y la data sintética D_{synth}. Entonces, si el rendimiento del modelo M en D_{train} es similar con D_{synth}, es probable que los datos sintéticos sean similares y sigan la distribución de los datos reales.

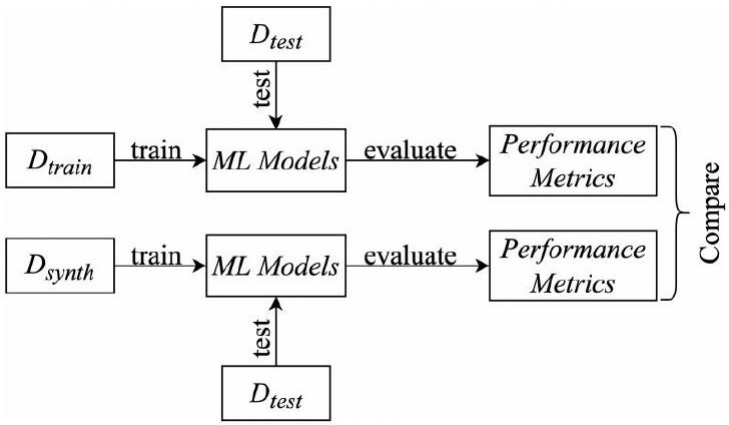

Figura 2.7: Diagrama de eficacia del aprendizaje automático [9].

La figura 2.7 muestra el diagrama de eficacia del aprendizaje automático. Se observa que el conjunto de pruebas D_{test}, se utiliza para comparar cuando un modelo M se entrena con datos de entrenamiento originales D_{train}, y cuando se entrena con datos sintéticos D_{synth}. El rendimiento de los modelos se visualiza mediante métricas o indicadores de rendimiento KPI (Key Performance Indicators), los más comunes son accuracy, precision y recall. Como se explica en [32], al accuracy indica la proporción de acierto de los modelos, precision indica la proporción de todas las clasificaciones positivas del modelo que son realmente positivas y recall indica la proporción de todos los casos positivos reales que fueron clasificados correctamente.

Lift

En el caso de ClaroVTR, como KPI principal se utiliza una métrica que tiene un uso extendido en el área de marketing [33] conocida como lift. En términos simples esta métrica se puede explicar de la siguiente manera; si se tiene un cierto modelo que predice si un cliente de un servicio cancelará su suscripción o no, se está en presencia de un problema de clasificación binaria, en donde un cliente cancela una suscripción (churn=1) o la mantiene (chrun=0), siendo el *Churn* la tasa de pérdida de clientes. La idea es agrupar datos en relación a la probabilidad de abandono prevista, ese agrupamiento generalmente se realiza mediante deciles, por lo que se tendrían 10 grupos entre propabilidad 0 y probabilidad 1. Lo siguiente es calcular la verdadera tasa de abandono por grupo, contando cuántos clientes de cada grupo abandonaron y dividiendo por el número total de cliente por grupo.

La utilizad de esta métrica, en este caso, viene dada porque el objetivo de este modelo de ejemplo es predecir si un cliente anula la suscripción, lo que quiere decir que la probabilidad prevista de abandono tiene que ser directamente proporcional a la de abandono real, vale decir, un valor alto debe correlacionarse con una tasa de abandono alta, y viceversa, si el modelo predice que un cliente no abandonará, se debe tener la certeza de que es poco probable que el cliente abandone.

Formalmente, el lift score se define en 2.2, que en el caso del modelo de ejemplo, *rate* hace referencia a la tasa de abandono. Para este cotenxto, se usa el top 10 % del lift.

$$\textbf{Top 10 \% lift} = \frac{\text{predicted rate}}{\text{average rate}} \tag{2.2}$$

Kolmogorov-Smirnov (KS)

ClaroVTR también utiliza la métrica KS en sus modelos, que es comúnmente usada en el contexto de predicción de modelos de clasificación binaria y puntuación de crédito. KS evalúa la discriminación de un modelo entre dos grupos distintos, como puede ser la clase positiva y la clase negativa. KS ordena todas las predicciones del modelo en orden descendente y calcula las funciones de distribución acumulativa para ambas clases. La métrica se determina al medir la máxima diferencia vertical entre estas dos funciones acumulativas.

Un valor alto de KS significa que el modelo es eficaz para discriminar entre las instancias positivas y negativas, lo que es muy importante en problemas donde se requiere que una clase crítica sea correcamente identificada. Por ejemplo, en el contexto de ClaroVTR, un alto KS podría indicar que el modelo puede distinguir de manera efectiva entre clientes que son propensos a incumplir con sus pagos (abandonan) y aquellos que no lo son (no abandonan).

2.2. Estado del Arte

Existen varias áreas en donde las técnicas de generación de datos tienen un alto impacto a la hora de abordar problemas del mundo real [28]. Entre las áreas más destacadas se tienen aplicaciones dentro de visión computacional, procesamiento de voz, procesamiento de lenguaje natural, salud, negocios, educación, entre otras. Muchos conjuntos de datos de estas áreas, tienen un formato tabular con propiedades únicas. Dentro del contexto de este trabajo, el tipo de dato que se abordará es justamente data tabular, por lo que en esta sección se mencionarán los últimos métodos o arquitecturas utilizadas en este tipo de datos.

Algunas de las propiedades únicas que tiene la data tabular, se mencionan en [9]. Un conjunto de datos tabulares puede tener tanto características continuas como características categóricas, por lo que acomodar los diferentes tipos de datos de variables de entrada es crucial al momento de seleccionar cualquier algoritmo. Además, se menciona que este tipo de datos puede presentar distribuciones no gaussianas y multimodales, lo que se debe tener en cuenta al generar datos sintéticos.

2.2.1. Métodos estándar

El uso de Borderline SMOTE se puede encontrar en [34], en donde los autores utilizan el método para tratar un problema de desequilibro y evaluar el impacto en un conjunto de datos de clasificación de electroencefalografía obtenidos de una interfaz cerebro-computadora. El resultado fue que el método no mejora sustancialmente la clasificación general, pero aumenta el rendimiento de los clasificadores que tenían los peores resultados.

ADASYN se usa en [35] en el contexto de identificación de fraude en telecomunicaciones a través de modelos de aprendizaje profundo. El conjunto de datos está desequilibrado y los autores usan ADASYN para tratar este problema. Se concluyó que ADASYN es más beneficioso que SMOTE y que las métricas de desempeño de los modelos mejoraron al utilizar este algoritmo.

En [36] también se usa ADASYN, al tratar el problema de la pérdida de clientes dentro de la industria de telecomunicaciones. En esta industria se tienen modelos de clasificación que predicen la pérdida de clientes, el problema es que la cantidad de clientes que no abandonan dominan a los clientes que sí abandonan, haciendo que los datos de estos últimos no sean familiares para el sistema. Los autores concluyeron que ADASYN mejora el algoritmo SMOTE.

En [37] los autores comparan el uso de K-Means SMOTE a otros algoritmos de generación de datos, demostrando que K-Means es mejor a la hora de equilibrar un conjunto de datos, permitiendo que los modelos de aprendizaje automático funcionen mejor en algunas métricas de desempeño de interés.

2.2.2. Métodos de aprendizaje profundo

En [38] se utilizan redes bayesianas en el contexto de compartir datos privados. Si bien es cierto, el enfoque de este trabajo se escapa al objetivo que tienen los autores de [38], es importante destacar que la red bayesiana agrega ruido a los datos para estimar una distribución

aproximada de los datos originales.

Existen varios modelos capaces de generar conjuntos de datos tabulares sintéticos, pero no muchos tienen la capacidad de producir conjuntos de datos relacionales. En [39] se propone modelar datos relacionales a través del uso de transformers, modelando una tabla principal en conjunto con sus relaciones entre tablas. Los resultados muestran que al utilizar conjuntos de datos reales, el modelo captura la estructura relacional mejor que otros modelos de referencia.

2.2.3. Generative AI

TGAN fue propuesta el 2018 por Lei Xu y Kaylan Veeramachaneni [40] como una arquitectura GAN para sintetizar datos tabulares. Teniendo un conjunto de datos D, dividido en D_{train} y D_{test}, TGAN tiene un doble objetivo; dado un modelo de ML, el accuracy en D_{test} cuando se entrena en D_{train} debe ser similar a su accuracy en D_{test} cuando se entrena con D_{synth} (data sintética). Además, la información mutua entre cada par de columnas en D y D_{synth} debe ser similar.

Como se describió en secciones anteriores, una GAN consta de dos redes neuronales, por lo que es necesario transformar los datos antes de usarlos como entrada a la red. En el caso de TGAN, se aplican dos técnicas específicas para preparar las variables numéricas y categóricas.

Como se indica en el *Estudio de Síntesis*, para las variables numéricas, se utiliza la normalización específica del modo. Esta técnica es útil para tratar con distribuciones de datos que son multimodales y no gaussianas. La normalización específica del modo implica ajustar un GMM, que modela la distribución de los datos como una combinación ponderada de múltiples distribuciones gaussianas. Para cada variable numérica, el GMM calcula la probabilidad de que una muestra provenga de cada una de estas distribuciones gaussianas. Estas probabilidades se usan para codificar los valores de las variables numéricas, permitiendo una representación más precisa de la distribución subyacente de los datos.

Para las variables categóricas, se realiza un suavizado mediante el uso de vectores one-hot encoder, de manera que los datos categóricos sean más útiles para la red al agregarles cierta aleatoriedad o ruido. Este proceso implica representar cada categoría como un vector binario. Luego, se agrega ruido a cada dimensión del vector one-hot y se renormaliza el vector resultante. Este suavizado ayuda a mejorar la representación de los datos categóricos, evitando que la red neuronal sea influenciada por valores atípicos y facilitando una mejor generalización.

En la figura 2.8 se muestra un ejemplo de juguete que tiene dos variables continuas y dos variables categóricas. Las redes usadas son una LSTM para el generador y una red fully-connected para el discriminador. La LSTM genera las variables numéricas en dos pasos y las variables categóricas en uno, el discriminador concatena todas las características y utiliza un perceptrón multicapa para distinguir entre muestras reales y falsas.

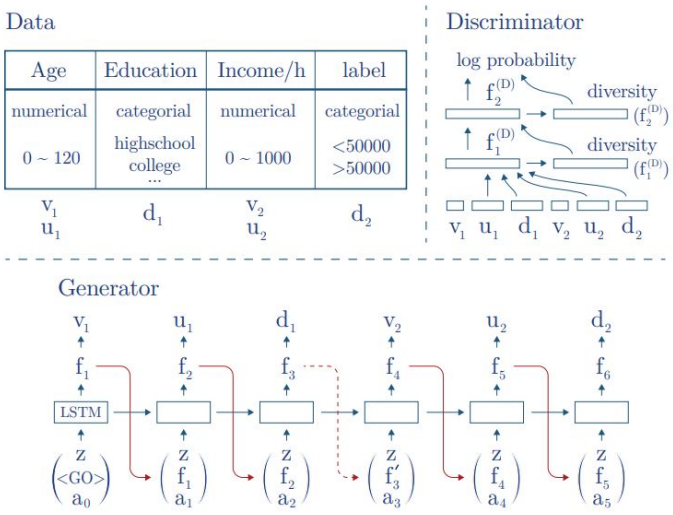

Figura 2.8: Diagrama del uso de una TGAN [40]

CTGAN que también fue propuesta por Lei Xu y Kaylan Veeramachaneni [41] el 2019, es una mejora a TGAN. Los objetivos son practicante los mismos, la única diferencia radica en que CTGAN aparte de preservar la correlación entre cualquier par de columnas de los datos generados, pretende mantener la distribución conjunta de todas las columnas.

Las transformaciones de los datos de entrada son similares a los realizados en TGAN. Para el caso de variables numéricas se reemplaza GMM por VGM (Mezcla Gaussiana Variacional), que a diferencia de tener un número de modos preestablecido, estima el número de modos para cada columna numérica, además, los valores continuos se representan como un vector one-hot que indica la moda y un escalar que indica el valor dentro de la moda. Las características categóricas se tratan usando vectores one-hot encoder sin agregar ruido.

La figura 2.9 muestra el modelo de CTGAN, en donde se utiliza un generador condicional que puede generar filas sintéticas que dependan de cualquiera de las columnas discretas. Adicionalmente, se propuso una técnica llamada entrenamiento por muestreo, que permite al CTGAN examinar de manera uniforme todos los valores discretos posibles. Este generador condicional se integra a la arquitectura GAN, mediante la utilización de un vector condicional (transformación simple a los vectores one-hot) que especifica que una columna categórica determinada debe ser igual a un valor determinado del conjunto de valores posibles para esa columna en particular.

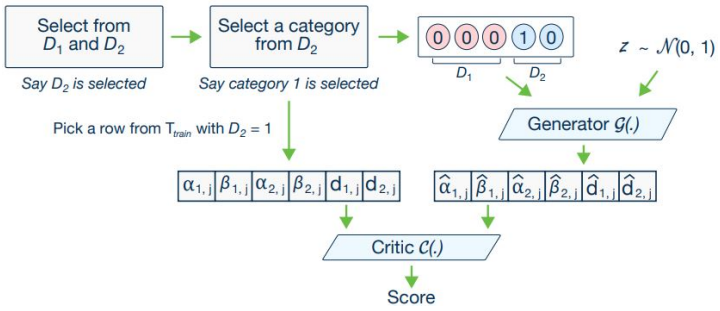

Figura 2.9: Diagrama CTGAN [41]

TabFairGAN propuesta el 2021 por Amirarsalan Rajabi y OzlemOzmen Garibay [42], es un tipo de GAN conocida como WGAN [43], que funciona como una extensión de GAN que modifica la fase de entrenamiento de modo que el discriminador, llamado crítico, se actualiza con más frecuencia que el generador en cada iteración [9]. Tal como en las arquitecturas anteriores es necesario preparar los datos de entrada, en este caso los autores utilizaron vectores one-hot para representar las características categóricas y una transformación cuantil para transformar las características numéricas.

La figura 2.10 muestra el modelo de esta arquitectura. El generador posee una capa inicial fully-connected, y una segunda capa que utiliza una función de activación para generar atributos numéricos y un tipo de softmax para representaciones de atributos categóricos. La salida se produce concatenando todos los atributos de la última capa del generador.

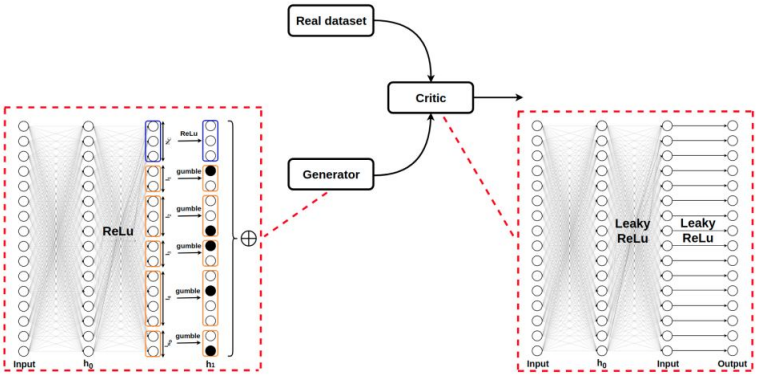

Figura 2.10: Diagrama TabFairGAN [42]

Capítulo 3

Metodología

3.1. Metodología

3.1.1. Bases de datos

ClaroVTR tiene en funcionamiento una serie de proyectos que operan en diversas áreas de la empresa, el punto de partida se realizará con un proyecto denominado Order-Entry, que consiste en un motor de crédito que a través de reglas de negocios y cálculos de riesgos de pago, entrega de cierta forma las posibles ofertas a un cliente. Los tipos de ventas tienen relación con la categorización que recibe un cliente según la solicitud que está haciendo o según el origen previo a pertenecer a ClaroVTR. Concretamente, se tienen 4 tipos de ventas; **Migraciones** son clientes que previamente tenían una línea prepago en ClaroVTR y ahora tienen una línea de postpago , **Línea Nueva** son clientes que adquieren una línea postpago sin información previa, **Post-Post** son clientes que tenían una línea postpago en otra compañía y ahora tienen una línea postpago en ClaroVTR, **Pre-Post** son clientes que tenían una línea prepago en otra compañía y ahora tienen una línea postpago en ClaroVTR.

Una de las entradas o inputs del proyecto Order-Entry corresponde a puntajes de riesgo que los modelos predictivos de los diferentes tipos de ventas pueden entregar. En este trabajo se es agnóstico sobre a qué tipo de ventas corresponden los datos, pero de todas formas, la primera base de datos para enfrentar el problema proviene del tipo de venta migraciones. Entonces, los modelos que están en operación son modelos ML predictivos y se entrenan con bases de datos que tienen un determinado número de features y provienen de algún tipo de venta. Se resuelve un problema binario, en que una clase representa los clientes que satisfacen condiciones de pago y no abandonan la línea, mientras que la otra clase representa los clientes que no satisfacen condiciones de pago y abandonan la línea.

La figura 3.1 muestra un diagrama simple y general del proyecto Order-Entry, en donde se puede observar los tipos de ventas, el motor de crédito, la oferta final y en qué etapa dentro del proyecto se ubica la data augmentation que servirá para los modelos predictivos que entregan un score de riesgo.

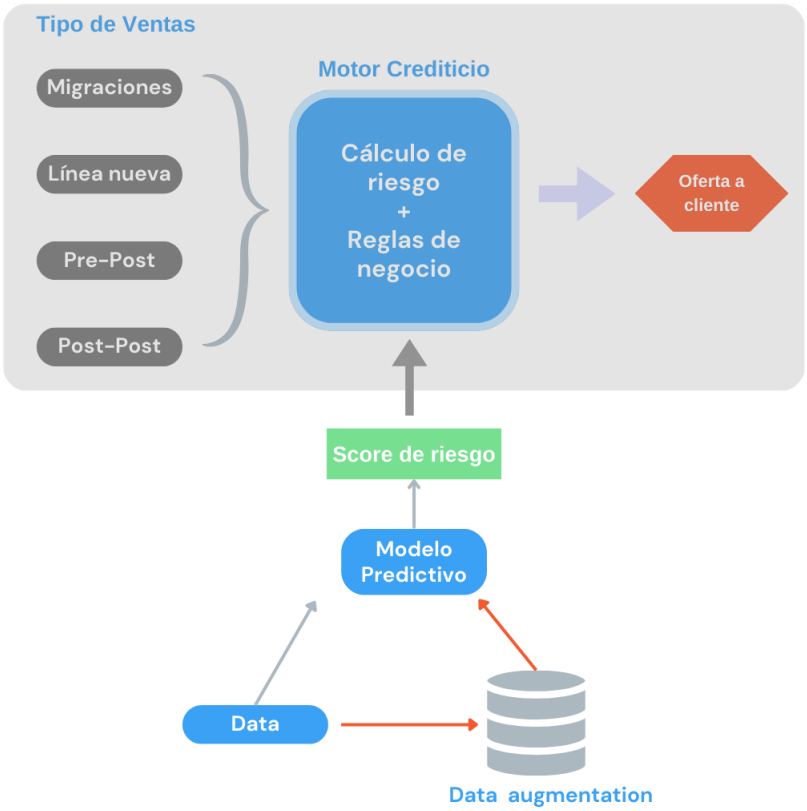

Figura 3.1: Diagrama Proyecto order-entry y modelo predictivo

Esta primera base de datos contiene 681 features o características de entrada, las cuales corresponden básicamente a patrones de consumo de los clientes, características socio demográficas y condiciones de portabilidad anteriores. Los datos se dividen en un conjunto de entrenamiento que tiene 71657 registros y un conjunto de test que tiene 18381. La tabla 3.1 muestra la distribución de clases del conjunto de train y la tabla 3.2 muestra la distribución de clases del conjunto de test.

Tabla 3.1: Conjunto de entrenamiento

Clase	**Cantidad**	**Porcentaje (%)**
0	44831	62.56
1	26828	37.44
Total	71657	100

Tabla 3.2: Conjunto de test

Clase	**Cantidad**	**Porcentaje (%)**
0	11183	60.84
1	7198	37.44
Total	18381	100

3.1.2. Resumen metodología

A continuación se presentará un orden de trabajo a seguir durante las 15 semanas de trabajo que corresponden al trabajo de título, para poder cumplir con los entregables y objetivos propuestos.

3.1.2.1. Baseline

Se realizará una configuración inicial que emulará las condiciones de los modelos que ClaroVTR tiene en operación, usando un conjunto de entrenamiento entregado. Esa configuración inicial consta en armar un set-up de experimentos de entrenamiento con distintos modelos de machine learning, considerando XGBoost, AdaBoost, y LightBoost. Para cada modelo existirán baseline en bruto sin ningún tipo de optimización y también existirán baseline en donde se haga una optimización de hiperparámetros. Cada baseline será testeado en el conjunto de test y se entregará un determinado KPI_{origen} a elección.

Más específicamente, en los experimentos iniciales se usará una configuración estándar para cada modelo, sin realizar cambios en los hiperparámetros. Esto servirá como referencia para entender el rendimiento inicial y proporcionará una línea de base para comparar con las mejoras posteriores. Luego, en los siguientes experimentos vendrá una etapa de mejora, en donde existirá una tratamiento y optimización de hiperparámetros. Esto podría incluir técnicas de búsqueda de hiperparámetros y validación cruzada.

3.1.2.2. Data augmentation

Se utilizarán 4 algoritmos de generación sintética de datos en el conjunto de entrenamiento, 2 algoritmos pertenecientes a los métodos estándar y 2 algoritmos pertenecientes a los métodos que utilizan aprendizaje profundo, con esto habrán 4 nuevos conjuntos de datos que tienen la data original y data aumentada. Todos los baseline armados anteriormente, serán entrenados con los 4 nuevos conjuntos de datos y serán testeados en el conjunto de test, de tal forma de obtener KPI_{new} para cada experimento.

Algunos algoritmos pueden generar datos más simples, mientras que otros pueden generar datos más complejos, por lo que al utilizar múltiples algoritmos es posible plantear una evaluación comparativa directa entre ellos, permitiendo explorar cómo la complejidad de los datos generados afecta el rendimiento de los modelos de machine learning.

3.1.2.3. Comparación estadística

Teniendo los KPI_{origen} y los KPI_{new} de cada experimento, es posible comparar los incrementos numéricamente y calcular Δ de diferencia entre los KPI, formando una matriz o tabla de comparación.

El Δ da una medida clara y cuantificable de la mejora o el empeoramiento en términos del KPI que se está utilizando. Permite una comparación uniforme entre diferentes configuraciones y modelos. Un Δ positivo indica una mejora, mientras que un Δ negativo sugiere un empeoramiento. Cuanto mayor sea el delta positivo, mayor será la mejora.

Analizar los Δ a lo largo de diferentes configuraciones y modelos ayuda a identificar tendencias y patrones. Se puede observar si ciertos algoritmos de data augmentation tienen un impacto más positivo en ciertos modelos o si hay consistencia en las mejoras.

3.1.2.4. Elección del algoritmo

Según los resultados del paso anterior, se podrá elegir el mejor algoritmo junto con la data aumentada que fue generada. Además, se realizarán comparaciones estadísticas básicas y se utilizarán representaciones visuales complementarias para ver si efectivamente la data generada es similar a la data original.

Se considerará realizar pruebas adicionales y validación de nuevos datos para verificar la robustez del algoritmo seleccionado. Se podría explorar el comportamiento en diferentes subconjuntos de nuevos datos.

3.1.2.5. Esquema simple de la metodología

La figura 3.2 ilustra la metodología en un esquema simple. En primer lugar, se tienen los distintos experimentos que varían según el modelo o el tratamiento de las características, esto se representa a través de la variable baseline que va desde un $i = 1, ..., n$, y que entrega un $KPI_{original}$. Luego, se tiene la sección de data augmentation con los 4 algoritmos seleccionados, entregando KPI_{new} a través del entrenamiento de los baseline en los nuevos conjuntos de datos aumentados. Con esto, se logran formar 4 tablas que registran los resultados y el Δ de diferencia entre los KPI para cada algoritmo. Siguiendo el flujo, se tiene la correspondiente comparación estadística para finalizar con la elección del mejor algoritmo junto con su data aumentada.

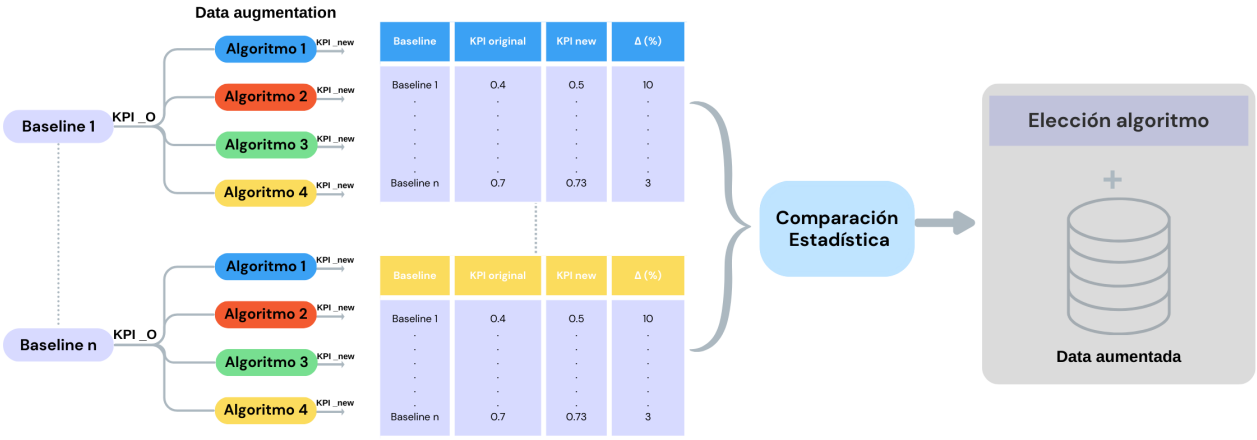

Figura 3.2: Esquema metodología

3.1.2.6. Solución final

Una vez que que la metodología se haya probado con las primeras bases de datos, se procederá a crear una librería/módulo en python, llamado desde ahora como motor, que realice todo este proceso de manera automática, de forma tal, que basta con entregar una base de datos como input para realizar el aumento de datos y entregar la data aumentada con el mejor algoritmo encontrado por el módulo.

3.1.3. Diagrama de flujo

La figura 3.3 muestra un diagrama de flujo simple, en donde se exponen las etapas explicadas anteriormente. A las primeras 4 etapas se les denominará como primera iteración.

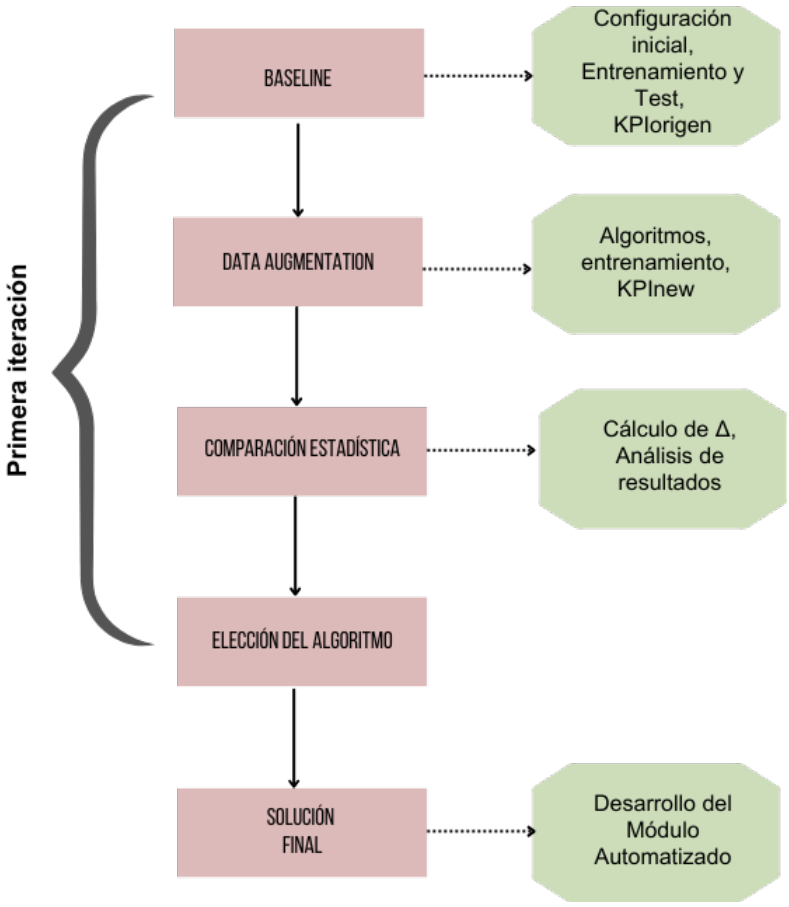

Figura 3.3: Diagrama de flujo metodología

Capítulo 4

Primera iteración

4.1. Resultados

En este capítulo se mostrarán los resultados de la primera iteración, vale decir, una base de datos con un conjunto de entrenamiento 3.1 y su respectivo conjunto de test 3.2. Estos resultados siguen la misma línea que la metodología mostrada en 3.1.2, previo a la elaboración de la solución final.

4.1.1. Baseline

Se tienen 3 modelos diferentes que utilizan la técnica de *gradient boosting*, AdaBoost, LightBoost y XGBoost. Cada modelo presenta dos configuraciones, una configuración de hiperparámetros estándar y una configuración con hiperparámetros optimizada. Para cada configuración se calcula el top 10 % lift, estableciendo las bases para mejoras.

La tabla 4.1 muestra el top 10 % lift en test que obtiene cada modelo en sus distintas configuraciones, en donde los modelos con subíndice 1 indican una configuración estándar y los modelos con subíndice 2 indican una configuración optimizada.

Tabla 4.1: Top 10 % lift modelos base

Modelo	**Top 10 % lift test**
$AdaBoost_1$	1.56
$AdaBoost_2$	1.61
$LightBoost_1$	1.65
$LightBoost_2$	1.64
$XGBoost_1$	1.59
$XGBoost_2$	1.59

4.1.2. Data augmentation

En esta sección se muestran los resultados que se obtienen utilizando 4 algoritmos de aumento de datos; BorderlineSmote, ADASYN, TVAE y CTGAN.

4.1.2.1. BorderlineSMOTE

La tabla 4.2 muestra el top 10 % del lift en test al utilizar BorderlineSMOTE.

Tabla 4.2: Top 10 % lift test - BorderlineSMOTE

Modelo	**Top 10 % lift test**
$AdaBoost_1$	1.50
$AdaBoost_2$	1.59
$LightBoost_1$	1.64
$LightBoost_2$	1.62
$XGBoost_1$	1.57
$XGBoost_2$	1.61

La tabla 4.3 muestra el desempeño de los modelos en las muestras sintéticas generadas por el algoritmo.

Tabla 4.3: Accuracy en muestras sintéticas - BorderlineSMOTE

Modelo	**Accuracy**	**Soporte**
$AdaBoost_1$	0.88	9038
$AdaBoost_2$	0.89	4555
$LightBoost_1$	0.99	11279
$LightBoost_2$	0.99	9038
$XGBoost_1$	0.99	11279
$XGBoost_2$	0.99	6797

4.1.2.2. ADASYN

La tabla 4.4 muestra el top 10 % del lift en test al utilizar ADASYN.

Tabla 4.4: Top 10 % lift test - ADASYN

Modelo	**Top 10 % lift test**
$AdaBoost_1$	1.54
$AdaBoost_2$	1.63
$LightBoost_1$	1.65
$LightBoost_2$	1.63
$XGBoost_1$	1.57
$XGBoost_2$	1.62

La tabla 4.5 muestra el desempeño de los modelos en las muestras sintéticas generadas por el algoritmo.

Tabla 4.5: Accuracy en muestras sintéticas - ADASYN

Modelo	**Accuracy**	**Soporte**
$AdaBoost_1$	0.37	728
$AdaBoost_2$	0.31	707
$LightBoost_1$	0.91	707
$LightBoost_2$	0.99	6196
$XGBoost_1$	0.99	6196
$XGBoost_2$	0.99	9493

4.1.2.3. TVAE

La tabla 4.6 muestra el top 10 % del lift en test al utilizar TVAE.

Tabla 4.6: Top 10 % lift test - TVAE

Modelo	**Top 10 % lift test**
$AdaBoost_1$	1.54
$AdaBoost_2$	1.60
$LightBoost_1$	1.64
$LightBoost_2$	1.67
$XGBoost_1$	1.61
$XGBoost_2$	1.62

La tabla 4.7 muestra el desempeño de los modelos en las muestras sintéticas generadas por el algoritmo.

Tabla 4.7: Accuracy en muestras sintéticas - TVAE

Modelo	**Accuracy**	**Soporte**
$AdaBoost_1$	1.00	8925
$AdaBoost_2$	1.00	4567
$LightBoost_1$	1.00	4567
$LightBoost_2$	1.00	4567
$XGBoost_1$	1.00	8925
$XGBoost_2$	1.00	8925

4.1.2.4. CTGAN

La tabla 4.8 muestra el top 10 % del lift en test al utilizar CTGAN.

Tabla 4.8: Top 10 % lift test - CTGAN

Modelo	Top 10 % lift test
$AdaBoost_1$	1.56
$AdaBoost_2$	1.63
$LightBoost_1$	1.66
$LightBoost_2$	1.62
$XGBoost_1$	1.60
$XGBoost_2$	1.64

La tabla 4.9 muestra el desempeño de los modelos en las muestras sintéticas generadas por el algoritmo.

Tabla 4.9: Accuracy en muestras sintéticas - CTGAN

Modelo	Accuracy	Soporte
$AdaBoost_1$	1.00	8399
$AdaBoost_2$	1.00	8399
$LightBoost_1$	1.00	2051
$LightBoost_2$	1.00	411
$XGBoost_1$	1.00	14917
$XGBoost_2$	1.00	10619

4.1.3. Comparación estadística

4.1.3.1. Comparación entre modelos

Al comparar el KPI_{origen} de los modelos en sus configuraciones estándar y optimizadas, se observa que para el modelo AdaBoost efectivamente existen mejoras, mientras que para LightBoost y XGBoost los resultados se mantienen similares. La diferencia en cómo la optimización de hiperparámetros afecta el rendimiento de AdaBoost en comparación con modelos más complejos como XGBoost o LightBoost, puede atribuirse a varias razones que se explican a continuación:

- AdaBoost es relativamente más simple que modelos como LightBoost o XGBoost, que son modelos de boosting más potentes y complejos. Este hecho se puede traducir en que los modelos más potentes son menos sensibles a pequeñas variaciones en los hiperparámetros, ya que tienen más capacidad para aprender patrones complejos en los datos de entrenamiento.

- LightBoost y XGBoost tienen más hiperparámetros para optimizar que AdaBoost. Esto significa que aunque se realice una optimización de hiperparámetros, es posible que no se encuentre una combinación que mejore significativamente el rendimiento en comparación con la configuración estándar. Por otro lado, AdaBoost tiene menos hiperparámetros para ajustar, lo que podría hacer que la optimización tenga un impacto más notable en el rendimiento.

- Modelos más complejos como LightBoost y XGBoost tienen una mayor capacidad para adaptarse a los datos de entrenamiento, lo que aumenta el riesgo de sobreajuste. En algunos casos la optimización agresiva de hiperparámetros puede conducir a un sobreajuste, lo que significa que el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos. Por lo tanto, incluso si se encuentra una combinación de hiperparámetros que mejore el rendimiento en los datos de entrenamiento, es posible que no se traduzca en una mejora en los datos de test.

Entonces, la diferencia en los resultados puede atribuirse a la naturaleza y complejidad de los modelos, la cantidad de hiperparámetros que se pueden ajustar y el riesgo de sobreajuste asociado con modelos más complejos.

4.1.3.2. Impacto del aumento de datos

Al evaluar cómo afecta el rendimiento de cada modelo el uso de datos aumentados por diferentes algoritmos de generación de datos sintéticos, se observa lo siguiente:

- En AdaBoost se observa que el impacto del aumento de datos varía según el algoritmo utilizado tanto para la versión estándar como para la optimizada. Por ejemplo, el top 10 % lift varía de 1.5 a 1.56 para la versión estándar usando el algoritmo CTGAN, y de 1.61 a 1.63 para la versión optimizada usando el algoritmo ADASYN o CTGAN.
- En LightBoost, de manera similar a AdaBoost, el impacto varía en ambas configuraciones. Por ejemplo, el top 10 % lift varía de 1.65 a 1.67 en la versión estándar usando CTGAN, y de 1.64 a 1.67 en la versión optimizada usando TVAE.
- Un comportamiento similar se observa en XGBoost. El top 10 % lift varía de 1.59 a 1.61 para la versión estándar usando TVAE, y de 1.59 a 1.64 para la versión optimizada usando CTGAN.

En resumen, para este conjunto de datos en específico, se observa que la selección del algoritmo de generación de datos tienen un impacto significativo en el rendimiento de los modelos, y estos efectos pueden diferir entre las versiones estándar y optimizadas. Además, al observar los resultados de 4.3, 4.5, 4.7 y 4.9, se desprende que los modelos tienen un mejor desempeño con la data sintética creada por los algoritmos más complejos y costosos computacionalmente.

Sin embargo, el 100 % de accuracy en datos sintéticos no siempre mejora el lift en datos reales porque el modelo puede sobreajustarse a los datos sintéticos, que podrían no reflejar bien la distribución de los datos reales. Además, el tamaño y balance de las muestras sintéticas pueden afectar la generalización del modelo. Por lo tanto, un alto desempeño en datos sintéticos no garantiza una mejora en el lift en datos reales. De todas formas, a pesar de que pueda existir un sobreajuste, los datos sintéticos generados por algoritmos más costosos, aún pueden ofrecer una representación más completa de las características generadas por los algoritmos más simples.

Otro punto importante a destacar, es que los modelos más complejos como XGBoost o LightBoost presentan una mejora mayor en comparación al modelo simple AdaBoost. Esto podría

deberse a que los modelos más complejos tienen mejor capacidad de aprendizaje, pues pueden capturar relaciones más complejas, por lo que al aumentar la cantidad datos estos modelos tienen la oportunidad para aprender y generalizar patrones más sutiles. Esta misma razón se conecta al hecho de que pueden presentar una mayor flexibilidad a la hora de adaptarse a la diversidad de los datos aumentados, pues dependiendo de la calidad de la data sintética, ésta puede introducir variedad y complejidad en los datos, haciendo más difícil que los modelos simples se puedan adaptar. Además, un gran número de ejemplos sintéticos que no son representativos puede afectar negativamente la capacidad del modelo para generalizar.

Finalmente, se observa que con algunos algoritmos los resultados no mejoran sino que empeoran, lo que es un comportamiento que se puede repetir en diversas bases de datos. Algunas de las razones que explican esto se exponen a continuación:

- Algunos algoritmos de data augmentation pueden introducir muestras sintéticas que no representan fielmente la distribución subyacente de los datos reales. Esto puede llevar a que el modelo se sobreajuste a la data sintética, lo que significa que aprende patrones específicos de los datos sintéticos que no se generalizan bien a nuevos datos.
- La data sintética generada pueden contener ruido o información irrelevante. Esto puede confundir al modelo durante el proceso de entrenamiento y llevar a un rendimiento deficiente en datos reales.
- La inclusión de data sintética en el conjunto de datos de entrenamiento puede interferir con la capacidad del modelo para aprender de la data real. Si las muestras sintéticas son poco representativas o contradictorias con la data real, pueden dificultar el proceso de aprendizaje del modelo.
- Algunos algoritmos de aumento de datos tienen hiperparámetros que deben ser ajustados cuidadosamente para adaptarse al conjunto de datos. Si estos hiperparámetros no se seleccionan correctamente, dado el alto costo computacional de realizar esa selección, puede llevar a una generación ineficaz de datos sintéticos.

4.1.3.3. Características del conjunto de datos real y sintético

Para evaluar la similitud entre el conjunto de datos real y el conjunto de datos sintético, se realizó un análisis comparativo de las características seleccionadas al azar en ambos conjuntos. Este procedimiento se realizó para todos los modelos base, en este caso particular se expondrán los resultados obtenidos en el modelo XGBoost con los algoritmos ADASYN y SMOTE. La idea de este análisis a través de representaciones gráficas es determinar en qué medida las características del conjunto sintético replican las distribuciones presentes en el conjunto real.

Si se desea consultar los gráficos de otro conjunto de características, es posible revisar el anexo A.

El proceso de comparación se llevó a cabo en dos pasos fundamentales:

1. **Probabilidad Acumulativa**: Se seleccionaron varias características al azar de ambos conjuntos de datos, asegurándose de que fueran las mismas columnas en cada conjunto.

Para cada característica seleccionada, se generaron las curvas de probabilidad acumulativa tanto para la data real ($X_{original}$) como para la data sintética (X_{synth}). Esta representación gráfica permite observar la acumulación de probabilidad a través de los valores de la característica, proporcionando una visión clara de cómo se distribuyen los datos en cada conjunto. Si las curvas se superponen estrechamente, indica que la característica sintética replica adecuadamente la distribución observada en la característica real.

2. **Densidad de Probabilidad**: Además, se analizaron las curvas de densidad de probabilidad para las mismas características en ambos conjuntos. Las curvas de densidad muestran la concentración de valores en diferentes puntos del rango de la característica, lo que facilita la comparación de la forma general de la distribución entre los datos reales y sintéticos. Una similitud en la forma de estas curvas, sugiere que el conjunto sintético captura adecuadamente la estructura de los datos reales.

En el gráfico 4.1 se presentan las curvas de probabilidad acumulativa para cuatro características seleccionadas al azar a partir del conjunto de datos reales y el conjunto de datos sintéticos generados por el algoritmo BorderlineSMOTE. Cada gráfico compara la distribución de una característica específica en el conjunto de datos real (línea azul) con su correspondiente en el conjunto de datos sintético (línea roja).

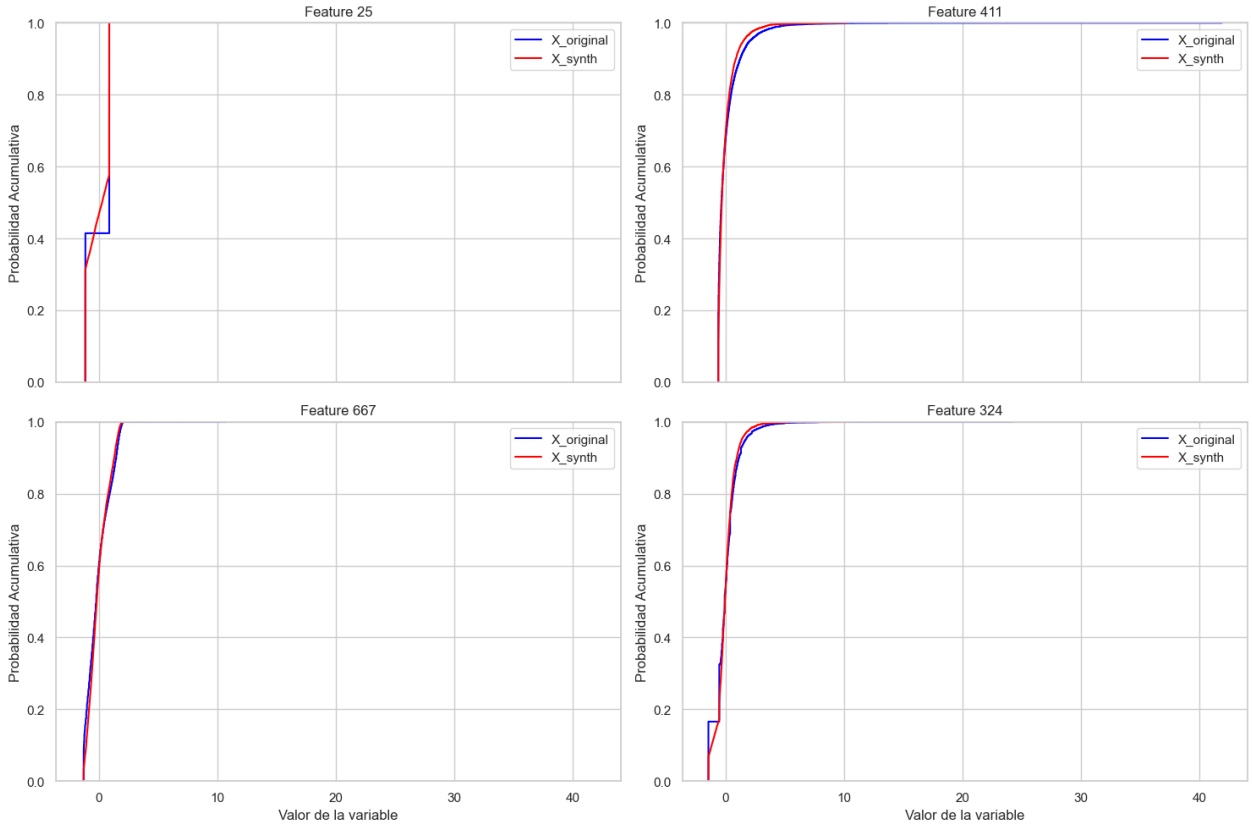

Figura 4.1: Probabilidad acumulativa $X_{original}$ vs X_{synth} SMOTE

En las cuatro características seleccionadas se observan dos casos, por un lado, las características 411 y 667 presentan una estrecha correspondencia entre la curva real $X_{original}$ y la

sintética X_{synth}, lo que indica una alta similitud en la distribución de esas características en particular. Para el caso de las características 25 y 324, las curvas siguen una tendencia similar, pero se observan diferencias en ciertos tramos. Estas discrepancias pueden indicar que en esas partes específicas del rango de valores, la distribución de la característica sintética no coincide completamente con la distribución real.

De manera análoga, en el gráfico 4.2 se presentan las mismas curvas explicadas anteriormente, pera para otras características seleccionadas aleatoriamente y para el caso de datos sintéticos generados por al algoritmo ADASYN.

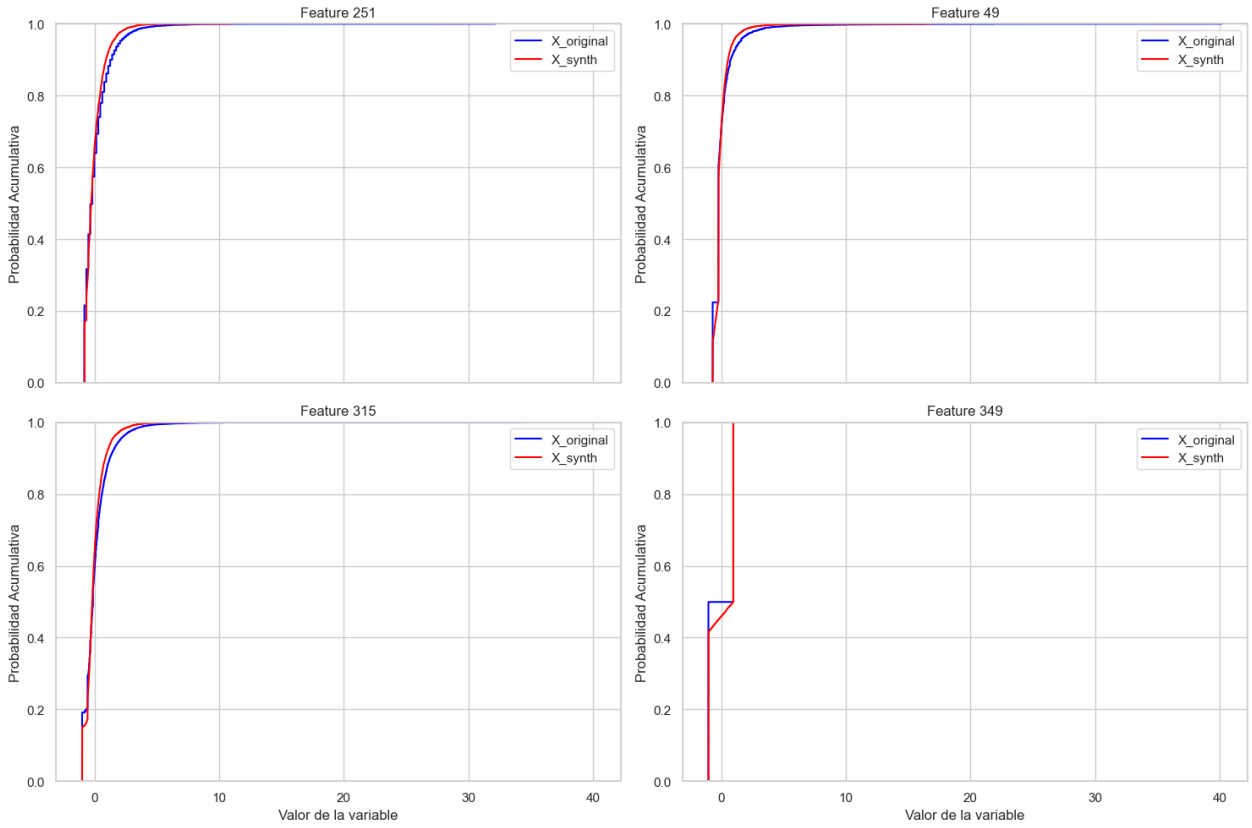

Figura 4.2: Probabilidad acumulativa $X_{original}$ vs X_{synth} ADASYN

En este caso las curvas también tienen una tendencia similar, pero de igual forma presentan discrepancias en ciertos tramos, destacando las característica 166 y 49.

En los gráficos 4.3 y 4.4 se presentan las curvas de densidad de probabilidad para cuatro características seleccionadas al azar a partir del conjunto de datos reales y el conjunto de datos sintéticos generados por los algoritmos BoderlineSMOTE y ADASYN, respectivamente.

En 4.3 la característica 25 tiene curva similar entre los datos reales y sintéticos, con dos peaks ubicados en las mismas posiciones, aunque los del conjunto sintético son más altos, indicando una mayor concentración en esas áreas. La característica 411 presenta curvas casi idénticas, lo que sugiere una alta fidelidad en la replicación de la distribución original. Por otro lado, las características 667 y 324 muestran tendencias similares, pero con diferencias en los peaks y valles, lo que refleja cierta variabilidad en la representación sintética de estas características.

En 4.4, las características 251 y 49 muestran curvas casi idénticas entre los datos reales y sintéticos, indicando una alta similitud en sus distribuciones. Sin embargo, para las características 315 y 349, aunque las curvas siguen una tendencia general similar, presentan diferencias en los peaks, lo que sugiere variaciones en cómo se distribuyen los datos en algunos puntos específicos.

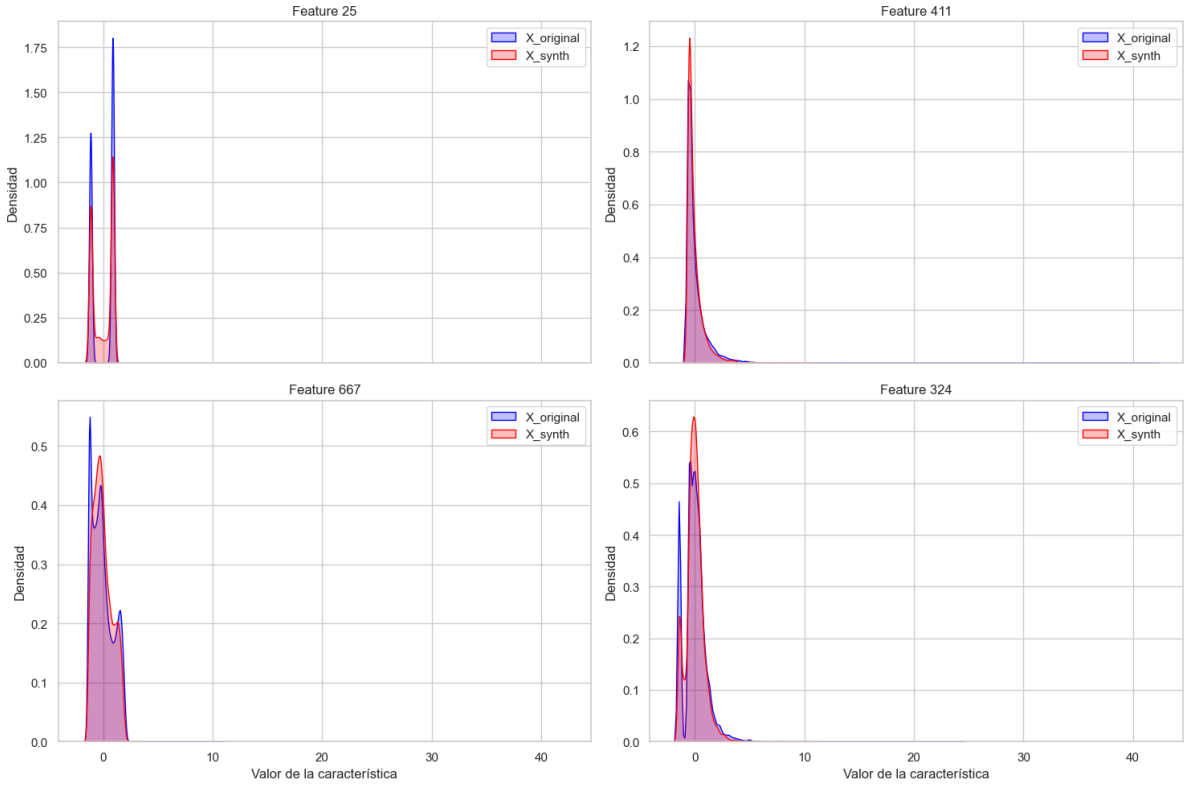

Figura 4.3: Densidad de probabilidad $X_{original}$ vs X_{synth} SMOTE

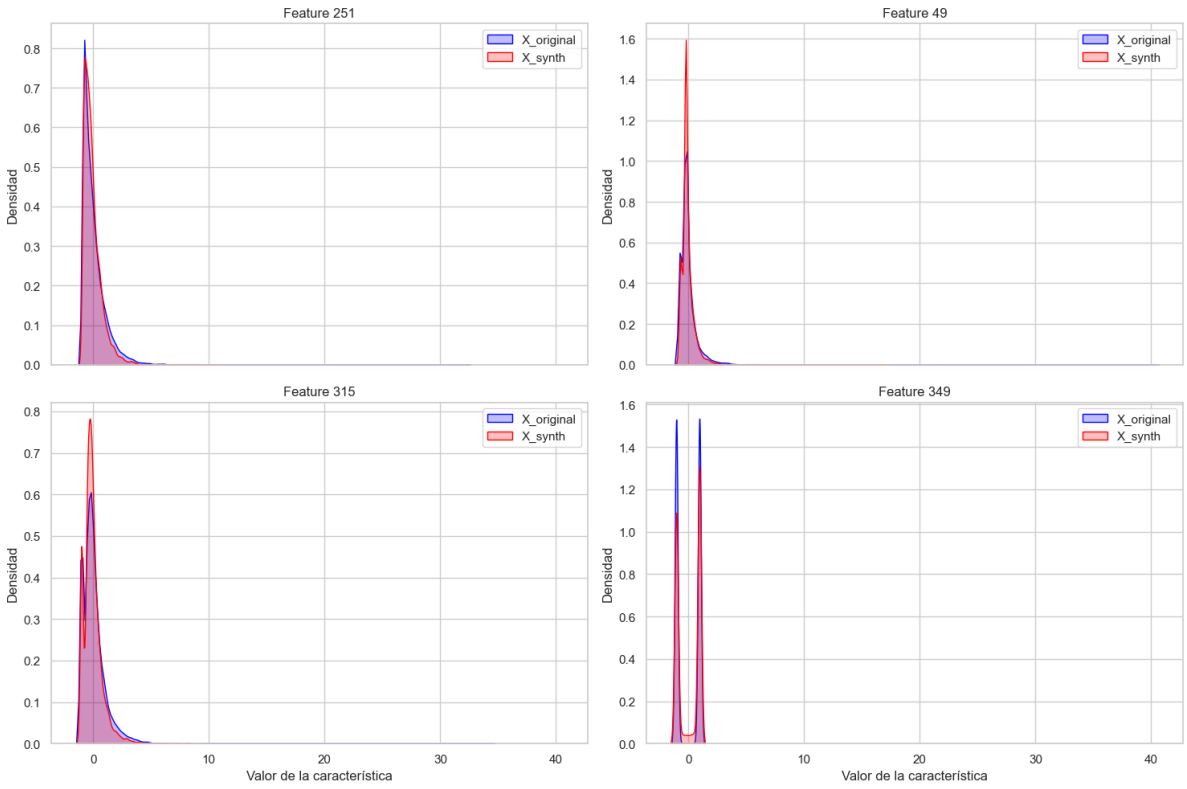

Figura 4.4: Densidad de probabilidad $X_{original}$ vs X_{synth} ADASYN

Este análisis comparativo de las características entre el conjunto de datos real y sintético a través de las curvas de probabilidad acumulativa y densidad de probabilidad, proporciona una comprensión de cómo los datos sintéticos reflejan la estructura de los datos reales. Considerando los resultados obtenidos en todos los modelos base, se tiene que en general, las características X_{synth} replican adecuadamente las distribuciones presentes en $X_{original}$ para este conjunto de datos, aunque con algunas variaciones que deben considerarse.

4.1.4. Elección del algoritmo

Esta etapa final va a depender del modelo de clasificación que se esté utilizando. Para el caso específico de ClaroVTR, la configuración que simula de manera más fehaciente la situación de los modelos utilizados en los proyectos de la empresa, es el modelo XGBoost optimizado ($XGBoost_2$). En consecuencia, el algoritmo que tiene mejor desempeño en el KPI para este conjunto específico de datos o en esta primera iteración, es CTGAN, por lo que la data aumentada final provendrá de ese algoritmo.

Capítulo 5

Diseño e implementación del motor

En este capítulo se explican aspectos del diseño e implementación de la solución final, denominada de ahora en adelante como *motor*. El objetivo principal es generar datos sintéticos que complementen los datos reales, mejorando así la capacidad de generalización de modelos de aprendizaje automático en entornos con datos limitados. El motor se estructura en módulos, cada uno desempeñando roles específicos en el flujo de trabajo de creación de modelos y aumento de datos. La implementación incluye detalles sobre la arquitectura, clases y métodos utilizados, funcionamiento de las tecnologías que utilizan los módulos, pruebas realizadas, y la validación del sistema.

5.1. Arquitectura del motor

5.1.1. Descripción general

El proyecto en su totalidad **DAEngine** está almacenado en un repositorio que se organiza en una arquitectura modular que facilita la comprensión y mantenimiento del código. Cada módulo del motor **DataAugmentationEngine** tiene una responsabilidad claramente definida, permitiendo la reutilización de componentes y la expansión del sistema implementado. La estructura general del proyecto se muestra a continuación.

```
DAEngine
├── notebook.ipynb
└── data_augmentation_engine
    ├── ___init___.py
    ├── engine.py
    ├── gan_wrapper.py
    ├── utils.py
    └── setup.py
```

5.1.2. Diagrama de clases

El diagrama de clases de la figura 5.1 proporciona una visión detallada de la interacción entre los módulos y las clases del motor. Muestra las relaciones y la estructura interna del motor, destacando cómo se comunican los diferentes componentes entre sí. Este diseño se realizó utilizando diagramas que siguen el estándar UML, en específico PlantUML, programa que permite la generación de diagramas que siguen el estándar UML mediante código [44].

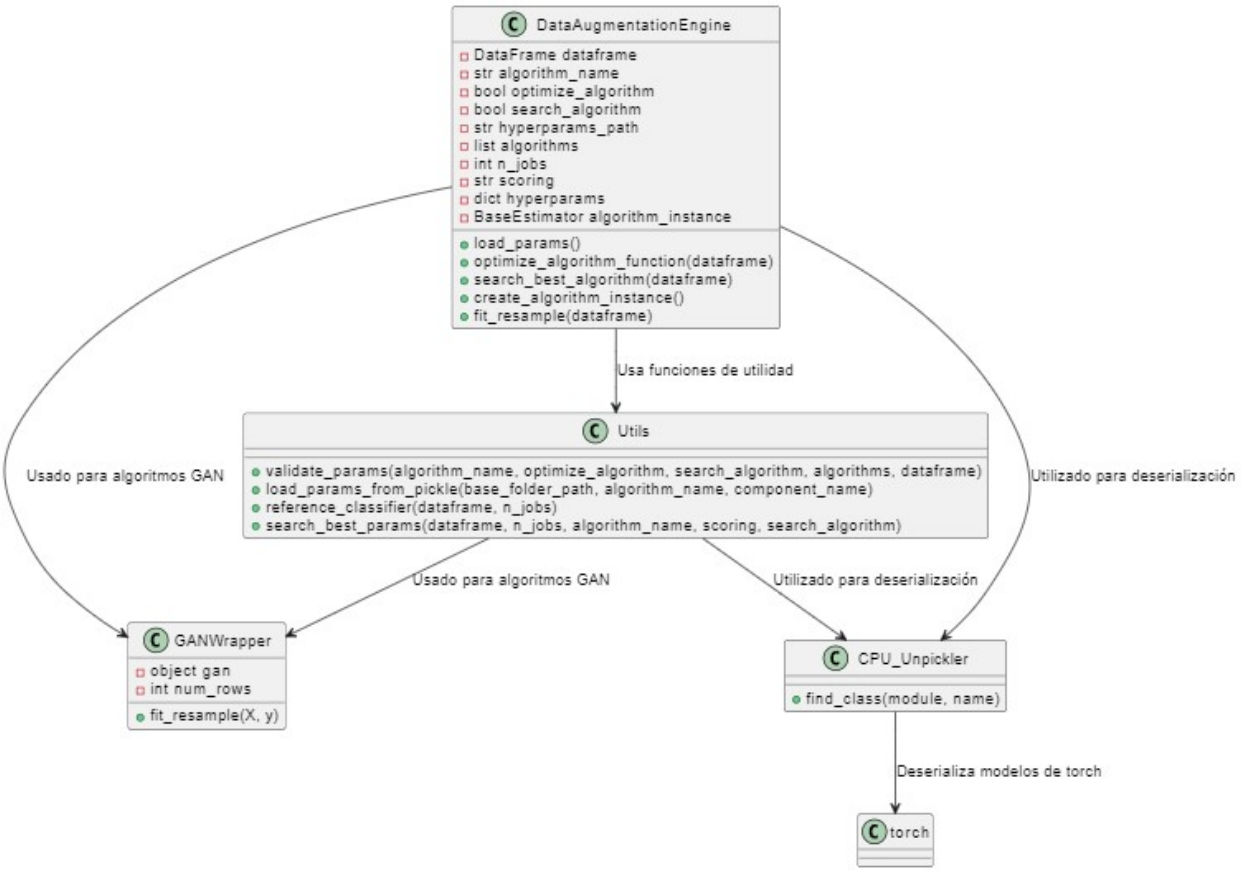

Figura 5.1: Diagrama de clases

5.1.3. Descripción de las clases y módulos

Es posible diferenciar 3 módulos principales que componen al motor, los cuales se describen a continuación:

5.1.3.1. DataAugmentationEngine

En la figura 5.1 es posible apreciar la clase DataAugmentationEngine (contenida en engine.py) que es el núcleo del motor, se encarga de la carga, aumento y salida de los datos. Utiliza algoritmos de sobremuestreo como ADASYN, BorderlineSMOTE, y también encapsula modelos GAN para la generación de datos sintéticos. Interactúa principalmente con la clase GANWrapper para generar datos sintéticos con algoritmos que utilicen objetos GAN y con Utils para tareas auxiliares.

5.1.3.2. GANWrapper

GANWrapper encapsula la funcionalidad de las redes GAN, proporcionando métodos para el entrenamiento, generación y remuestreo de los datos. Esta clase permite ajustar el número de filas a generar durante el entrenamiento de una red GAN. Se utiliza un wrapper o encapsulador para proporcionar una interfaz más intuitiva y personalizada en el contexto del motor, de forma tal, que se puedan implementar métodos personalizados acorde a las necesidades que tiene el motor de data aumentada. Esto abstrae la complejidad del uso directo de las redes GAN y ofrece una interfaz más amigable.

5.1.3.3. Utils

Este módulo contiene funciones de utilidad que se utilizan en la configuración y entrenamiento del motor de aumento de datos. Incluye la validación de parámetros, la carga de hiperparámetros desde archivos, la búsqueda de los mejores hiperparámetros para los algoritmos de aumento de datos y el modelo de referencia. Además, presenta la clase 'CPU_Unpickler' que permite encontrar un tipo de clase durante la deserialización.

5.1.3.4. Dependencias entre módulos

La comunicación entre módulos en la arquitectura propuesta se basa en una interacción bien definida y estructurada. Cada clase tiene un conjunto de responsabilidades específicas y se comunica con otras clases según sea necesario para cumplir con su función dentro del flujo del sistema. A continuación, se detalla cómo se comunican los módulos del motor:

'engine.py' y 'utils.py'

- La clase 'DataAugmentationEngine' en 'engine.py' es el punto de entrada principal para utilizar el motor de aumento de datos. Dentro de esta clase, se utilizan varias funciones definidas en 'utils.py' para realizar tareas como la validación de parámetros, la carga de hiperparámetros, la búsqueda de los mejores parámetros y la creación de un modelo de referencia.

- Cuando se inicializa un objeto 'DataAugmentationEngine', se llaman a las funciones de validación de parámetros en 'utils.py' para garantizar que los parámetros proporcionados sean válidos.

- El método *load_params* presente en la clase 'DataAugmentationEngine' utiliza la función *load_params_from_pickle* en 'utils.py' para cargar los hiperparámetros desde archivos pickle.

- El método *optimize_algorithm_function* y *search_best_algorithm* en DataAugmentationEngine llaman a *search_best_params* en 'utils.py' para buscar los mejores hiperparámetros o algoritmo de aumento de datos según sea el caso y los inputs.

- Finalmente, el método *fit_resample* en 'DataAugmentationEngine' coordina el proceso de aumento de datos, ajustando el motor y remuestreando la data. Este proceso se realiza utilizando los métodos anteriores que a su vez usan funciones de 'utils.py', como se explicó anteriormente.

'engine.py' y 'gan_wrapper.py'

- La clase 'DataAugmentationEngine' utiliza la clase 'GANWrapper' para envolver un modelo GAN cuando el algoritmo seleccionado es un modelo generativo adversario. Esto ocurre en el método *create_algorithm_instance* de DataAugmentationEngine.
- La clase 'GANWrapper' proporciona un método *fit_resample*, que es utilizado por 'DataAugmentationEngine' para generar muestras sintéticas utilizando el modelo GAN.

'utils.py' y 'gan_wrapper.py'

- La función *search_best_params* en 'utils.py' utiliza la clase 'GANWrapper' para generar muestras sintéticas cuando se busca el mejor algoritmo o se optimizan los hiperparámetros para un modelo GAN.

5.2. Implementación del motor

En esta subsección se presenta más en detalle la implementación de los módulos fundamentales del motor. Se incluyen explicaciones detalladas de las clases, funciones y métodos utilizados, junto con información sobre las licencias y tecnologías empleadas en el desarrollo.

5.2.1. Módulos y Funcionalidades

5.2.1.1. engine.py

Como se vio en secciones anteriores, este módulo contiene la clase DataAugmentationEngine, que actúa como el núcleo central del motor. Sus funcionalidades clave son:

- Método *__init__*: Este método inicializa el motor de Data Augmentation con una serie de parámetros configurables, incluido el DataFrame de entrada, el nombre del algoritmo, las opciones de optimización y búsqueda, y las métricas de evaluación. Permite una configuración flexible del proceso de aumento de datos.
- Método *load_params*: Carga los mejores hiperparámetros predefinidos del algoritmo de aumento de datos desde un archivo pickle. Esto asegura una configuración inicial coherente y optimizada para el algoritmo seleccionado.
- Método *optimize_algorithm_function*: Optimiza los hiperparámetros del algoritmo especificado utilizando una búsqueda exhaustiva con validación cruzada. Esto permite adaptar los parámetros del algoritmo a las características específicas del conjunto de datos.
- Método *search_best_algorithm*: Realiza una comparación entre varios algoritmos de aumento de datos para determinar el más adecuado para el conjunto de datos dado. Utiliza una estrategia de búsqueda exhaustiva para evaluar el rendimiento de cada algoritmo y seleccionar el mejor según una métrica de evaluación o KPI definido.
- Método *create_algorithm_instance*: Crea una instancia del algoritmo seleccionado, ya sea un algoritmo de sobremuestreo como ADASYN, BorderlineSMOTE, o un modelo GAN como CTGAN o TVAE. Esta instancia se utilizará posteriormente para generar datos sintéticos.

- Método *fit_resample*: Ajusta el motor de aumento de datos y realiza el aumento de datos en el conjunto de datos de entrada. Dependiendo de la configuración y el estado del motor, este método puede cargar hiperparámetros predefinidos, optimizar hiperparámetros, buscar el mejor algoritmo y crear instancias de algoritmos, todo esto antes de aplicar el aumento de datos y entregar la salida final.

5.2.1.2. gan_wrapper.py

Este módulo define la clase 'GANWrapper', que envuelve un modelo GAN y proporciona una interfaz para generar muestras sintéticas de la clase minoritaria. Sus principales funcionalidades claves son:

- Método *__init__*: Inicializa el encapsulador GAN con un modelo GAN preentrenado y el número de filas a generar. Permite una configuración flexible de la cantidad de datos sintéticos a generar.
- Método *fit_resample*: Genera muestras sintéticas de la clase minoritaria utilizando el modelo GAN encapsulado y las agrega al conjunto de datos original.

5.2.1.3. utils.py

Como se expuso anteriormente, este módulo contiene varias funciones auxiliares. Sus funcionalidades clave son:

- Función *validate_params*: Valida los parámetros de entrada para el motor, asegurando la coherencia y validez de las opciones seleccionadas.
- Función *load_params_from_pickle*: Carga los mejores hiperparámetros predefinidos del algoritmo de aumento de datos desde un archivo pickle, garantizando una configuración inicial óptima.
- Función *search_best_params*: Busca los mejores hiperparámetros para un algoritmo dado, utilizando una búsqueda exhaustiva con validación cruzada. Permite ajustar los parámetros del algoritmo para optimizar su rendimiento en el conjunto de datos de entrada.
- Función *reference_classifier*: Entrena un modelo de referencia utilizando un clasificador XGBoost, que sirve como punto de partida para evaluar el rendimiento de los algoritmos de aumento de datos.
- Clase *CPU_Unpickler*: Clase personalizada para deserializar objetos torch desde archivos pickle, especialmente útil para cargar modelos GAN en entornos sin GPU.

5.2.2. Manejo de errores y validación

En el código del motor se maneja la detección y gestión de errores en varias partes, de manera de poder anticipar y gestionar una variedad de situaciones excepcionales que podrían surgir durante su operación. Esto es fundamental para garantizar que el código pueda manejar casos inesperados sin fallar repentinamente y proporcionar información útil para la resolución de problemas. A continuación, se explican en detalle las estrategias de manejo de errores que fueron implementadas:

5.2.2.1. Validación de parámetros

En el módulo 'utils.py', se encuentra la función *validate_params*, que verifica que los parámetros y los datos de entrada cumplan con los requisitos esperados antes de proceder con el flujo del motor. Utiliza excepciones como *ValueError* y *RuntimeError* para gestionar errores relacionados con la validación de entradas. Esta estrategia de validación incluye:

- Verificación de Algoritmos: Se verifica si el nombre del algoritmo especificado está entre los permitidos ('ADASYN', 'BorderlineSMOTE', 'CTGAN', 'TVAE'). Si no lo está, se lanza un ValueError.
- Compatibilidad de GPU: Se verifica si se requiere una GPU para ejecutar ciertos algoritmos como CTGAN y TVAE. Si no hay GPU disponible y se intenta usar estos algoritmos, se lanza un RuntimeError.
- Verificación de DataFrame: Se asegura que el objeto proporcionado sea un DataFrame y que no contenga valores NaN. Si la columna label no está presente en el DataFrame o si contiene NaN, se lanza un ValueError.

La función *load_params_from_pickle* gestiona la carga de parámetros desde un archivo pickle. Maneja errores como FileNotFoundError cuando el archivo o la carpeta esperada no se encuentra.

La función *reference_classifier* entrena o carga un modelo de referencia, manejando posibles errores como la falta de existencia de archivos mediante FileNotFoundError. La función *search_best_params* busca los mejores hiperparámetros para un algoritmo dado y maneja errores relacionados con la inexistencia de archivos o directorios.

De esta forma, la implementación del manejo de errores en el motor de aumento de datos sigue una estrategia que incluye validación de parámetros de entrada, gestión de errores durante la carga de archivos, manejo de errores durante el entrenamiento y optimización de los modelos, captura de excepciones comunes y proporción de mensajes de error informativos. Estas medidas aseguran que el código pueda manejar de manera efectiva diferentes tipos de errores y excepciones, mejorando así la fiabilidad del motor.

5.2.2.2. Testing y validación del motor

En lugar de realizar un testeo formal y estructurado, se optó por utilizar notebooks interactivos para probar y validar el funcionamiento del motor. Esta metodología proporciona una manera efectiva y flexible de verificar la funcionalidad del motor utilizando conjuntos de datos simplificados, conocidos como conjuntos de datos de juguete.

Los notebooks se utilizaron como una herramienta dinámica para probar diferentes configuraciones. Usando este enfoque se permite la ejecución de código en bloques, facilitando la experimentación y la visualización de los resultados de manera rápida. Esto permite iterar fácilmente sobre diferentes ideas y detectar problemas o mejoras necesarias en el motor.

Para la validación, se emplearon conjuntos de datos de juguete que son pequeños y simplificados, diseñados para ilustrar la funcionalidad básica del motor sin la complejidad que pueden tener los datos reales. La idea general, es comprobar la ejecución correcta de todas

las funcionalidades.

El testeo implícito que se realizó del motor a través de notebooks y conjuntos de datos de juguete no reemplaza un testeo formal, en este sentido, para futuros trabajos se recomienda complementar el testeo en notebooks con un conjunto de pruebas más estructurado y automatizado para asegurar una cobertura completa y robusta del funcionamiento del motor en entornos más complejos o que puedan tener datos más variados.

En términos de resultados, una forma de validar el código es a través de la evaluación de la calidad de los datos sintéticos generados y el impacto en los modelos de aprendizaje automático, en otras palabras, la eficacia de los modelos. Como se explicó en secciones anteriores, la evaluación de datos sintéticos implica comparar los datos generados con los datos reales para asegurar que sean plausibles y útiles. Esto se puede lograr mediante la evaluación de similitud y el desempeño del modelo. Además, evaluar el impacto del aumento de datos en el desempeño del modelo de aprendizaje automático es crucial para validar la efectividad del enfoque propuesto. Esto se hace probando modelos entrenados con y sin datos sintéticos, que es justamente lo aplicado en el motor.

5.2.3. Modo de uso

En esta sección se realiza una descripción simple de cómo configurar, inicializar y aplicar los módulos que conforman el motor. El proyecto además, consta con una documentación respectiva y con tutoriales explícitos sobre cómo usar el paquete. En la sección B es posible encontrar de manera explícita el modo de uso del motor.

5.2.3.1. Preparación del entorno

Antes de utilizar el motor, es importante preparar el entorno adecuadamente para garantizar una ejecución fluida. Los siguientes pasos describen el proceso de configuración del entorno de trabajo:

Instalación de Dependencias

Se requieren ciertas dependencias que deben ser instaladas previamente. Estas están listadas en un archivo de requerimientos.

Configuración del Hardware

Para ejecutar los módulos que conforman el paquete, especialmente cuando se utiliza para algoritmos intensivos en cómputo como CTGAN o TVAE, se recomienda el uso de una GPU. En este proyecto, se utiliza Google Cloud Platform (GCP) para tener una instancia con GPU, proporcionando una mayor capacidad de procesamiento. Es imprescindinble instalar los controladores necesarios y las bibliotecas CUDA para la compatibilidad con GPU.

5.2.3.2. Importación

Para utilizar el motor que se encuentra dentro de una biblioteca/paquete, primero se debe importar dentro de un entorno de trabajo. Tomando en cuenta que el motor está implementado data_augmentation_engine, se realiza la importación como se muestra a continuación:

```
1 from data_augmentation_engine import DataAugmentationEngine
```

5.2.3.3. Carga de datos

El motor trabaja con datos cargados en un DataFrame de Pandas. Es fundamental que el DataFrame contenga una columna *label* que represente las etiquetas de la clase objetivo y que además no presente valores indefinidos o no representables.

5.2.3.4. Inicialización

La inicialización implica configurar sus parámetros para adaptarlo a la tarea específica de aumento de datos. Los parámetros a configurar incluyen el nombre del algoritmo, la optimización de parámetros, y la selección de algoritmos. La configuración inicial se realiza de la siguiente manera:

Parámetros de inicialización

- *dataframe*: Contiene los datos originales en formato de DataFrame.
- *algorithm_name*: Define el algoritmo de aumento de datos a utilizar.
- *optimize_algorithm*: Booleano que indica si se debe optimizar el algoritmo seleccionado.
- *search_algorithm*: Booleano que indica si se debe buscar el mejor algoritmo disponible.
- *hyperparams_path*: Ruta donde se encuentran almacenados los hiperparámetros predeterminados.
- *algorithms*: Lista de algoritmos para ser considerados en la comparación.
- *n_jobs*: Número de hilos de CPU a utilizar.
- *scoring*: Métrica utilizada para evaluar el rendimiento de los modelos.

Esta inicialización debe considerar la selección y configuración de lo que busca el usuario, puesto que el motor puede cargar parámetros predeterminados, optimizarlos o buscar el mejor algoritmo basado en un conjunto de datos. La tarea más simple es utilizar el algoritmo por defecto con sus parámetros que también están por defecto, y a partir de eso realizar la generación de datos. Otra posibilidad es que el usuario seleccione un algoritmo y lo optimice buscando los mejores parámetros según la data que ingresó, o utilizar todas las funcionalidades del paquete, que sería realizar una búsqueda completa entre los algoritmos que el usuario ponga en la lista.

5.2.3.5. Aplicación del Data Augmentation

Teniendo todo configurado, el siguiente paso es aplicar el aumento de datos. Esto implica ajustar el motor y generar un nuevo DataFrame que contenga los datos aumentados:

```
# se inicializa motor con sun configuración inicial
engine = DataAugmentationEngine(dataset, optimize_algorithm=True)

# se ajusta y se realiza el remuestreo
data_aumentada = engine.fit_resample(dataset)
```

La variable *data_aumentada* contiene los datos con sobremuestreo, incrementando la diversidad y cantidad de ejemplos para la clase minoritaria. Esta data puede ser guardada en el formato que estime el usuario.

5.2.3.6. Diagrama de flujo del motor

La figura 5.2 muestra un diagrama de flujo del motor, explicitando las etapas y las decisiones que se toman según lo que ingrese el usuario. El diagrama fue elaborado con Mermaid, que es una herramienta de código abierto para generar diagramas mediante código [45].

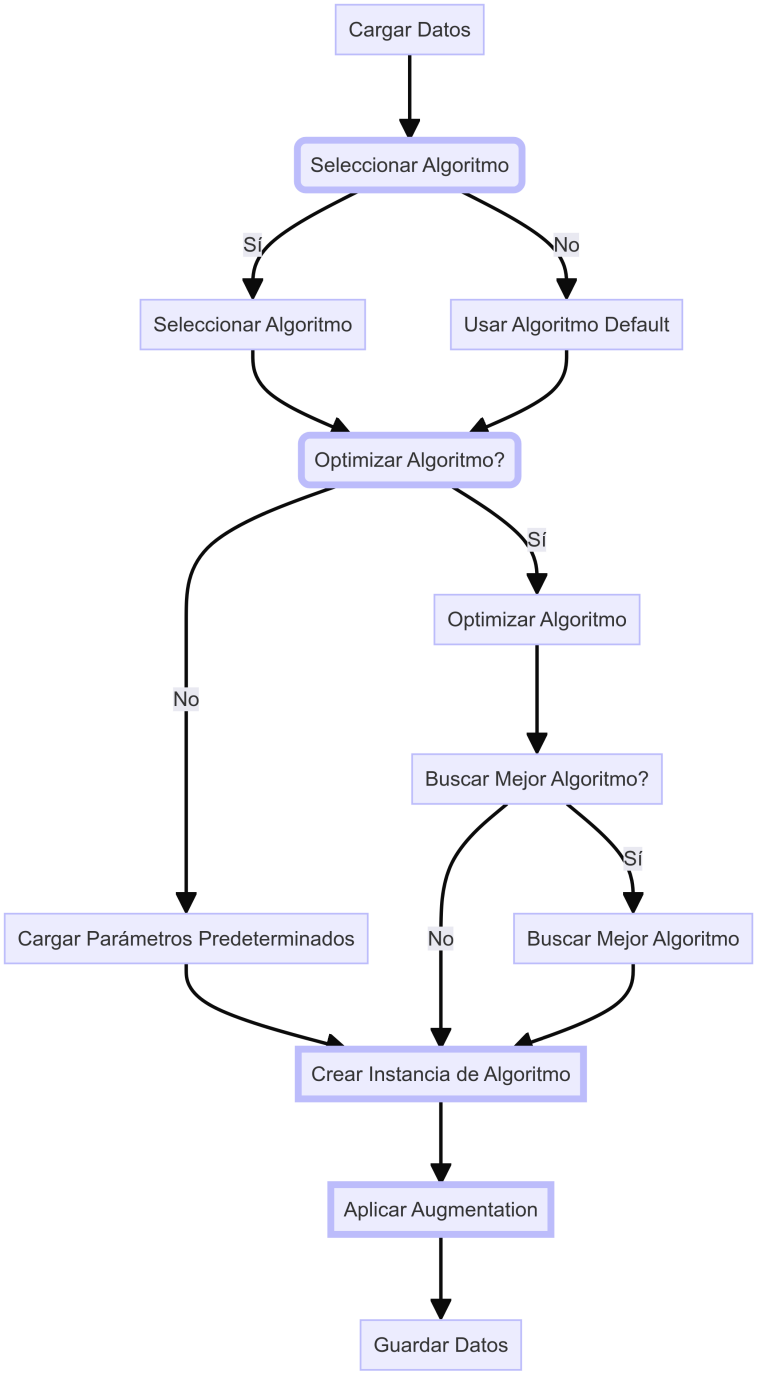

Figura 5.2: Diagrama de flujo del motor

El proceso de aumento de datos comienza con la preparación del entorno, asegurando que todos los recursos y configuraciones iniciales estén listos para la operación. Con el entorno preparado, se procede a cargar los datos en un DataFrame.

Una vez cargados los datos, se toma una decisión; seleccionar un algoritmo específico para el aumento de datos o utilizar el algoritmo predeterminado del motor. Esta elección permite al usuario adaptar el proceso según sus necesidades o dejar que el motor utilice su configuración por defecto.

El siguiente paso es considerar la optimización del algoritmo elegido. La optimización de hiperparámetros puede mejorar el rendimiento del algoritmo con los datos actuales, pero también es posible optar por usar los hiperparámetros predeterminados si no se requiere optimización.

En situaciones donde se disponen de múltiples algoritmos, como en el caso de que el usuario ingrese una lista de opciones, se puede optar por buscar el mejor algoritmo basado en la evaluación de su rendimiento con los datos. Esto implica una evaluación comparativa para determinar cuál algoritmo produce los mejores resultados. Hay que considerar que esta etapa es opcional y depende de la disponibilidad y necesidad de alternativas.

Con el algoritmo seleccionado y los hiperparámetros ajustados (ya sea optimizados o predeterminados), se crea una instancia del motor configurada con estas especificaciones. Esta instancia se utiliza para aplicar las técnicas de aumento datos. El proceso concluye con el guardado de los datos, completando así el flujo.

5.2.3.7. Uso de GPU

En aplicaciones que requieren una capacidad de procesamiento importante, como en el caso de los algoritmos CTGAN y TVAE, se utilizó Google Cloud Platform (GCP) para aprovisionar instancias con GPU. Los pasos esenciales que fueron necesarios para configurar y utilizar esas instancias, se resumen en una selección inicial de instancia que proporcione algún tipo de GPU junto con una instalación de controladores de GPU necesario y bibliotecas CUDA.

5.2.4. Tecnologías utilizadas

La implementación del motor se llevó a cabo principalmente en Python, aprovechando las bibliotecas de aprendizaje automático como PyTorch, Scikit-learn, XGBoost y SDV, para la construcción y evaluación de modelos. A continuación, se describen brevemente las tecnologías más importantes.

Python

El motor fue desarrollado en lenguaje Python, esto dado que presenta un gran número de librerías especializadas y que ayudan en contextos de aprendizaje automático, facilitando tareas como manipulación de datos, entrenamiento de modelos y evaluación de rendimiento. Además, permite el desarrollo eficiente de las funcionalidades del motor, junto con ser multiparadigma, haciendo más fácil la extensibilidad y mantención del código.

La naturaleza de código abierto de Python y su comunidad activa garantizan la disponibilidad de recursos, soporte y actualizaciones regulares que se pueden ir dando en las librerías o bibliotecas que se utilizan.

Pytorch

PyTorch es una biblioteca de aprendizaje profundo que ofrece una excelente combinación de flexibilidad y rendimiento [46]. Su capacidad para ejecutar modelos dinámicamente en tiempo de ejecución la hace especialmente adecuada para la el desarrollo de prototipos en aprendizaje profundo.

Sumado a lo anterior, proporciona una interfaz intuitiva y flexible para la construcción y entrenamiento de modelos de redes neuronales, que en el caso del motor, se utiliza en las redes GAN.

Scikit-learn

Sklearn es una biblioteca de aprendizaje automático de código abierto que proporciona una amplia gama de algoritmos de clasificación, regresión, clustering y preprocesamiento de datos [47]. Es ampliamente usada para tareas de aprendizaje supervisado y no supervisado debido a su facilidad de uso y eficacia. La interfaz que ofrece es fácil de usar para la construcción y la evaluación de modelo de aprendiza automático. Posee una extensa gama de algoritmos predefinidos y tiene herramientas de preprocesamiento que simplifican el proceso de desarrollo de modelos, facilitando una experimentación más rápida.

XGBoost

XGBoost es una biblioteca de gradient boosting diseñada para optimizar la velocidad y el rendimiento en conjuntos de datos que poseen un gran tamaño [48]. Es ampliamente reconocida por su precisión y eficacia en una variedad de problemas de aprendizaje automático, principalmente en clasificación y regresión. Sus características de regularización y manejo de datos faltantes lo hacen robusto frente a una variedad de escenarios en el contexto de datos reales. Además, su implementación eficiente en términos de velocidad lo hace adecuado para conjuntos de datos grandes y complejos.

En este sentido, XGBoost se eligió por su capacidad para manejar grandes volúmenes de datos de manera eficiente y por su precisión en tareas de predicción. Su algoritmo de boosting basado en árboles ofrece ventajas significativas en términos de velocidad y rendimiento, lo que lo hace adecuado para aplicaciones en donde la precisión y eficacia son fundamentales.

Synthetic Data Vault

SDV es un conjunto de bibliotecas de generación de datos sintéticos tabulares [49]. Utiliza una variedad de algoritmos de aprendizaje automático para aprender patrones a partir de datos reales y emularlos en datos sintéticos. En SDV se ofrecen múltiples modelos, que van desde métodos estadísticos clásicos hasta métodos de aprendizaje profundo como las CTGAN.

Git y GitHub

Git se utilizó para el control de versiones y almacenamiento del código en un repositorio, lo que permite gestionar cambios de manera estructurada y colaborativa. GitHub facilita la colaboración entre desarrolladores, el seguimiento de problemas y la gestión de versiones, proporcionando una plataforma centralizada para el desarrollo de software.

Google Cloud Platform

Se utilizó Google Cloud Platform (GCP) para acelerar tiempos de ejecución. Las instancias en GCP proporcionan capacidades avanzadas de procesamiento mediante CPUs y GPUs. GCP permite crear instancias con múltiples CPUs y GPUs, necesarias para manejar tareas de procesamiento intensivo.

Pueden existir una diversidad de bases de datos a las que se requiera realizar un aumento, si no se posee un entorno local el cual cumpla con los requerimientos del motor, es necesario recurrir a GCP y sus instancias. Por ejemplo, las GPUs aceleran el entrenamiento de modelos y la generación de datos, reduciendo significativamente los tiempos de cómputo, y si se desea realizar un búsqueda completa del motor, es necesario tener este tipo de recursos.

5.2.5. Licencias utilizadas

El proyecto utiliza varias librerías y frameworks de código abierto. Es importante asegurar el cumplimiento de las licencias respectivas, que permiten el uso, modificación y distribución del motor, de manera de garantizar el cumplimiento legal y el respeto por los derechos de propiedad intelectual. Las siguientes licencias se aplican en el desarrollo del motor y sus módulos.

- Licencia MIT: Utilizada para el código desarrollado, permitiendo su libre uso, modificación y distribución. La Licencia MIT promueve la colaboración en proyectos de código abierto, facilitando la integración y el desarrollo continuo. Según esto, el código desarrollado en esta memoria se distribuye bajo la licencia MIT, lo que permite su uso, modificación y distribución con pocas restricciones.
- Licencias de Bibliotecas de Terceros: Las bibliotecas utilizadas (NumPy, Pandas, Scikit-learn, PyTorch, SDV, XGBoost) tienen sus propias licencias de código abierto que permiten su uso y distribución. Estas licencias se han revisado para garantizar que sean compatibles con los objetivos del proyecto y los términos de la Licencia MIT.

En la implementación de sistemas de aumento de datos, como el propuesto en el motor, es fundamental considerar las implicaciones de privacidad, especialmente cuando se trabaja con datos sensibles. En este sentido, es importante que los datos sintéticos no infrinjan la privacidad involucrada en los datos reales, recomendando trabajar con datos anonimizados.

Capítulo 6

Resultados y análisis

En este capítulo, se presentan resultados y análisis del rendimiento del motor. El objetivo es evaluar cómo el motor responde a diferentes de conjuntos de datos, identificando en qué condiciones o contextos específicos logra mejoras y en cuáles no logra ofrecer ventajas importantes. Para ello, se han seleccionado una serie de bases de datos que varían en tamaño (filas), número de características (features), y complejidad de la estructura de los datos. Esta selección permite obtener una visión amplia y representativa del desempeño del motor en contextos diversos. Todas las bases de datos expuestas pertenecen al área de las telecomunicaciones, en donde se predice el churn de un cliente. El significado de cada feature no es necesario conocerlo, pues los datos están totalmente anonimizados.

6.1. Bases de datos

Para cada caso se mostrarán las dimensiones del conjunto de entrenamiento y test, junto con la distribución de clases. Además, se mostrará el top 10 % del lift en test, en el caso de que el modelo de clasificación se entrene con la data original y la data aumentada entregada por el motor.

6.1.1. Caso 1

La tabla 6.1 muestra la distribución de clases del conjunto de train y la tabla 6.2 muestra la distribución de clases del conjunto de test. Esta base de datos cuenta con 9 features por cada registro.

Tabla 6.1: Conjunto de entrenamiento caso 1

Clase	**Cantidad**	**Porcentaje** [%]
0	5505	83.48
1	1089	16.51
Total	6594	100

Tabla 6.2: Conjunto de test caso 1

Clase	**Cantidad**	**Porcentaje** [%]
0	1377	83.50
1	272	16.49
Total	1649	100

La tabla 6.3 muestra el top 10 % del lift en test al utilizar la data real y la data aumentada. El algoritmo escogido fue BorderlineSMOTE.

Tabla 6.3: Top 10 % lift test caso 1

Algoritmo	**Top 10 % lift test**
Data original	4.06
Data aumentada	4.25

La tabla 6.4 muestra los tiempos de ejecución de los diferentes pasos involucrados en la búsqueda.

Tabla 6.4: Tiempos de ejecución caso 1

Etapa	**Tiempo de ejecución [min]**
Modelo de referencia	0.1
ADASYN	0.01
BorderlineSMOTE	0.01
CTGAN	0.70
TVAE	0.29

6.1.2. Caso 2

La tabla 6.5 muestra la distribución de clases del conjunto de train y la tabla 6.6 muestra la distribución de clases del conjunto de test. Esta base de datos cuenta con 26 features por cada registro.

Tabla 6.5: Conjunto de entrenamiento caso 2

Clase	**Cantidad**	**Porcentaje** [%]
0	3622	73.47
1	1308	26.53
Total	4930	100

Tabla 6.6: Conjunto de test caso 2

Clase	**Cantidad**	**Porcentaje** [%]
0	1552	73.45
1	561	26.54
Total	10000	100

La tabla 6.7 muestra el top 10 % del lift en test al utilizar la data real y la data aumentada. El algoritmo escogido fue ADASYN.

Tabla 6.7: Top 10 % lift test caso 2

Algoritmo	**Top 10 % lift test**
Data original	2.78
Data aumentada	2.82

La tabla 6.8 muestra los tiempos de ejecución de los diferentes pasos involucrados en la búsqueda.

Tabla 6.8: Tiempos de ejecución caso 2

Etapa	**Tiempo de ejecución [min]**
Modelo de referencia	0.09
ADASYN	0.06
BorderlineSMOTE	0.06
CTGAN	0.50
TVAE	0.25

6.1.3. Caso 3

La tabla 6.9 muestra la distribución de clases del conjunto de train y la tabla 6.10 muestra la distribución de clases del conjunto de test. Esta base de datos cuenta con 12 features por cada registro.

Tabla 6.9: Conjunto de entrenamiento caso 3

Clase	**Cantidad**	**Porcentaje** [%]
0	136308	79.95
1	34179	20.04
Total	170487	100

Tabla 6.10: Conjunto de test caso 3

Clase	**Cantidad**	**Porcentaje** [%]
0	58418	79.85
1	14648	20.04
Total	73066	100

La tabla 6.11 muestra el top 10 % del lift en test al utilizar la data real y la data aumentada. El algoritmo escogido fue BorderlineSMOTE.

Tabla 6.11: Top 10 % lift test caso 3

Algoritmo	**Top 10 % lift test**
Data original	1.01
Data aumentada	1.02

La tabla 6.12 muestra los tiempos de ejecución de los diferentes pasos involucrados en la búsqueda.

Tabla 6.12: Tiempos de ejecución caso 3

Etapa	**Tiempo de ejecución [min]**
Modelo de referencia	1.00
ADASYN	0.40
BorderlineSMOTE	0.41
CTGAN	10.06
TVAE	6.52

6.1.4. Caso 4

La tabla 6.13 muestra la distribución de clases del conjunto de train y la tabla 6.14 muestra la distribución de clases del conjunto de test. Esta base de datos cuenta con 685 features por cada registro.

Tabla 6.13: Conjunto de entrenamiento caso 4

Clase	**Cantidad**	**Porcentaje** [%]
0	44831	62.56
1	26826	37.43
Total	71657	100

Tabla 6.14: Conjunto de test caso 4

Clase	**Cantidad**	**Porcentaje** [%]
0	11183	60.83
1	7198	39.46
Total	18381	100

La tabla 6.15 muestra el top 10 % del lift en test al utilizar la data real y la data aumentada. El algoritmo escogido fue CTGAN.

Tabla 6.15: Top 10 % lift test caso 4

Algoritmo	**Top 10 % lift test**
Data original	1.61
Data aumentada	1.64

La tabla 6.16 muestra los tiempos de ejecución de los diferentes pasos involucrados en la búsqueda.

Tabla 6.16: Tiempos de ejecución caso 4

Etapa	**Tiempo de ejecución [min]**
Modelo de referencia	29.89
ADASYN	6.83
BorderlineSMOTE	9.18
CTGAN	338.97
TVAE	238.79

6.1.5. Caso 5

La tabla 6.17 muestra la distribución de clases del conjunto de train y la tabla 6.18 muestra la distribución de clases del conjunto de test. Esta base de datos cuenta con 200 features por cada registro.

Tabla 6.17: Conjunto de entrenamiento caso 5

Clase	**Cantidad**	**Porcentaje** [%]
0	1064963	97.01
1	32786	2.98
Total	1097749	100

Tabla 6.18: Conjunto de test caso 5

Clase	**Cantidad**	**Porcentaje** [%]
0	456413	97.01
1	14051	2.98
Total	470464	100

La tabla 6.19 muestra el top 10 % del lift en test al utilizar la data real y la data aumentada. El algoritmo escogido fue CTGAN.

Tabla 6.19: Top 10 % lift test caso 5

Algoritmo	**Top 10 % lift test**
Data original	3.96
Data aumentada	3.94

La tabla 6.2.5 muestra los tiempos de ejecución de los diferentes pasos involucrados en la búsqueda.

Tabla 6.20: Tiempos de ejecución caso 5

Etapa	**Tiempo de ejecución [min]**
Modelo de referencia	26.22
ADASYN	47.10
BorderlineSMOTE	46.47
CTGAN	614.86
TVAE	475.80

6.1.6. Caso 6

La tabla 6.21 muestra la distribución de clases del conjunto de train y la tabla 6.22 muestra la distribución de clases del conjunto de test. Esta base de datos cuenta con 201 features y cerca de 5 millones de registros. Dada las dimensiones, la búsqueda se hizo entre los algoritmos del área de machine learning, es decir, ADASYN y BorderlineSMOTE.

Tabla 6.21: Conjunto de entrenamiento caso 6

Clase	**Cantidad**	**Porcentaje** [%]
0	4921792	99.6
1	17720	0.4
Total	4939512	100

Tabla 6.22: Conjunto de test caso 6

Clase	**Cantidad**	**Porcentaje** [%]
0	2109340	99.6
1	7594	0.4
Total	2116934	100

La tabla 6.23 muestra el top 10 % del lift en test al utilizar la data real y la data aumentada.

Tabla 6.23: Top 10 % lift test caso 6

Algoritmo	**Top 10 % lift test**
Data original	4.03
Data aumentada	3.54

La tabla 6.2.5 muestra los tiempos de ejecución de los diferentes pasos involucrados en la búsqueda.

Tabla 6.24: Tiempos de ejecución caso 6

Etapa	**Tiempo de ejecución [min]**
Modelo de referencia	113.44
ADASYN	93.71
BorderlineSMOTE	95.47

6.2. Análisis

6.2.1. Caso 1

En 6.1.1 la simplicidad del dataset con sólo 9 features y un número relativamente pequeño de registros (6594), ayuda a que el modelo aprenda patrones relevantes, incluso con la data sintética. La mejora del lift de 4.06 a 4.25 sugiere que la data aumentada ha sido efectiva para abordar el desbalanceo severo de clases. En este caso, la generación de ejemplos adicionales de la clase minoritaria probablemente ha permitido al modelo capturar mejor las features distintivas de esta clase, mejorando así su rendimiento predictivo. Además, con un menor número de features, la probabilidad de que la data sintética introduzca ruido o información irrelevante es menor, lo que contribuye a una mejora más clara en el desempeño del modelo.

En este caso, los tiempos de ejecución son relativamente cortos debido al pequeño tamaño del dataset y al bajo número de features. Esto permite iteraciones más rápidas durante el entrenamiento del modelo y la generación de datos sintéticos.

6.2.2. Caso 2

En el caso 6.1.2 que tiene 26 features, el modelo tiene más información para trabajar, lo que puede proporcionar una representación más detallada de los patrones en los datos. Sin embargo, la mejora es leve, con el lift aumentando de 2.78 a 2.82. Esto podría deberse a que,

aunque el desbalanceo de clases es menos severo en comparación con el caso 6.1.1, la complejidad adicional introducida por el mayor número de features requiere una generación de datos sintéticos que sea muy precisa y representativa de la distribución real de los datos. La data aumentada parece haber entregado algunos beneficios adicionales, pero estos beneficios son limitados porque el modelo ya tenía suficiente información para capturar algunas features de la clase minoritaria sin necesidad de un aumento significativo de datos.

Aumentar el número de features y registros incrementa los tiempos de entrenamiento y generación. Aunque todavía es manejable, estos tiempos pueden comenzar a impactar la eficiencia del modelo, haciendo que ajustes adicionales sean más costosos en términos de tiempo.

6.2.3. Caso 3

Aunque el desbalanceo es alto en 6.1.3 y el dataset tiene una pequeña cantidad de features (12), el gran número de registros (170487) sugiere que el modelo ya tiene suficiente información para capturar diferencias significativas. La mejora marginal del lift de 1.01 a 1.02, indica que la data aumentada tuvo un impacto limitado. Esto podría ser porque la data sintética generada no fue lo suficientemente representativa o porque el modelo ya estaba cerca de su capacidad máxima de rendimiento con los datos originales, pues en datasets con muchos registros, la influencia de los datos sintéticos puede diluirse, especialmente si los datos originales ya son suficientes para entrenar un modelo robusto. Sumado a esto, en datasets con un número moderado de características, la calidad de la data sintética y su capacidad para reflejar con precisión la diversidad de los datos reales es muy importante para cualquier mejora significativa.

En este caso, el gran volumen de registros eleva los tiempos de entrenamiento y generación de datos sintéticos, pero sigue siendo manejable.

6.2.4. Caso 4

Para el caso 6.1.4 se observa un número alto de features (685), el dataset presenta una alta complejidad. La ligera mejora del lift de 1.61 a 1.64 sugiere que la data aumentada puede haber proporcionado más variabilidad y ejemplos adicionales que ayudan al modelo a capturar mejor las features importantes. Sin embargo, la alta dimensionalidad puede también haber limitado la capacidad de la data sintética para mejorar significativamente el lift, ya que puede ser difícil para los algoritmos de aumento de datos generar ejemplos sintéticos que sean verdaderamente representativos en un espacio de características tan grande y complejo. Además, el riesgo de introducir ruido aumenta con la dimensionalidad, lo que puede contrarrestar los beneficios de tener más ejemplos sintéticos.

La alta dimensionalidad no sólo complica la generación de datos sintéticos, sino que también aumenta considerablemente los tiempos de entrenamiento y validación del modelo. Estos tiempos se traducen en mayores costos computacionales y pueden limitar la viabilidad de probar configuraciones más complejas o exhaustivas. Si bien es cierto, al usar algoritmos tradicionales los tiempos siguen siendo manejables, los algoritmos como CTGAN o TVAE aumentan considerablemente sus tiempos de ejecución, y en este caso CTGAN es el que presenta los mejores resultados, por lo que en este tipo de bases de datos comienza a aparecer el trade-off entre mejora en el KPI y tiempo de ejecución.

6.2.5. Caso 5 y 6

El caso de 6.1.5 presenta una alta dimensionalidad y un gran volumen de datos. El desbalanceo extremo (97.01 % clase 0, 2.98 % clase 1) y la gran cantidad de ejemplos (1097749) plantean algunas dificultades en la generación sintética de datos.

El dataset del caso 6.1.6 tiene una alta dimensionalidad (201 características) y un desbalanceo extremadamente alto (99.6 % clase 0, 0.4 % clase 1), con una gran cantidad de ejemplos (4939512). Un punto a destacar es que en este caso se disminuyó el espacio de búsqueda y se excluyeron algoritmos.

La leve disminución del lift, de 3.96 a 3.94 (Dataset 5), y la disminución significativa, de 4.04 a 3.54 (Dataset 6), puede ser explicada por varios factores:

Con más de 200 features se tiene el problema de la *maldición de dimensionalidad*, es decir, el espacio de características se vuelve muy grande y disperso, dificultando encontrar patrones significativos, tal como ocurre en el caso 6.1.4. La data aumentada podría no proporcionar suficiente información adicional relevante. Además, podría ser difícil agregar ejemplos que mantengan la estructura de los datos reales sin agregar variabilidad no deseada.

En datasets grandes como este, agregar data sintética puede llevar a que el modelo se sobreajuste a esta data, especialmente si la data aumentada no es de alta calidad o no representa bien la distribución real. El sobreajuste a la data sintética puede resultar en un peor rendimiento cuando el modelo se aplica a datos reales.

Al agregar datos sintéticos se puede introducir ruido o ejemplos irrelevantes que pueden confundir al modelo en lugar de ayudarlo. En datasets con muchas características, este problema se agrava, ya que es más probable que se generen ejemplos sintéticos con combinaciones de características que no son representativas de la realidad de los datos.

Tener una dimensión del orden de millones de filas, hace que la proporción de datos sintéticos a datos reales puede ser un factor determinante. Si la cantidad de data sintética generada es pequeña en comparación con la data real, su impacto puede ser insignificante. Por otro lado, si es demasiado grande, puede dominar el aprendizaje del modelo, especialmente si no es de alta calidad.

En datasets muy grandes, el tiempo de ejecución y los costos computacionales aumentan significativamente (tal como se expone en las tablas y) , especialmente cuando se utilizan instancias de GCP. Para gestionar estos recursos, es necesario reducir el número de épocas de entrenamiento de las redes generativas y limitar los cálculos en la validación cruzada. Incluso, en el caso 6.1.6, fue necesario excluir los métodos del área de deep learning, debido a los largos tiempos de ejecución y costos prohibitivos, lo cual afectó negativamente la posible mejora de la data sintética y, en consecuencia, el rendimiento del modelo.

El trade-off entre la reducción de tiempos de ejecución y la calidad de los datos sintéticos puede llevar a una generación menos precisa y menos representativa. Esto afecta la capacidad del modelo para mejorar su rendimiento, ya que no sólo se ve limitado por los recursos

computacionales, sino también por la necesidad de equilibrar estos aspectos para evitar degradaciones significativas en el rendimiento.

6.2.6. Reflexión final

La efectividad de la data aumentada depende de varios factores, incluyendo el desbalanceo del conjunto de datos, el número de features, la cantidad de registros y la calidad de los datos sintéticos generados. En datasets con menos features y un número moderado de ejemplos, como el caso 6.1.1 con 9 features y 6594 registros, la data aumentada puede ser más efectiva debido a la simplicidad y menor complejidad del espacio de características. En contraste, en datasets con alta dimensionalidad, gran volumen de datos y un gran número de ejemplos, como en el Dataset 5 ó 6, los beneficios de la data aumentada pueden ser limitados o incluso negativos debido a la maldición de la dimensionalidad, el riesgo de sobreajuste, la introducción de ruido, la proporción de datos sintéticos a datos reales y el trade-off entre tiempo, gasto y calidad. Por lo tanto, es crucial evaluar cuidadosamente el impacto de la data aumentada en cada caso específico y ajustar los parámetros del algoritmo de generación de datos sintéticos para minimizar estos efectos negativos y mejorar el rendimiento del modelo.

De esta forma, según lo anterior, en casos donde ya se dispone de un volumen significativo de datos reales que cubren bien el espacio de entrada, agregar datos sintéticos no proporciona beneficios adicionales. Los modelos ya han capturado los patrones existentes de manera efectiva. Además, en datasets que contienen relaciones complejas y no lineales que no son fáciles de replicar, el motor puede generar datos sintéticos que no capturan adecuadamente la complejidad de las interacciones. Esto puede llevar a un rendimiento neutro o incluso a la degradación del modelo, ya que los datos sintéticos no contribuyen a una mejor representación del espacio de características. Finalmente, se puede afirmar que en conjuntos de datos complejos, la generación de datos sintéticos puede introducir ruido adicional que no corresponde a la estructura real de los datos, lo que puede confundir al modelo en lugar de ayudarlo a generalizar mejor.

Capítulo 7

Conclusiones

El desarrollo e implementación del motor de aumento de datos representa un avance al abordar uno de los problemas críticos en el aprendizaje automático, que es la escasez de datos y el desbalance de clases. A través de la integración de técnicas avanzadas de machine learning y deep learning, incluidas las Redes Generativas Adversarias (GANs), este motor ha demostrado ser una herramienta valiosa para la generación de datos sintéticos que pueden potenciar la capacidad predictiva de modelos.

El motor de aumento de datos se diseñó con una arquitectura modular, lo que permite la incorporación de múltiples técnicas de generación de datos sintéticos. Esta versatilidad no sólo facilita la experimentación con diferentes enfoques, sino que también permite su adaptación a una amplia gama de problemas. La capacidad del motor para integrar métodos tradicionales de machine learning con técnicas avanzadas de deep learning asegura que pueda abordar tanto conjuntos de datos simples como aquellos de mayor complejidad.

Los experimentos realizados con diversos conjuntos de datos han demostrado que el uso de datos sintéticos generados por el motor puede conducir a mejoras en la capacidad de generalización de los modelos de aprendizaje automático en diferentes escenarios. Sin embargo, en algunos casos, la generación de datos sintéticos no proporcionó mejoras claras e incluso resultó en una disminución del rendimiento. Esto se observó principalmente en conjuntos de datos muy complejos donde los patrones subyacentes son difíciles de replicar sintéticamente, lo que sugiere que la calidad y la naturaleza del dataset original juegan un papel crucial en la efectividad del aumento de datos, entrando en juego lo que es el tiempo de ejecución, el costo computacional y calidad de los datos según los algoritmos utilizados.

La implementación del motor se basa en bibliotecas de código abierto, lo que garantiza la transparencia, integridad y reproducibilidad del sistema. Este facilita la adaptación y mejora del motor en futuros trabajos. La utilización de herramientas y frameworks ampliamente reconocidos en la comunidad científica asegura que el motor pueda ser integrado fácilmente en diferentes pipelines de desarrollo y experimentación.

Para probar y evaluar la eficacia del motor de aumento de datos, se pueden utilizar diversas fuentes de datos disponibles públicamente. Algunas de estas fuentes incluyen Kaggle, una plataforma que ofrece una amplia gama de conjuntos de datos en diversos dominios, incluidos competiciones donde se proporcionan datos de entrenamiento y prueba; el UCI Machine

Learning Repository, conocido por su extensa colección de conjuntos de datos de diferentes campos de estudio; Google Dataset Search, un motor de búsqueda que ayuda a encontrar conjuntos de datos disponibles públicamente en la web; y OpenML, una plataforma que proporciona conjuntos de datos, algoritmos y experimentos listos para ser utilizados y comparados en investigaciones de aprendizaje automático.

La realización de esta memoria permitió la aplicación de una amplia gama de conocimientos adquiridos en el ámbito del aprendizaje automático y el deep learning. Se puso en práctica la aplicación de redes GAN y otros algoritmos de generación de datos sintéticos, además de técnicas de preprocesamiento de datos, diseño de pipelines de machine learning, y evaluación de modelos. Esta memoria también requirió una comprensión de las arquitecturas de redes neuronales pertenecientes al área de inteligencia artificial generativa, la programación en Python, y el uso de bibliotecas de código abierto como Scikit-learn y SDV.

7.1. Trabajo futuro

La incorporación de nuevas técnicas y la optimización del rendimiento en diferentes contextos podrían potenciar aún más las capacidades del motor. En particular, la exploración de nuevos algoritmos de deep learning y la combinación de técnicas de aumento de datos con enfoques de aprendizaje activo representan los siguientes pasos en la evolución del motor. Además, es posible aplicar el motor en diferentes áreas como podría ser la medicina, la ciberseguridad, y otros campos críticos.

Un área clave para futuras investigaciones es la mejora de la calidad de los datos sintéticos generados. Esto incluye el desarrollo de técnicas más sofisticadas para la evaluación de la calidad de los datos sintéticos y la integración de métodos de validación cruzada más robustos. Asimismo, la adaptación del motor para trabajar con datos no estructurados, como texto e imágenes, representa una oportunidad emocionante para ampliar su aplicabilidad.

Otro punto a destacar como posible trabajo, es optimizar lo mayor posible el tiempo de ejecución al tener dataset muy grandes, de esta forma, se podrá ampliar los espacios de búsquedas y las restricciones de los algoritmos serán menores.

En conclusión, el desarrollo e implementación de este motor de aumento de datos no sólo proporciona una solución efectiva a la escasez de datos, sino que también establece una base sólida para futuras investigaciones y aplicaciones en el campo del aprendizaje automático. La capacidad del motor para mejorar el rendimiento de los modelos y su potencial para ser adaptado a diferentes dominios subraya su importancia y relevancia en el avance de la inteligencia artificial y la ciencia de datos.

Bibliografía

[1] Plasencia Moreno, Lieter y Anías Calderón, C., "Arquitectura referencial de Big Data para la gestión de las telecomunicaciones," Ingeniare. Revista chilena de ingeniería, vol. 25, pp. 566 – 577, 2017, doi:http://dx.doi.org/10.4067/S0718-33052017000400566.

[2] Escolano Ruiz, F., Inteligencia artificial : Modelos, tecnicas y areas de aplicacion. Thomson, 2003.

[3] Alloghani, M., Al-Jumeily, D., Mustafina, J., Hussain, A., y Aljaaf, A. J., A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science. 2020, doi:10.1007/978-3-030-22475-2_1.

[4] Ferreira, A. J. y Figueiredo, M. A. T., Boosting Algorithms: A Review of Methods, Theory, and Applications, pp. 35–85. Springer New York, 2012, doi:10.1007/978-1-4419-9326-7_2.

[5] Goodfellow, I., Bengio, Y., y Courville, A., Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Schmidhuber, J., "Deep learning in neural networks: An overview," Neural Networks, vol. 61, pp. 85–117, 2015, doi:https://doi.org/10.1016/j.neunet.2014.09.003.

[7] Ibm, "Ai vs. machine learning vs. deep learning vs. neural networks: What's the difference?," 2021, https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks.

[8] Krawczyk, B., Learning from imbalanced data: open challenges and future directions, vol. 5. 2016, doi:https://doi.org/10.1007/s13748-016-0094-0.

[9] Figueira, A. y Vaz, B., "Survey on synthetic data generation, evaluation methods and gans," Mathematics, vol. 10, 2022, doi:10.3390/math10152733.

[10] Nikolenko, S. I., "Synthetic data for deep learning," 2019.

[11] Chawla, N. V., Bowyer, K. W., Hall, L. O., y Kegelmeyer, W. P., "SMOTE: Synthetic minority over-sampling technique," Journal of Artificial Intelligence Research, vol. 16, pp. 321–357, 2002, doi:10.1613/jair.953.

[12] López, F., "Smote: Synthetic data augmentation for tabular data," 2021, https://towardsdatascience.com/smote-synthetic-data-augmentation-for-tabular-data-1ce28090debc.

[13] Han, H., Wang, W.-Y., y Mao, B.-H., "Borderline-smote: A new over-sampling method in imbalanced data sets learning," pp. 321–357, 2005, doi:https://doi.org/10.1007/11538059_91.

[14] Bunkhumpornpat, C., Sinapiromsaran, K., y Lursinsap, C., "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem,"

pp. 475–482, Springer Berlin Heidelberg, 2009, https://api.semanticscholar.org/CorpusID:9477920.

[15] He, H., Bai, Y., Garcia, E. A., y Li, S., "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," pp. 1322–1328, 2008 IEEE International Joint Conference on Neural Networks, 2008, doi:10.1109/IJCNN.2008.4633969.

[16] Douzas, G., Bacao, F., y Last, F., "Improving imbalanced learning through a heuristic oversampling method based on k-means and smote," Information Sciences, vol. 465, pp. 1–20, 2018, doi:https://doi.org/10.1016/j.ins.2018.06.056.

[17] Jo, T. y Japkowicz, N., "Class imbalances versus small disjuncts," vol. 6, no. 1, p. 40–49, 2004, doi:10.1145/1007730.1007737.

[18] Learn, S., "Gaussian mixture models," 2023, https://scikit-learn.org/stable/modules/mixture.html.

[19] Chokwitthaya, C., Zhu, Y., Mukhopadhyay, S., y Jafari, A., Applying the Gaussian Mixture Model to Generate Large Synthetic Data from a Small Data Set, pp. 1251–1260, doi:10.1061/9780784482865.132.

[20] Ibm, "What are convolutional neural networks?," 2024, https://www.ibm.com/topics/convolutional-neural-networks.

[21] Lipton, Z. C., "A critical review of recurrent neural networks for sequence learning," ArXiv, 2015, doi:https://doi.org/10.48550/arXiv.1506.00019.

[22] Ibm, "What are recurrent neural networks?," 2021, https://www.ibm.com/topics/recurrent-neural-networks.

[23] Bengio, Y., Simard, P., y Frasconi, P., "Learning long-term dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 1994, doi:10.1109/72.279181.

[24] Hochreiter, S. y Schmidhuber, J., "Long Short-Term Memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997, doi:10.1162/neco.1997.9.8.1735.

[25] Rumelhart, D. E., Hinton, G. E., y Williams, R. J., "Learning representations by back-propagating errors," Nature, vol. 323, pp. 533–536, 1986, doi:10.1038/323533a0.

[26] LeCun, Y., Bengio, Y., y Hinton, G., "Deep learning," Nature, vol. 521, p. 436–444, 2015, doi:10.1038/nature14539.

[27] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., y Polosukhin, I., "Attention is all you need," CoRR, 2017, doi:https://doi.org/10.48550/arXiv.1706.03762.

[28] Lu, Y., Shen, M., Wang, H., y Wei, W., "Machine learning for synthetic data generation: A review," 2023, doi:10.48550/arXiv.2302.04062.

[29] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., y Bengio, Y., "Generative adversarial networks," 2014, doi:https://doi.org/10.48550/arXiv.1406.2661.

[30] Stewart, M., "Comprehensive introduction to autoencoders," 2019, https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368.

[31] Kingma, D. P. y Welling, M., "Auto-encoding variational bayes," 2022, doi:https://doi.

org/10.48550/arXiv.1312.6114.

[32] Google, "Classification: Accuracy, recall, precision, and related metrics," 2024, https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

[33] Tufféry, S., Data Mining and Statistics for Decision Making, cap. 11, pp. 301–553. John Wiley Sons, Ltd, 2011, doi:https://doi.org/10.1002/9780470979174.ch11.

[34] Lee, T., Kim, M., y Kim, S.-P., "Data augmentation effects using borderline-smote on classification of a p300-based bci," 2020, doi:10.1109/BCI48061.2020.9061656.

[35] Lu, C., Lin, S., Liu, X., y Shi, H., "Telecom fraud identification based on adasyn and random forest," 2020, doi:10.1109/ICCCS49078.2020.9118521.

[36] Aditsania, A., Adiwijaya, y Saonard, A. L., "Handling imbalanced data in churn prediction using adasyn and backpropagation algorithm," 2017, doi:10.1109/ICSITech.2017.8257170.

[37] Sarkar, S., Pramanik, A., Maiti, J., y Reniers, G., "Predicting and analyzing injury severity: A machine learning-based approach using class-imbalanced proactive and reactive data," Safety Science, vol. 125, p. 104616, 2020, doi:https://doi.org/10.1016/j.ssci.2020.104616.

[38] Zhang, J., Cormode, G., Procopiuc, C. M., Srivastava, D., y Xiao, X., "Privbayes: Private data release via bayesian networks," vol. 42, no. 4, 2017, doi:10.1145/3134428.

[39] Solatorio, A. V. y Dupriez, O., "Realtabformer: Generating realistic relational and tabular data using transformers," 2023, doi:https://doi.org/10.48550/arXiv.2302.02041.

[40] Xu, L. y Veeramachaneni, K., "Synthesizing tabular data using generative adversarial networks," CoRR, 2018, doi:https://doi.org/10.48550/arXiv.1811.11264.

[41] Xu, L., Skoularidou, M., Cuesta-Infante, A., y Veeramachaneni, K., "Modeling tabular data using conditional gan," en Advances in Neural Information Processing Systems, 2019.

[42] Rajabi, A. y Garibay, O. O., "Tabfairgan: Fair tabular data generation with generative adversarial networks," Machine Learning and Knowledge Extraction, vol. 4, no. 2, pp. 488–501, 2022.

[43] Arjovsky, M., Chintala, S., y Bottou, L., "Wasserstein generative adversarial networks," vol. 70 de Proceedings of Machine Learning Research, pp. 214–223, 2017, doi:https://doi.org/10.48550/arXiv.1701.07875.

[44] Roques, A. y Contributors, P., "PlantUML Software.", https://github.com/plantuml/plantuml.

[45] "Mermaid: Generate diagrams from markdown-like text.", https://mermaid.js.org/.

[46] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., y Chintala, S., "Pytorch: An imperative style, high-performance deep learning library," en Advances in Neural Information Processing Systems 32, pp. 8024–8035, Curran Associates, Inc., 2019, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[47] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., y Duchesnay, E., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[48] Chen, T. y Guestrin, C., "XGBoost: A scalable tree boosting system," en Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794, ACM, 2016, doi:10.1145/2939672.2939785.

[49] Patki, N., Wedge, R., y Veeramachaneni, K., "The synthetic data vault," pp. 399–410, 2016, doi:10.1109/DSAA.2016.49.

Anexos

Anexo A. Comparación de features

Las figura expuestas a continuación, muestran la probabilidad acumulativa y la densidad de probabilidad comparada entre características originales y su versión aumentada. Los gráficos van exponiendo 4 características escogidas al azar del total de features del conjunto 3.1. Cada conjunto de 4 características serán enumeradas progresivamente.

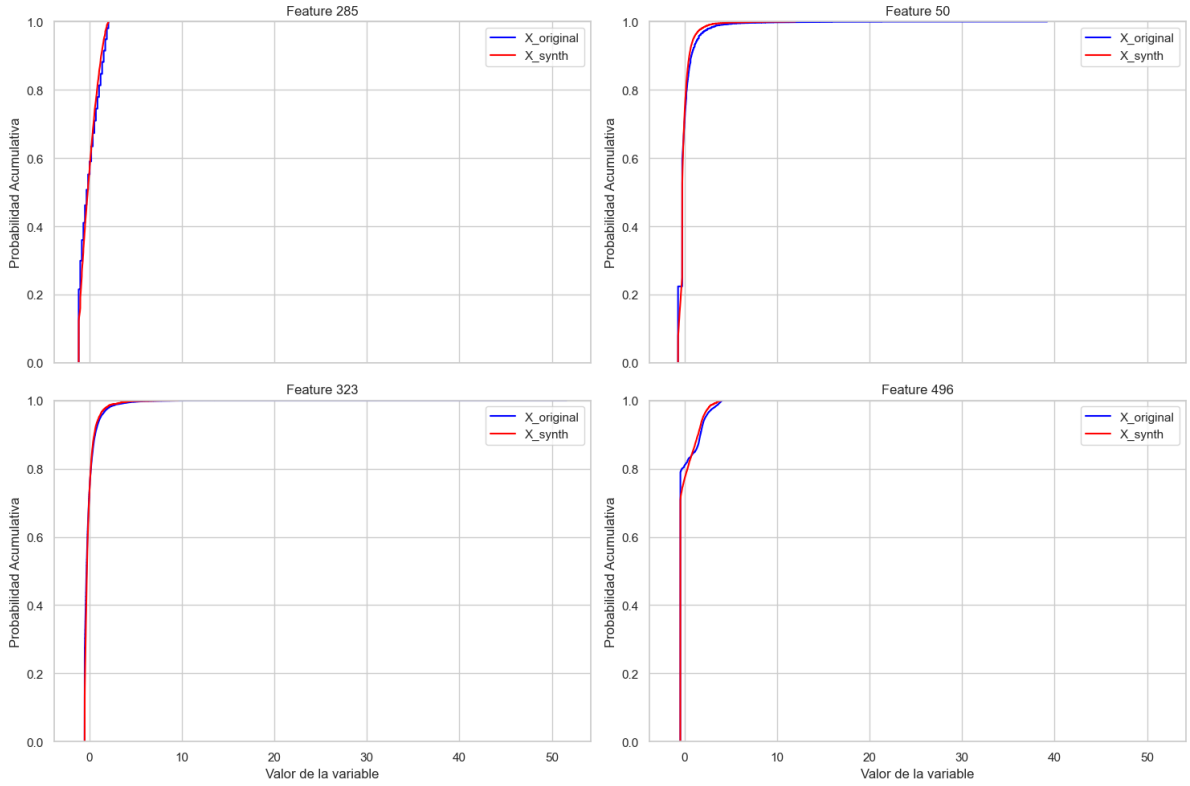

Figura A.1: Probabilidad acumulativa $X_{original}$ vs X_{synth} SMOTE; Features 2

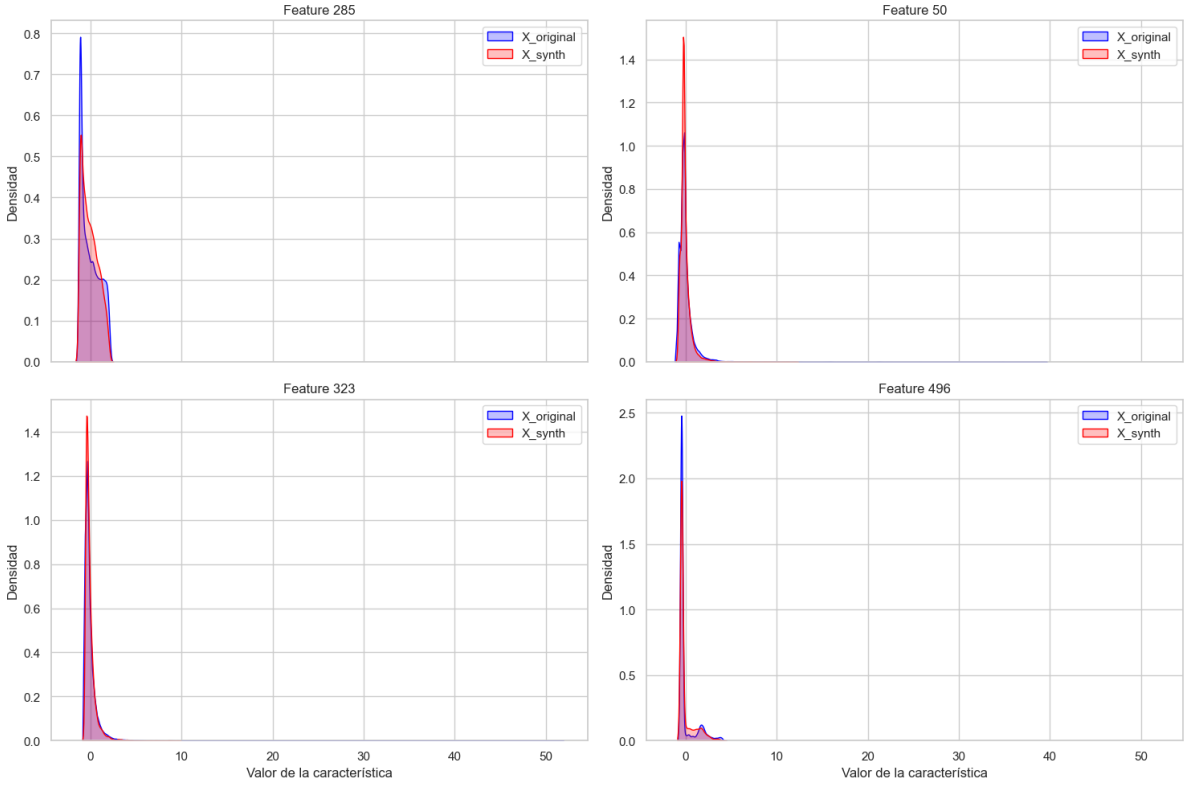

Figura A.2: Densidad de probabilidad $X_{original}$ vs X_{synth} SMOTE; Features 2

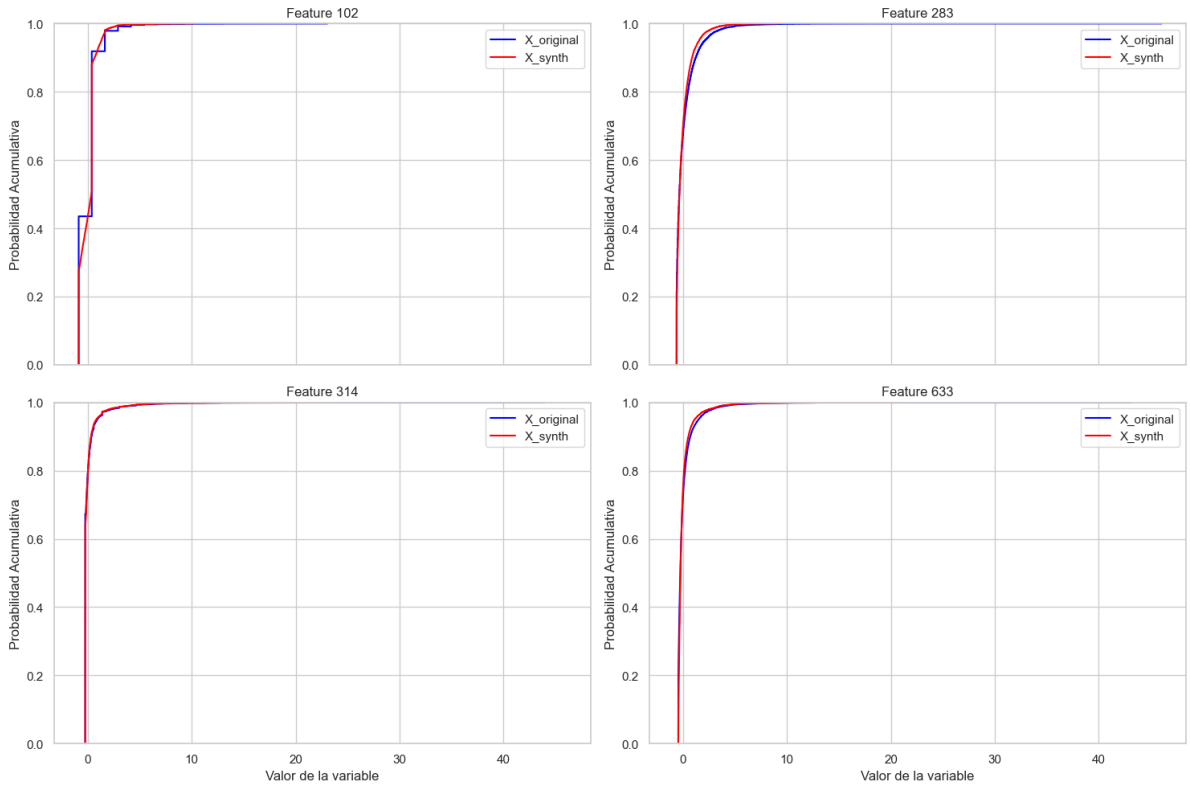

Figura A.3: Probabilidad acumulativa $X_{original}$ vs X_{synth} ADASYN; Features 3

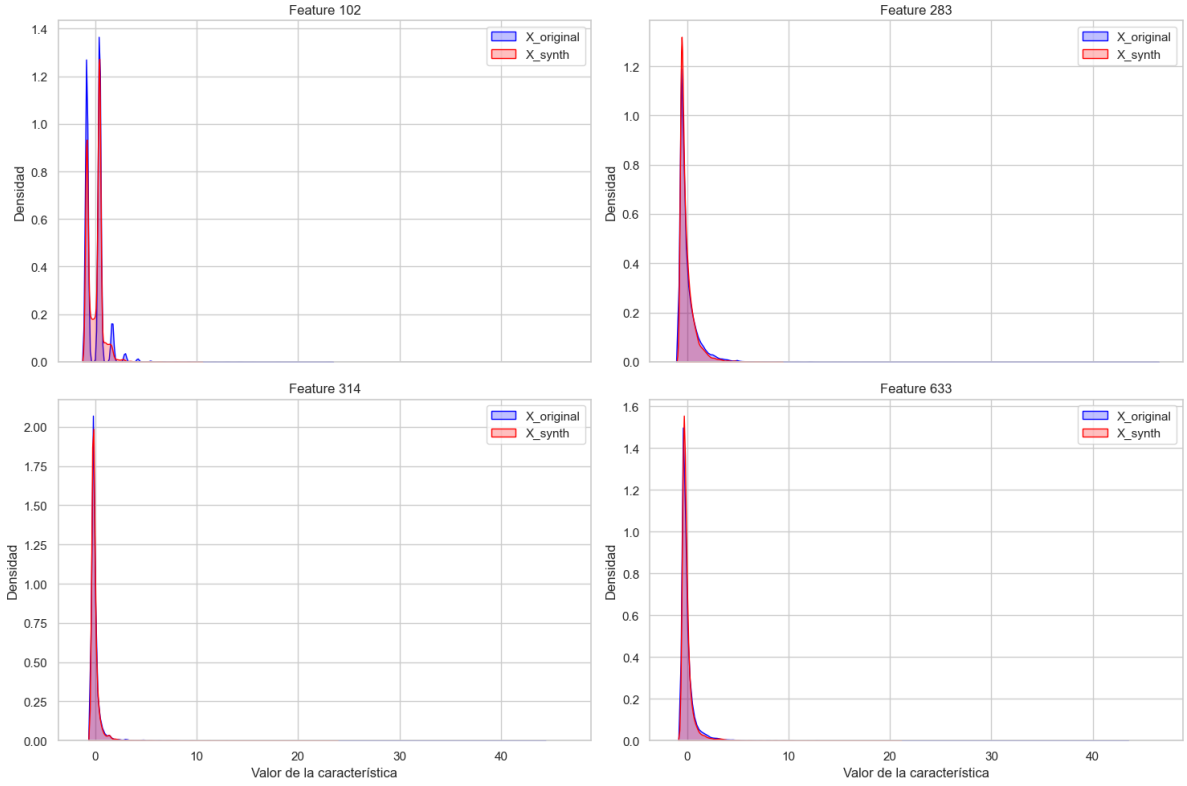

Figura A.4: Densidad de probabilidad $X_{original}$ vs X_{synth} ADASYN; Features 3

Anexo B. Modo de uso

DataAugmentationEngine es una biblioteca de Python para generar datos de manera sintética. La bibliteca permite tratar con problemas binarios, remuestrando la clase minoritaria a través de la utilización de diferentes técnicas de aumento de datos.

B.1. Instalación

Se recomienda fuertemente usar máquinas virtuales con alta disponibilidad de vCPU. Se puede clonar el repositorio del paquete dentro de una máquina virtual.

```
DAEngine
├── notebook.ipynb
└── data_augmentation_engine
    ├── ___init___.py
    ├── engine.py
    ├── gan_wrapper.py
    ├── utils.py
    └── setup.py
```

Se debe posicionar dentro del directorio de *data_augmentation_engine*

```
cd DAEngine
cd data_augmentation_engine
```

Luego, basta con que se ejecute el comando

```
pip install .
```

Finalmente, reinicia el kernel del entorno.

B.2. Uso

Aquí hay un ejemplo de cómo utilizar la biblioteca en un notebook.ipynb

Código B.1: Ejemplo de uso 1

```
from data_augmentation_engine.engine import DataAugmentationEngine

# se cargan los datos
data = ...

# se crea una instancia del motor de aumento de datos
da_engine = DataAugmentationEngine(data)

# se aplica el motor
data_aumentada = da_engine.fit_resample(data)
```

El siguiente ejemplo muestra un caso particular en donde el usuario desea utilizar el algoritmo BorderlineSMOTE y además desea optimizarlo.

Código B.2: Ejemplo de uso 2

```
from data_augmentation_engine.engine import DataAugmentationEngine

# se cargan los datos
data = ...

# se crea una instancia del motor de aumento de datos
da_engine =  DataAugmentationEngine(data, algorithm_name='BorderlineSMOTE',
    ↪ optimize_algorithm=True)

# se aplica el motor
data_aumentada = da_engine.fit_resample(data)
```

En este último ejemplo se muestra el caso en que el usuario pone una lista de algoritmos para poder comparar entre ellos.

Código B.3: Ejemplo de uso 3

```
from data_augmentation_engine.engine import DataAugmentationEngine

# se cargan los datos
data = ...

# se crea una instancia del motor de aumento de datos
da_engine =  DataAugmentationEngine(data, search_algorithm=True, algorithms=['
    ↪ ADASYN', 'BorderlineSMOTE'])

# se aplica el motor
data_aumentada = da_engine.fit_resample(data)
```