UNIVERSIDAD DE CÓRDOBA

GRADO EN INGENIERÍA INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR

Trabajo fin de Grado

Creación de patrones sintéticos para conjuntos de datos desbalanceados mediante la metaheurística voraz iterativa.

Manual técnico

***Autor**:*

Francisco Javier Maestre García

Directores

D. Pedro Antonio Gutiérrez Peña

D. Carlos García Martínez

Enero, 2017

UNIVERSIDAD DE CÓRDOBA

GRADO EN INGENIERÍA INFORMÁTICA
ESCUELA POLITÉCNICA SUPERIOR

Trabajo fin de Grado

Creación de patrones sintéticos para conjuntos de datos desbalanceados mediante la metaheurística voraz iterativa.

Manual técnico

***Autor**:*

Francisco Javier Maestre García

Directores

D. Pedro Antonio Gutiérrez Peña

D. Carlos García Martínez

Enero, 2017

Datos del proyecto

Título: Creación de patrones sintéticos para conjuntos de datos desbalanceados mediante la metaheurística voraz iterativa.

Autor: *Francisco Javier Maestre García*
Email: *I12magaf@uco.es*
***DNI**: 31012172-F*
***Especialidad**: Grado en ingeniería informática*
***Directores del proyecto**:*

- D. Pedro Antonio Gutiérrez Peña
- D. Carlos García Martínez

Firma del autor y directores

El autor:

Fdo.: Francisco Javier Maestre García

Los directores:

Fdo.: Pedro Antonio Gutiérrez Peña

Fdo.: Carlos García Martínez

AGRADECIMIENTOS

Índice

PARTE VI. Apéndices

Índice de Tablas

Índice de Figuras

Índice de Gráficos

Índice de Códigos

Parte I
Introducción

Capítulo 1:
Introducción

Con la continua expansión de la disponibilidad de los datos y la facilidad de su obtención (muchas veces a gran escala), es normal ver cómo se ha incrementado proporcionalmente el uso y desarrollo de algoritmos de aprendizaje automático o toma de decisiones.

Dichos algoritmos se basan, generalmente, en una fase de aprendizaje utilizando un conjunto de datos con gran cantidad de instancias, donde cada una de ellas posee una etiqueta inequívoca de pertenencia a un tipo de dato o clase. Tras aprender de los datos existentes, estos algoritmos son capaces, con mayor o menor precisión, de predecir la etiqueta de nuevas instancias y clasificarlas automáticamente en alguna de las clases estudiadas en el proceso de aprendizaje del algoritmo.

Para este tipo de algoritmos existe un problema recurrente identificado como *Imbalanced Learning* o aprendizaje desbalanceado [1]. En líneas generales, el aprendizaje desbalanceado ocurre cuando uno o varios tipos de datos (clases mayoritarias) dominan con mucha diferencia la distribución del conjunto en comparación con otros tipos de datos (clases minoritarias).

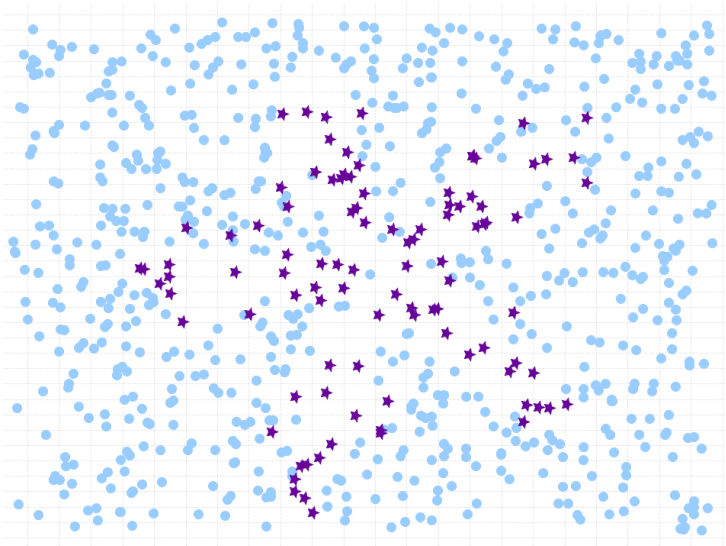

Figura 1: Representación gráfica de base de datos desbalanceada con dos clases

Los problemas de clasificación con conjuntos que presentan una distribución desigual en las muestras entre las diferentes clases, plantean un reto importante para cualquier método de clasificación, ya que este hecho complica el aprendizaje de las instancias de las clases minoritarias. Dicho problema tiene que ver con el rendimiento de los algoritmos en presencia de datos con poca representación, en comparación con la gran representación que poseen la clase o clases mayoritarias, debido a que los algoritmos, normalmente, no tienen en cuenta la desproporción entre las clases o usan un coste de clasificación errónea simétrico.

Este tipo de problemas se da con mayor frecuencia en conjuntos con solo dos clases, donde la información más importante se encuentra tras los individuos más raros o menos comunes, como por ejemplo, personas con una determinada enfermedad frente a personas sanas, u operaciones bancarias fraudulentas frente a las operaciones legales. Estas diferencias pueden llegar a ratios de desbalanceo de 1:1000 o incluso más, y lo más importante es obtener la mejor clasificación posible para los individuos minoritarios. Por convenio, se utiliza la etiqueta positiva para referirse a la clase minoritaria y negativa para referirse a la mayoritaria.

Debido a las características inherentes a los conjuntos de datos desequilibrados, ha sido necesario investigar nuevas técnicas y herramientas con el objetivo de corregir o evitar este problema. Algunos de estos métodos se basan en la idea de realizar un sobremuestreo a través de la generación de patrones sintéticos que se añadirían a las clases minoritarias para equilibrar la distribución de las muestras de las diferentes clases. Por ejemplo SMOTE [2], donde un patrón sintético es un individuo virtual creado a partir de patrones reales de una misma clase utilizando una interpolación. Por tanto, dicho patrón sintético posee unas características ficticias que potencialmente representan a un individuo real.

Las técnicas basadas en la generación de patrones sintéticos de la clase minoritaria pueden llegar a producir una cantidad suficiente de sintéticos, para hacer que su distribución sea simétrica, igualando la presencia de la clase minoritaria a la de la mayoritaria. Una vez realizado este preprocesamiento del conjunto de datos, y que se haya igualado la distribución de las clases, podría aplicarse cualquier método de clasificación sin que su resultado se vea afectado por la diferencia en la distribución, como si lo haría antes de la inclusión de patrones sintéticos. La inclusión de patrones sintéticos en una zona equivocada, podría no solo no ayudar al clasificador, sino dificultar gravemente al proceso de clasificación, por tanto, es muy importante hacer uso de alguna técnica para evitar que un sintético sea generado en una zona no adecuada, como se intenta en las propuestas ADASYN [3] o MWMOTE [4].

Por otro lado, las metaheurísticas son algoritmos de optimización aproximada, que proporcionan soluciones de cierta calidad a problemas en los que buscar la solución óptima no es posible. Éstas se han aplicado a numerosos problemas del mundo real y entre ellos, a problemas de aprendizaje automático como el entrenamiento de redes neuronales, la selección de prototipos, la detección de reglas de asociación, etc.

En este proyecto, nos planteamos abordar el problema de la generación de patrones sintéticos para conjuntos de datos desbalanceados mediante una metaheurística. En particular, centraremos nuestra atención en la metaheurística voraz iterativa, o en inglés "*Iterated Greedy*". Su objetivo será, por tanto, la generación de patrones sintéticos que resulten posteriormente en la inducción de clasificadores con buenos comportamientos sobre este tipo de conjuntos de datos desbalanceados.

Iterated Greedy parte de una solución válida y consta de dos fases:

1. Fase de destrucción: La solución es destruida de forma parcial y aleatoria
2. Fase de reconstrucción: La solución es reconstruida guiada en base a la parte superviviente a la fase de destrucción.

Por tanto, iterando con una metaheurística como es *Iterated Greedy*, se busca crear un conjunto de sintéticos más óptimo y con garantías de contribuir positivamente en la clasificación final.

Así pues el objetivo de este proyecto sería definir el remuestreo de patrones para bases de datos no balanceadas como un problema de optimización y utilizar una metaheurística, como es *Iterated Greedy*, para su resolución, con el fin de que dichos patrones aporten la máxima información útil y mejoren la clasificación obtenida con respecto a otros algoritmos donde la generación de patrones sintéticos es determinista o puramente aleatoria.

Capítulo 2:
Definición del problema

En este apartado se pretende dar una definición concisa del problema abordado en este proyecto. A continuación, identificaremos las necesidades y establecemos los objetivos del proceso general del Trabajo Fin de Grado (TFG). Dividiremos, por tanto, este apartado del manual en dos subapartados:

- **Identificación del problema real:** El problema se define desde el punto de vista del usuario final.
- **Identificación del problema técnico:** El problema se define desde el punto de vista del desarrollador y proyectista.

2.1. Identificación del problema real

El problema real que se plantea para este Trabajo Fin de Grado es, por una parte, la implementación de un algoritmo de remuestreo utilizando técnicas de over-sampling, inspirándose en los algoritmos existentes hasta la fecha, aplicándole además, algunas mejoras que se creen necesarias para hacerlo más eficiente o eficaz. Por otra parte, se implementará la metaheurística voraz iterativa para optimizar la creación de los patrones sintéticos, de forma que, se consiga generar un conjunto de sintéticos que aporte el mayor beneficio posible a la clasificación. En resumen, este Trabajo Fin de Grado, se divide en dos partes:

- **Algoritmo de remuestreo:** Se encargará de generar una serie de patrones sintéticos pertenecientes a la clase minoritaria, y que se unirán a la base de datos durante la fase de aprendizaje del algoritmo de clasificación , de tal forma, que se consiga un mejor resultado en el conjunto de test o al clasificar nuevas instancias.

- **Metaheurística voraz iterativa:** Encargada de evolucionar el conjunto de sintéticos generado inicialmente hasta otro que ostente un mayor aporte de información al proceso de aprendizaje y clasificación. Esta metaheurística constará de una fase de destrucción aleatoria y parcial del conjunto de sintéticos y una reconstrucción guiada para llevar a cabo dicha evolución. Para asegurar la convergencia se utilizará un conjunto de validación.

El problema real se divide, por tanto, en cuatro partes bien diferenciadas:

- Diseño e implementación de un algoritmo de remuestreo.
- Diseño e implementación de una metaheurística voraz iterativa para el proceso de generación de sintéticos.
- Entrenamiento de un árbol de decisión uniendo los sintéticos con el conjunto de entrenamiento.
- Realizar una clasificación del conjunto de *Test*, y analizar el resultado.

2.2. Problema técnico

Una vez que ha sido definido el problema real, se pasará a realizar una definición técnica del problema que este TFG pretende resolver. Esta definición se va a realizar según la metodología PDS *(Product Design Specification*).

2.2.1. Funcionamiento

La herramienta software que se pretende desarrollar será capaz de generar un conjunto de patrones sintéticos para apoyar el aprendizaje de la clase minoritaria, y así mejorar la clasificación obtenida en el conjunto de test y nuevas instancias.

Las base de datos será leída en formato CSV (*Comma separated values*). Una vez sea almacenada, se dividirá en dos partes, *inputs y outputs,* y estas a su vez en tres subconjuntos más que son *Train, Test* y *Validation*, con un tamaño porcentual configurable por el usuario. El conjunto de *Train* será utilizado para generar los patrones sintéticos, el conjunto de validación se utilizará para guiar la convergencia durante la evolución que realiza la metaheurística *Iterated Greedy* y finalmente el conjunto de *Test* será utilizado para evaluar la clasificación conseguida con un determinado clasificador para el conjunto de patrones reales y sintéticos. Con el objetivo de poder realizar un análisis más fiable de los datos finales, el sistema se ejecutara repetidas veces y se calculará la media aritmética resultante de todas las ejecuciones para algunas medidas de precisión de la clasificación, como podrían ser Precisión o G-mean. Dicha función podrá mostrar, tanto los resultados por pantalla, como volcarlos en un CSV con el nombre que desee el usuario.

De forma complementaria se añadirán algunas funciones opcionales pero útiles para el correcto funcionamiento del algoritmo, como una función que busca elementos duplicados en la base de datos, otra que puede imprimir una gráfica con los valores obtenidos durante la evolución de la metaheurística, o una más que identifica y elimina los patrones considerados como ruido.

Los pasos más generales del funcionamiento de la herramienta serán:

- **Introducción de datos:** El usuario deberá proporcionar una base de datos como argumento en el momento de la ejecución y sobre la que se aplicará el algoritmo. Esta base de datos deberá estar en formato CSV.

- **Preparación de los datos:** Una vez cargada la base de datos se procederá a prepararla para poder ser utilizada por el algoritmo:

    1º. Se separarán las entradas (características de los patrones) y las salidas (etiquetas de clase) en dos conjuntos llamados, *inputs* y *outputs*.

2º. En el caso de que el conjunto original tenga más de dos clases, será necesario realizar una restructuración en las etiquetas de clase (*outputs*). Estas etiquetas pasarán a identificarse como pertenencia a la "clase minoritaria" (positiva) o a la "clase mayoritaria" (negativa), de modo que al final del proceso solo existan dos clases**.** Esto es debido a que nuestra aplicación solo generará patrones sintéticos a partir de la nueva definición de "clase minoritaria". Esta transformación requiere una configuración por parte del usuario, ya que la "clase minoritaria" podría estar formada por una o más clases del conjunto original.

3º. Tanto las entradas como las nuevas salidas se dividirán de forma estratificada en 3 subconjuntos *Train, Test* y *Validation*, manteniendo la misma proporción de las clases en cada subconjunto y sin repetir elementos entre ellos. El tamaño porcentual podrá ser elegido por el usuario, pero la división será aleatoria.

- **Establecer la configuración:** El usuario podrá realizar multitud de cambios en la configuración del algoritmo para adaptar el funcionamiento del algoritmo a sus necesidades o incluso ejecutar los módulos opcionales disponibles.

- **Ejecución:** En el momento de la ejecución del algoritmo, es necesario que el usuario elija y facilite la base de datos donde se aplicará el algoritmo y una semilla para controlar la aleatoriedad en la ejecución del mismo. Por defecto, el algoritmo será ejecutado unas 100 veces, cambiando la semilla en cada ejecución. Para hacer esto de forma controlada, la semilla introducida por el usuario se utilizará en la creación de un vector aleatorio de semillas cuya longitud sea igual al número de ejecuciones, y en cada ejecución se utilizara un elemento de este vector como semilla.

- **Mostrar y almacenar resultados:** La aplicación proporcionará la opción al usuario de mostrar los resultados en pantalla y almacenarlos en un fichero si se considera necesario.

2.2.2. Entorno

El entorno de programación que se utilizará es *PyCharm* (PyCharm community 2016.2.3) [5], que está preparado para el lenguaje de programación interpretado *Python* y cuenta con un depurador integrado entre otras herramientas útiles. Para el proyecto, se utilizará la distribución de Linux Ubuntu 14.04.

Los usuarios a los que van dirigidas las aplicaciones necesitarán conocimientos avanzados sobre algoritmos genéticos, bases de datos desbalanceadas y técnicas de remuestreo, para que así puedan introducir correctamente todos los parámetros y configuraciones requeridos en el

algoritmo principal o en algunos de sus módulos, además de para poder interpretar los resultados de los experimentos.

2.2.3. Vida esperada

La vida esperada en los casos de proyectos de investigación es difícil de predecir. Cada año aparecen gran cantidad de algoritmos nuevos o mejora de algoritmos antiguos (y no tan antiguos). Esto implica que aparezcan algoritmos que mejoren tanto en eficacia como en eficiencia los resultados obtenidos en este Trabajo. Al estar nuestro software basado en un campo de investigación en continuo desarrollo y dada la naturaleza cambiante del mundo tecnológico que nos rodea, nos hace estimar la vida esperada como corta.

Dejando a un lado la parte más teórica del Trabajo, no es extraño ver como este tipo de algoritmos son utilizados como base en otros que consiguen alcanzar los mismos o mejores resultados con menor coste computacional, o por otro lado, que se produzcan mejoras constantes en las librerías utilizadas en nuestra aplicación y otras disponibles.

En conclusión, el presente Trabajo no debe ser visto nunca como un trabajo totalmente terminado, sino como una base para el desarrollo futuro de un sistema o sistemas más completos.

2.2.4. Ciclo de mantenimiento

Debido a que la vida esperada de la aplicación que se pretende desarrollar en el presente proyecto es limitada, posiblemente sea necesario llevar a cabo en un futuro cercano una serie de posibles mejoras y/o ampliaciones en futuros Trabajos Fin de Grado, implementación de métodos que se adapten y ayuden al funcionamiento de este algoritmo o la implementación de técnicas diferentes que se puedan adaptar a conjuntos con características especiales.

La ausencia de una planificación de mantenimiento se debe a las distintas circunstancias que rodean nuestro proyecto.

2.2.5. Competencia

El problema derivado de las base de datos desbalanceadas ha sido muy estudiado en la historia reciente, y aun así, no existen muchas soluciones a este problema tan recurrente en muchos entornos de investigación o recolección de datos.

En lo que a algoritmos de over-sampling se refiere, el más básico y utilizado es el SMOTE [2]. También existen otros basados en este, pero ninguno de ellos incorpora una metaheurística para optimizar la generación de patrones sintéticos. Además en el Trabajo que aquí se presenta, se han añadido una serie de módulos que, con la configuración adecuada, pueden ayudar a alcanzar mejores resultados, y que tampoco están presentes en los algoritmos existentes (véase capítulo el capítulo de Capítulo 3: Antecedentes).

Aun así, el fin de este Proyecto es la investigación y ampliación del campo en cuestión, y otorgar una herramienta adicional a todos aquellos investigadores que lo necesiten y no encuentren saciadas sus necesidades en la utilización de otros algoritmos. Por tanto, se puede decir que, no se busca tener y no se considera que tiene una competencia real en ese sentido.

2.2.6. Aspecto externo

La aplicación se pretende que sea presentada al usuario en un CD-ROM, debido a que tiene buena capacidad de almacenamiento, rapidez, durabilidad, resistencia, gran utilización en muchos de los equipos actuales, coste mínimo y fácil portabilidad.

Junto al CD, se entregará al usuario un manual técnico encuadernado y redactado de forma clara para hacer sencillo el uso de la aplicación, mediante ilustraciones y gráficos explicativos, de forma que su lectura sea agradable para el usuario final de la aplicación.

Tanto el manual de usuario como el manual técnico serán incluidos en el CD-ROM en formato PDF, formato muy conocido y utilizado para el intercambio de documentos. Se incluirá así mismo el manual de código dirigido a aquellos usuarios que deseen hacer un uso del código tanto para resolver algunas dudas, como para mejorarlo o realizar alguna modificación en él.

2.2.7. Estandarización

En relación a la estandarización en la programación, se ha seguido la guía de estilo PEP8 de *Python* [6].

En cuanto al código en sí, se seguirán las convenciones recomendadas de nombramiento tanto general como especifico [7]. Algunas de ellas son:

- Todos los nombres deben ser lo más auto explicativos posible.

- Los nombres de las variables y de las funciones deben estar en minúscula, aun cuando el nombre es la unión de varias palabras. En ese caso la primera palabra debe empezar por minúscula y cada una de las posteriores precedidas por un guion bajo, siendo el resto de letras minúsculas.

- Las abreviaciones y acrónimos no deben ser escritos en mayúscula cuando sean utilizados como nombres.

Como este Trabajo se ha inspirado en algoritmos anteriores, se han respetado algunos nombres de variables que eran recurrentes en la mayoría de ellos, por tanto se han considerado como excepciones y no han sido adaptados a las normas descritas arriba.

2.2.8. Calidad y fiabilidad

La calidad de este TFG será evaluada directamente por el alumno y sus directores. A la terminación del mismo, será calificado por el Tribunal que evalúe el Trabajo. En la medida de lo posible se intentará seguir los estándares de Ingeniería de Software para favorecer la mayoría de los factores que afectan a la calidad de una aplicación.

En relación a la fiabilidad, la aplicación se diseñará para que, en caso de que el usuario introduzca parámetros incorrectos, esta le informe con un mensaje de error. Asimismo, en caso de que el usuario no introduzca algunos parámetros, la aplicación estará preparada para usar unos valores por defecto que, aun no siendo perfectos, funcionan bien en la mayoría de los casos. Algunos parámetros necesitan una configuración obligatoria por parte del usuario. Si al ejecutar el algoritmo, el usuario lo hace de forma incorrecta, se le informará de cuál ha sido el error y se mostrará un ejemplo de cómo debería hacerlo de forma correcta.

2.2.9. Programa de tareas

De manera general, el desarrollo del Trabajo se divide en las siguientes etapas:

1. **Fase de preparación y obtención de conocimientos:** En esta primera fase se estudiará el problema desde un punto de vista teórico, así como los conceptos generales y básicos que forman el contorno del problema. También se estudiarán las distintas técnicas usadas en los algoritmos existentes de remuestreo por over-samplig, además de estudiar cómo puede ser implementada una metaheurística durante este proceso. Adicionalmente, también se estudiará el lenguaje de programación *Python*, algunas librerías como *Scikit-learn* [8], *NumPy* [9], *Pandas* [10] o *Matplotlib* [11] y la herramienta *PyCharm* [5] para aprovechar sus ventajas durante la implementación del código.

2. **Fase de diseño:** Una vez analizados y estudiados los requisitos y objetivos del sistema, se procederá a diseñarlo para su posterior codificación. Se analizará la funcionalidad del sistema, el diseño de los datos, el flujo de control que estos seguirán, etc.

3. **Fase de implementación:** Esta fase se corresponde con la implementación del algoritmo principal y todos sus módulos de forma óptima, utilizando para ello el lenguaje interpretado *Python*.

4. **Fase de pruebas:** Esta fase será fundamental para refinar nuestro código, dado que servirá para determinar los errores del mismo y poder así corregirlos y mejorar la aplicación. Se realizarán diversas pruebas, las cuales serán evaluadas para proceder en función de los resultados de las mismas.

5. **Fase de documentación:** Se corresponde con la realización de la memoria final del Trabajo Fin de Grado.

6. **Fase de aplicación:** En esta fase aplicaremos nuestro software a una serie de bases de datos de clasificación disponibles en UCI[1].

2.2.10. Pruebas

En la fase de pruebas, se pretende aportar una visión de las estrategias seguidas en la certificación del software y de los resultados obtenidos en las mismas. La estrategia de pruebas a seguir abarca:

- Pruebas de unidad a nivel de módulos.
- Pruebas de integración para comprobar el ensamblado de los módulos.
- Pruebas de validación, para probar el software como un conjunto.
- Pruebas de sistema, para asegurar el cumplimiento de los objetivos.
- Pruebas de aceptación, para que el usuario pueda verificar si el producto se adapta a los requisitos.
- Además, se probará la validez del algoritmo sobre diferentes bases de datos para así determinar su desempeño y funcionamiento.

Una vez se hayan corregido los posibles errores que puedan surgir en cada una de las pruebas y se hayan cumplido todos los requisitos para los que se diseñó, consideraremos que la aplicación es válida.

Una vez se generen los patrones sintéticos que ayudarán en el aprendizaje, serán sometidos a una serie de pruebas que permitirán certificar su correcto funcionamiento y corregir todos los posibles fallos o errores. Se comprobará si los resultados finales cumplen los objetivos ya planteados, es decir, que se genere un conjunto de patrones sintéticos capaz de beneficiar a la clasificación de los patrones de la clase minoritaria.

2.2.11. Seguridad

La temática del Trabajo realizada no permite la oportunidad de realizar operaciones incorrectas que salgan de sus objetivos determinados. Cabe destacar que todas las modificaciones que se realizan al conjunto de entrada se hacen a nivel local, y nunca afectan al conjunto original, por lo que este nunca se ve comprometido en ningún caso.

[1] UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets.html

La privacidad no es requerida en este Trabajo, pues el tipo de datos manejados no hacen que sea primordial la implementación de ningún sistema que los proteja y oculte.

Capítulo 3:
Antecedentes

A lo largo del tiempo se han propuesto una gran cantidad de algoritmos para abordar problemas de clasificación en base de datos desbalanceadas. En este capítulo, se pretende describir las metodologías que más relación presentan con respecto a la idea expuesta en el presente documento.

Estas metodologías se engloban bajo el nombre de técnicas de remuestreo, y están claramente divididas en dos grandes grupos, *under-sampling* y *over-sampling*. Será esta última la que comentaremos en mayor profundidad, ya que la propuesta de este proyecto se identifica como una técnica de *over-sampling*

3.1. Técnicas de Under-Sampling

En este grupo se engloban aquellas técnicas que pretenden conseguir una distribución de las clases más balanceada a través de la eliminación patrones de la clase mayoritaria. La más extendida es el *Random Under-Sampling* (RUS) [1] [12] [13], que se trata de un procedimiento no heurístico cuyo funcionamiento consiste en eliminar, de forma totalmente aleatoria, patrones de la clase mayoritaria.

Existen otros métodos que pretenden mejorar la selección de los patrones a eliminar, como "Tomek Links" [14] o "The Condensed Nearest Neighbour Rule" (CNN) [15], pero no pasaremos a analizarlos en profundidad ya que en este proyecto no se aplicará ninguna de estas técnicas, debido a que varios investigadores han destacado el peligro que supone la eliminación de información en bases de datos pequeñas, como las que se han considerado en este proyecto [16].

3.2. Técnica de Over-Sampling

Al contrario de lo que ocurre en *under-sampling*, las técnicas de *over-sampling* se encargan de generar una serie de patrones sintéticos pertenecientes a la clase minoritaria, y de ese modo se consigue balancear la distribución entre las clases.

Algunos de los métodos más utilizados, y sobre los que se ha inspirado este documento serán comentados a continuación.

3.2.1. Random Over Sampling (ROS)

Random Over Sampling es el algoritmo más básico para realizar un sobremuestreo de la clase minoritaria. En él, simplemente se eligen patrones al azar y se replican hasta conseguir una

distribución de las clases equilibrada [1] [12] [13]. Este método introduce una gran duplicidad de los patrones de la clase minoritaria en el conjunto, con todas sus consecuencias. Al existir muchas instancias repetidas, estas serán aprendidas fácilmente por el clasificador, pero de una forma tan estricta y rígida que provocará sin duda un sobreentrenamiento y la incapacidad por parte del clasificador de generalizar y acertar en la clasificación del conjunto de test y nuevas instancias.

Este algoritmo podría funcionar ligeramente bien replicando instancias consideradas como raras, ya que estas podían dificultar gravemente al clasificador como se explicará posteriormente en el apartado 7.1 (Aprendizaje sobre conjunto de datos desbalanceados). Pero aun así, la existencia de datos duplicados en un conjunto de datos podía afectar negativamente a un algoritmo de aprendizaje automático, por lo que de manera general no se recomienda el uso de este algoritmo.

3.2.2. Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE [2] es el algoritmo más usado de este grupo y sobre el que se basan todos los algoritmos más actuales.

Como ya se ha comentado, es un algoritmo de sobremuestreo de la clase minoritaria. En este algoritmo no se replican las instancias como tal, sino que se generan patrones sintéticos para que el conjunto quede parcialmente balanceado, o incluso si se desea, totalmente balanceado.

Estos patrones son creados seleccionando cada instancia de la clase minoritaria y generando un nuevo patrón sintético en algún punto de la recta que une al patrón original con todos o algunos de sus *k* vecinos más cercanos (también de la clase minoritaria). Dependiendo del grado de desbalanceo que sufra el conjunto se utilizarán más o menos vecinos dentro de los *k* más cercanos. Esta técnica se conoce como interpolación y sigue la siguiente fórmula:

$$S_i = x_i + (x_{ji} - x_i) \cdot \lambda$$

donde S_i es el patrón sintético resultante, x_i y x_{ji} son el conjunto de n características que definen a dos patrones del conjunto original pertenecientes a la clase minoritaria, y λ es un número aleatorio entre 0 y 1. De esta forma, los patrones sintéticos creados poseerán características basadas en patrones reales y potencialmente representaran a un posible individuo del mundo real.

La elección del vecino con el que se genera el sintético es completamente aleatoria. El punto de la recta que une a los dos vecinos donde será generado el sintético, también es elegido aleatoriamente (determinado por λ).

3.2.3. Adaptive Synthetic Sampling (ADASYN)

ADASYN [3] sigue el mismo razonamiento que SMOTE, pero con algunas diferencias. Las más importantes serán explicadas a continuación.

La idea es generar los patrones sintéticos lo más cerca posible del borde que separa la clase minoritaria de la mayoritaria. Para identificar el mencionado borde, se buscan los patrones de la clase minoritaria que tengan vecinos de la otra clase. Posteriormente, se asigna una importancia a cada patrón de la clase minoritaria en función de cuantos vecinos haya de la clase mayoritaria dentro de sus *k* vecinos más cercanos, es decir, cuanto mayor sea la proporción de vecinos de la clase mayoritaria dentro de los *k* vecinos, mayor importancia se le asignará al patrón.

Una vez asignada una importancia a cada patrón, esta es utilizada en el proceso de generación de sintéticos de modo que los patrones con mayor importancia generan más sintéticos. De esta forma, se busca potenciar el borde, que es la zona más conflictiva para la clasificación.

En la generación de los sintéticos, los vecinos con los que se genera dicho sintético también se eligen de forma aleatoria y se utiliza la misma interpolación que en SMOTE, solo que en ADASYN el número de sintéticos que se generan de cada patrón es desigual y depende de la importancia asignada en el proceso anterior.

Las principales ventajas de este algoritmo es que utiliza un procedimiento sencillo para la detección del borde que separa ambas clases y que los patrones con mayor importantica (más cercanos al borde) generan un mayor número de sintéticos. Al mismo tiempo, el vecino con el que se genera un sintético sigue una selección puramente aleatoria, hecho que le hace perder un gran potencial en su intención de reforzar el borde que separa las clases.

3.2.4. Borderline-SMOTE

Borderline-SMOTE [17] sigue el mismo razonamiento que SMOTE y al igual que ADASYN busca los patrones cercanos al borde, ya que son los que tienen más posibilidades de ser clasificados de forma errónea, y por tanto, se pretende reforzar esa zona. Además, consta de un proceso de eliminación de ruido para que el borde quede mejor definido y facilite el proceso de clasificación.

El procedimiento empieza calculando los *k* vecinos más cercanos de cada patrón de la clase minoritaria y según la proporción de vecinos de la clase mayoritaria se actúa de la siguiente forma:

- Si todos los vecinos son de la clase mayoritaria, se considera ruido y se elimina el patrón.

- Solo aquellos patrones que tengan más de la mitad de los vecinos pertenecientes a la clase mayoritaria pasaran a formar parte del "grupo de peligro".

Por último se decide *s*, que representa cuantos sintéticos se generaran por cada patrón en el "grupo de peligro". Así pues, por cada patrón del "grupo de peligro" se eligen aleatoriamente *s* vecinos de todos los existentes de la clase minoritaria y se genera un sintético por cada uno utilizando la misma interpolación que utiliza SMOTE.

La única diferencia significativa con SMOTE, aparte de la eliminación de ruido, es que en este algoritmo solo se generan sintéticos a partir de patrones cercanos al borde (grupo de peligro).

3.2.5. Majority Weighted SMOTE (MWMOTE)

MWMOTE [4] añade un proceso de clustering que no es incluido en los demás procesos aquí mencionados. Al igual que algunos de los métodos anteriores genera un subgrupo solo con los patrones pertenecientes a la clase minoritaria y que además están cercanos al borde. Una vez generado este subgrupo utiliza un procedimiento de clustering sobre el subgrupo mencionado, con lo que crea clústeres restringidos localizados en pequeñas zonas de densidad.

Cuando llega el momento de generar los patrones sintéticos, solo son generados con vecinos pertenecientes al mismo clúster, por tanto nunca se genera un sintético dentro de la zona de la clase mayoritaria.

Al no generar sintéticos fuera de los clúster se soluciona el problema general de estos algoritmos que consiste en generar sintéticos en zonas conflictivas, pero también se pierden zonas potencialmente beneficiosas y "pertenecientes" a la clase minoritaria donde no será generado ningún sintético y por tanto no serán reforzadas.

MWMOTE incluye también un proceso de eliminación de ruido muy similar al utilizado en Borderline-SMOTE, ya que cuando un patrón de la clase minoritaria tiene todos sus vecinos pertenecientes a la clase mayoritaria, es considerado ruido y se elimina.

3.2.6. Safe-Level-SMOTE

Al contrario que algunos de los algoritmos anteriormente descritos, "Safe-Level-SMOTE" [18] potencia las zonas seguras, más alejadas del borde. Esto de algún modo sacrifica los patrones cercanos al borde ya que seguramente serán confundidos durante la clasificación, pero por otro lado refuerza fuertemente los patrones que presentan más diferencias con los de la clase mayoritaria. Al igual que Borderline-SMOTE, también elimina ruido utilizando el mismo proceso.

Por cada patrón de la clase minoritaria se escoge un vecino (también minoritario) y se genera un sintético cercano al patrón más seguro, es decir aquel que tenga menos vecinos pertenecientes a la clase mayoritaria. Si ambos patrones tienen el mismo nivel de seguridad, el patrón será generado utilizando la misma interpolación que SMOTE.

Si una vez que se ha generado un sintético por cada patrón es necesario generar más sintéticos, podría repetirse el procedimiento hasta que se genere el número de sintéticos deseado.

3.3. Metaheurísticas

La primera aparición del termino metaheurística fue en 1986 [19], donde se combina el prefijo griego "meta" ("más allá", aquí con el significado de "nivel superior") y "heurístico" (del griego ευρισκειν, heuriskein, "encontrar").

Una heurística es un “procedimiento simple, a menudo basado en el sentido común, que se supone que ofrecerá una buena solución (aunque no necesariamente la óptima) a problemas difíciles, de un modo fácil y rápido”. (Zanakins y Evans, 1981 [20]).

Como su propio nombre indica, los algoritmos metaheurísticos intentan ir más allá de los algoritmos heurísticos clásicos y se definen como procesos heurísticos de alto nivel en los que se combinan diferentes técnicas heurísticas para conseguir una exploración más completa de la región de búsqueda. Osman y Kelly (1996) nos aportan la siguiente definición “Los procedimientos metaheurísticos son una clase de métodos aproximados que están diseñados para resolver problemas difíciles de optimización combinatoria, en los que las heurísticas clásicas fracasaron en ser efectivas y eficientes. Las metaheurísticas proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de las heurísticas clásicas, la inteligencia artificial, la evolución biológica, sistemas neuronales y la mecánica estadística” [21].

Las metaheurísticas se utilizan, por ejemplo, cuando no existe ningún método exacto de resolución, cuando el método de resolución conocido consume mucho tiempo para ofrecer la solución óptima, cuando existen restricciones de tiempo o como paso intermedio para obtener una solución inicial a la que se le aplicará otra técnica o metaheurística diferente.

3.3.1. Iterated Greedy.

Iterated Greedy o metaheurística voraz iterativa [8] [22], es un procedimiento donde se calcula una solución inicial de forma voraz, y a través de un proceso de optimización se evoluciona esta hasta dar con otra que satisfaga mejor las necesidades del problema a solucionar.

La idea principal de *Iterated Greedy* se basa en utilizar un constructor para formar una solución inicial mediante un procedimiento voraz, y posteriormente destruir parcialmente dicha solución. Los elementos supervivientes a la destrucción guían al constructor en un proceso de reconstrucción de la solución, de forma que la nueva solución reconstruida sea diferente a la que se destruyó en el proceso anterior. Repitiendo varias veces los procesos de destrucción y reconstrucción es posible optimizar la solución original y explorar el espectro de soluciones del problema.

Los elementos que conforman *Iterated Greedy* se analizarán a continuación:

1. **Constructor voraz**: En la definición más simple de *Iterated Greedy*, el constructor es puramente voraz, pero este puede ser optimizado añadiendo aleatoriedad mediante heurísticas u otros procedimientos de optimización. Otras incorporaciones interesantes podrían ser la realización de una búsqueda local durante la construcción o la utilización de una lista tabú para evitar visitar soluciones ya exploradas anteriormente.

2. **Destructor:** Originalmente el proceso de destrucción se define como un proceso totalmente aleatorio, pero este también podría ser visto como un procedimiento a optimizar y utilizar varias técnicas para seleccionar los elementos a destruir.

3. **Criterio de aceptación:** Para conseguir la optimización de la solución durante el proceso, es necesario seguir un criterio de aceptación que facilite la convergencia. El más simple sería aceptar una solución cuando esta mejora a la mejor solución encontrada hasta el momento, pero si esto provoca sobre-entrenamiento es posible añadir otras técnicas conocidas, como por ejemplo, enfriamiento simulado.

Es importante recordar que en la primera iteración del algoritmo el constructor parte de una solución vacía, y por tanto debe construir una solución completa.

Capítulo 4:
Objetivos

En este capítulo, se analizará los objetivos del Trabajo Fin de Grado. En líneas generales, se pretende diseñar e implementar un algoritmo de remuestreo de patrones donde el proceso de generación será optimizado mediante la metaheurística voraz iterativa.

En concreto, se han dividido en objetivos de formación y objetivos operacionales y se detallan a continuación.

4.1. Objetivos de formación

Los objetivos de formación de este trabajo son los siguientes:

1. Estudio en profundidad del lenguaje de programación *Python*.

2. Estudio en profundidad sobre el manejo y la utilización de algunas librerías existentes en *Python* como *Scikit-Learn*, *NumPy*, *Pandas*, *Scipy* o *Matplotlib*.

3. Realizar un estudio teórico sobre las temáticas del proyecto, incluyendo aprendizaje automático o técnicas de remuestreo.

4. Estudio del algoritmo ADASYN como base de inspiración para la creación de nuestro sistema.

5. Estudio de un método para la selección de los mejores vecinos con los que generar sintéticos en lugar de utilizar una selección aleatoria, con el fin de generar unos patrones que puedan otorgar una información más valiosa durante el aprendizaje y al mismo tiempo evitar, de una forma más eficaz, la creación de patrones sintéticos en zonas poco adecuadas.

6. Profundizar en el aprendizaje de todo el proceso relacionado con la elaboración de una aplicación y las herramientas de apoyo necesarias, incluyendo la creación del manual de usuario, manual técnico, etc.

7. Estudio final de los resultados obtenidos de diferentes algoritmos, como son:
   - **Adaptive Synthetic Sampling** (ADASYN)
   - **Adaptive neighbors Synthetic Sampling** (ANESYN). Algoritmo que solo utiliza el algoritmo de remuestreo explicado en el apartado 10.1 sin ser optimizado a través de ninguna metaheurística
   - **Adaptive neighbors Iterated Greedy Synthetic Sampling** (ANEIGSYN). Algoritmo completo que aplica tanto el algoritmo de remuestreo (apartado 10.1. Algoritmo de

over-sampling), como la metaheurística voraz iterativa (apartado 10.2. Aplicación de Iterated Greedy).

4.2. Objetivos operacionales

Los objetivos que deberá cumplir la aplicación que se pretende desarrollar son los siguientes.

- Proporcionar un método de over-sampling capaz de generar un conjunto de sintéticos para equilibrar el conjunto de datos original y beneficiar la clasificación obtenida en el conjunto de *Test* y nuevas instancias.

- Definición del proceso de remuestreo como un problema de optimización, donde se busca que los patrones generados beneficien lo máximo posible al proceso de clasificación.

- Conseguir optimizar el proceso de creación de patrones sintéticos a través de una metaheurística voraz iterativa.

- Dar la posibilidad al usuario de poder configurar tanto el algoritmo principal como algunos de los módulos implementados, para así poder adaptar el funcionamiento del mismo a las necesidades que dicho usuario pueda tener.

- Ofrecer diferentes formas de tratar los datos obtenidos:
  - Opción de mostrar por pantalla
  - Opción de almacenar en un archivo CSV para su posterior análisis.

Capítulo 5:
Restricciones

A continuación, se expondrán todas aquellas restricciones existentes en el ámbito del diseño que condicionan la elección de diferentes alternativas en las fases posteriores del proyecto. Dividiremos este apartado en dos bien diferenciados que representarán los dos tipos de factores que se tendrán en cuenta.

- **Factores dato:** Son aquellas restricciones que no pueden ser modificadas, inherentes al entorno en el que se desarrolla el Proyecto. Son generalmente indicadas por el cliente, en nuestro caso, por los directores del proyecto.

- **Factores estratégicos**: Representan aquellas opciones elegidas por el proyectista de entre otras posibilidades, sobre la base de un interés determinado, sin que sean impuestas por agentes externos.

5.1. Factores dato.

Los factores dato comprenden un grupo de restricciones que deben ser cumplidas a todos los efectos a lo largo del desarrollo del proyecto, sin tener que estar necesariamente justificadas.

- **Restricciones de personal**: Debido a que nos encontramos desarrollando un TFG, este factor viene condicionado por la normativa de la Universidad de Córdoba que solo permite el desarrollo del mismo por parte de un alumno con la ayuda de su director o directores.

- **Restricciones económicas:** Al tratarse de un TFG y no del desarrollo de una aplicación profesional, el proyectista debe apoyarse el máximo posible en el uso de software libre, y si no queda más remedio, de un software cuya utilización en ámbitos fuera del estrictamente profesional esté permitido sin pago alguno o que el proyectista posea alguna licencia para la utilización de un determinado software por motivos ajemos a la realización de este TFG.

- **Restricciones temporales:** Están condicionadas por el cumplimiento de todos los objetivos para que la aplicación desarrollada en nuestro proyecto se considere válida. Sin embargo, se espera que el tiempo de elaboración no sobrepase marzo de 2017, siempre y cuando no ocurran sucesos imprevistos.

- **Restricciones de hardware:** El alumno proyectista usará sus propios equipos informáticos para la realización del proyecto.

5.2. Factores estratégicos.

Los factores estratégicos son restricciones impuestas por el propio proyectista o por consenso con sus directores del Trabajo Fin de Grado.

Se ha elegido *Python* como lenguaje de programación debido a los siguientes motivos:

- El tiempo de aprendizaje del este lenguaje no es muy elevado, lo que permite preocuparse por el desarrollo del algoritmo en sí mismo y no de aspectos externos de programación.

- Es un lenguaje en auge con usos múltiples en diversos campos, más allá de la ingeniería informática.

- Suficiente flexibilidad a la hora de implementación de algoritmos de aprendizaje automático.

- Existencia de una gran variedad de librerías con multitud de procesos que facilitan tanto la lectura de los datos como la implementación de algunos procedimientos necesarios en la ejecución de todo el algoritmo

- Existencia de una gran documentación y comunidad.

En cuanto a las librerías de *Python* que se utilizarán, se encuentran las siguientes:

- **NumPy:** Es una librería con una gran capacidad para realizar operaciones vectoriales, y por tanto otorga a *Python* un gran potencial al momento de operar con estructuras de datos como matrices [9].

- **Pandas:** Es una librería open source (BSD) para la manipulación y análisis de grandes estructuras de datos. Nos será de gran ayuda a la hora de cargar las bases de datos al sistema y separar entradas y salidas [10].

- **Scikit-learn:** Es una librería orientada especialmente a la creación de software de aprendizaje automático. Tiene una inmensa variedad de herramientas que nos pueden ser de ayuda en la implementación de la aplicación. En principio podríamos destacar la capacidad de poder dividir la base de datos en subconjuntos estratificados, disponer de varios clasificadores para utilizarlos en el proceso final del algoritmo y muchos evaluadores para poder medir la precisión de la clasificación realizada [8].

- **Scipy:** Es una librería capaz de realizar operaciones científicas y matemáticas con gran eficiencia. Se pretende utilizar en el cálculo de distancias para hallar la vecindad de los patrones del conjunto de datos, un cálculo que a priori parece muy costoso.

- **Matplotlib:** Es una librería de para pintar gráficas en 2D, y algunas opciones para generar gráficas en 3D. Fundamentalmente, se utilizará para controlar la evolución de la metaheurística gráficamente y ver que no se produce sobre-entrenamiento.

Se ha utilizado Microsoft Word 2013 para la generación del presente documento, gracias a su capacidad de edición de documentos y al conocimiento previo del proyectista en el uso de esta herramienta. Además se intentado seguir en la medida de lo posible la estructura de documento descrita en el estándar IEEE 830-1998 [23]. En este documento se detalla que aun no siendo obligatorio, si es recomendable de una forma o de otra, incluir toda la información de la que se habla en el mismo. Por ello, se han tenido en cuenta estas recomendaciones durante el desarrollo de todo el documento.

Las bases de datos utilizadas deben estar en CSV (*Comma separated values*), debido a que es el formato más utilizado en algunos ámbitos y por tanto hay una amplia disponibilidad de bases de datos en este formato, además de ser un tipo de documento con formato abierto. Adicionalmente presenta una codificación muy sencilla que facilita su modificación en caso de ser necesario.

Como entorno de programación se ha elegido *Pycharm* [5] debido a sus funcionalidades adicionales, entre las que se encuentra un depurador gráfico con el que poder identificar y solucionar los posibles errores que se detecten durante el desarrollo de la aplicación.

Se ha elegido como base de inspiración el algoritmo ADASYN por los siguientes motivos:

- Busca reforzar el borde de separación de clases, que es la zona más difícil de aprender para un clasificador.

- Posee una idea única y con un gran potencial donde se otorga más importancia a los patrones cercanos al borde para que estos generen más sintéticos que los patrones más alejados.

- La detección del borde se realiza mediante un procedimiento sencillo de cálculo de vecindad, y no necesita la utilización de técnicas más costosas como en MWMOTE.

Algunas de las debilidades de ADASYN que se pretenden solventar en el diseño de nuestro sistema con las siguientes:

- Elección del vecino con el que se genera un patrón sintético de una forma más eficaz para realmente potenciar el borde que separa ambas clases.

- Al mismo tiempo que se pretende potenciar el borde, se busca evitar que los patrones sintéticos sean generados en zonas poco adecuadas (superposición de clases), ya que si esto ocurriera, podría provocar el efecto contrario al que buscamos y perjudicar al clasificador durante el entrenamiento.

Capítulo 6:
Recursos

En este apartado serán expuestos los distintos recursos humanos, de hardware y de software, con los que se cuentan para la realización del presente Trabajo Fin de Grado.

6.1. Recursos humanos

Los participantes en el trabajo se especifican en las siguientes tablas.

Participante	Francisco Javier Maestre Garcia
Organización	EPSC UCO
Rol	Desarrollador
Es desarrollador	Sí
Es Cliente	No
Es usuario	No
Comentarios	Alumno de Grado de ingeniería Informática de la Escuela Politécnica Superior de Córdoba de la universidad de Córdoba, encargado de desarrollar la aplicación.

Tabla 1: Participante Francisco Javier Maestre García

Participante	Pedro Antonio Gutiérrez Peña
Organización	AYRNA[2]
Rol	Tutor
Es desarrollador	No
Es Cliente	Sí
Es usuario	Sí
Comentarios	Profesor Titular de Universidad del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo AYRNA en la Escuela Politécnica Superior de Córdoba de la universidad de Córdoba. Estará al cargo de dar soporte para consultas de diseño e implementación de la aplicación.

Tabla 2: Participante Pedro Antonio Gutiérrez Peña

[2] Grupo de investigación TIC 148 "Aprendizaje y Redes Neuronales Artificiales" (AYRNA) de la Universidad de Córdoba.

Participante	Carlos García Martínez
Organización	KDIS[3]
Rol	Tutor
Es desarrollador	No
Es Cliente	Sí
Es usuario	Sí
Comentarios	Profesor Titular de Universidad del Dpto. de Informática y Análisis Numérico y miembro investigador del grupo KDIS en la Escuela Politécnica Superior de Córdoba de la universidad de Córdoba. Estará al cargo de dar soporte para consultas de diseño e implementación de la aplicación.

Tabla 3: Participante Carlos García Martínez

6.2. Recursos materiales

6.2.1. Recursos hardware

Equipo personal del proyectista, consistente en un PC Portátil con las siguientes características:

- **Procesador**: Intel Core i7 – 2630QM, 2.00GHz
- **Disco duro**: 750GB
- **Memoria RAM**: 6 GB DDR3
- **Memoria caché**: 6Mb

6.2.2. Recursos software

- Ubuntu 14.04 como sistema operativo.
- *Python* como lenguaje de programación.
- *Scikit-Learn, NumPy, Pandas, Scipy* o *Matplotlib* como librerías o bibliotecas de recursos para la implementación del sistema y sus módulos.
- *Atom y Pycharm* como editor de textos y entornos de programación.
- Word 2013 para edición de documentos.

[3] Grupo de investigación TIC 222 "Knowledge discovery and intelligent systems" (KDIS) de la Universidad de Córdoba.

Parte II
Especificación de requisitos

Capítulo 7:
Descripción general del problema

En este capítulo, describiremos algunas de las principales dificultades y condicionantes que pueden influir de alguna forma en el dominio del problema planteado en este documento de manera general, y en las diferentes metodologías utilizadas en la implementación del sistema de manera particular. Se otorgará un visión interesante sobre varios aspectos que cualquier investigador que pretenda crear o analizar un sistema similar debería tener en consideración, de una manera u otra.

7.1. Aprendizaje sobre conjunto de datos desbalanceados

En este apartado hablaremos sobre las características de los conjuntos desbalanceados y cómo afectan estos a un proceso de aprendizaje. A lo largo y ancho del mundo existen muchos problemas de clasificación, pero centraremos nuestro interés en los problemas binarios con solo dos clases ya que nuestro sistema irá orientado a solucionar este tipo de problemas.

Técnicamente, cualquier conjunto de datos que presente una distribución desigual para las diferentes clases que lo forman se puede considerar desbalanceado. Sin embargo, el concepto de desbalanceo que se suele ver en la comunidad alrededor de este tipo de problemas se refiere a problemas significativamente desbalanceados o extremadamente desbalanceados, con ratios de desbalanceo de entre 100:1 y 10000:1 o incluso superior, donde una clase domina con mucha diferencia la distribución en comparación con la otra clase. Nuestra experiencia nos dice que un conjunto de datos podría considerarse lo suficientemente balanceado si se encuentra en el límite 60% - 40%, es decir, que la clase mayoritaria no supere el 60% de la distribución, de otra forma podría considerar como un conjunto desbalanceado.

Los desequilibrios dentro de un conjunto de datos se deben normalmente a motivos intrínsecos al propio problema, dicho de otra forma, se dan por motivos propios a la naturaleza del mismo, como por ejemplo que el número de personas sanas es muy superior al número de personas enfermas de una determinada enfermedad. No obstante, las distribuciones desequilibradas no se limitan a motivos intrínsecos. Factores como el tiempo o el almacenamiento también podrían dar lugar a conjuntos de datos desbalanceados. Este tipo de desequilibrio se denomina como extrínseco, y no tiene relación alguna con la naturaleza del problema. Por ejemplo, supongamos que un conjunto de datos se obtiene mediante un flujo continuo de información proveniente de datos balanceados a lo largo de un intervalo de tiempo específico, si durante este intervalo la transmisión tiene interrupciones esporádicas donde los datos no se transmiten o si se dieran fallos en la recepción, es posible que el conjunto de datos adquiridos pueda quedar desequilibrado, en cuyo caso el conjunto de datos sería un conjunto de datos desequilibrados debido a un factor extrínseco, obtenido de un espacio de datos equilibrado. Todos los conjuntos desequilibrados son de interés para la comunidad, ya sean debidos a factores intrínsecos o extrínsecos.

Las diferentes soluciones para los conjuntos de datos desbalanceados se dividen en dos categorías, soluciones a nivel de datos y a nivel de clasificador. Las soluciones a nivel de datos intentan balancear la distribución del conjunto para que esto no suponga una complicación para cualquier clasificador. Las soluciones a nivel de clasificador intenta diseñar nuevos clasificadores que no se vean tan afectados por la diferencia de la distribución entre las clases. En el documento de "Learning from Imbalanced Data" [1], se habla de que el desequilibrio de los conjuntos de datos no es el único factor que puede afectar en gran medida a un clasificador y separa los conjuntos de datos desequilibrados en dos tipos, desequilibrio relativo y desequilibrio debido a instancias raras. En el documento se habla de que, aun cuando un conjunto está desbalanceado, si su distribución es sencilla un clasificador podría conseguir buenos resultados en la clasificación, pero que si por el contrario existen instancias raras en el conjunto, estas podrían afectar negativamente al clasificador, más incluso que el hecho de que el conjunto esté desbalanceado. En cualquier caso, nosotros nos centraremos principalmente en solucionar los problemas relacionados con el desequilibrio en la distribución.

En general, el principal problema de los clasificadores es que no están preparados para asumir una distribución desigual de los datos ni para adaptarse a ella, ya que entre otras cosas utilizan un coste de clasificación errónea simétrico (el mismo para ambas clases). La clase mayoritaria posee un mayor número de individuos y estos adquieren más representación debido a la gran diversidad de los datos que existe en ella. Esta heterogeneidad dentro de la clase mayoritaria no solo hace que existan más casos diferentes, y por tanto un aprendizaje más amplio, sino que además, muchas de las características que estos poseen se ven reforzadas como consecuencia de las similitudes que se dan entre algunos de ellos. Este hecho no se da con tanta presencia en la clase minoritaria, y adicionalmente al estar superada significativamente por la presencia de la clase mayoritaria, el clasificador prioriza el aprendizaje de la misma en detrimento de la clase minoritaria.

Como ejemplo hablaremos de un árbol de decisión. Los arboles de decisión utilizan un algoritmo de búsqueda recursivo codicioso desde arriba hacia abajo para seleccionar la mejor característica, basándose, en la ganancia de información. Este proceso se repite de forma continuada dando lugar a la selección de varias características. Como resultado, el conjunto de entrenamiento se divide sucesivamente en subconjuntos cada vez más pequeños conforme bajamos en el árbol hasta acabar en un nodo hoja que simbolizaría a una clase.

El problema de utilizar este procedimiento sobre base de datos desbalanceadas es que a medida que los subconjuntos se hacen más pequeños, aparecen cada vez menos comprobaciones sobre las características propias de los individuos de la clase minoritaria, ya que estos son cada vez más escasos y su características aportan muy poca información, llegando a clasificarse mal, o incluso a no clasificarse debido a la poca representatividad que poseen.

Algunos estudios han demostrado que la mayoría de los clasificadores base proporcionan un mejor rendimiento sobre conjuntos de datos equilibrados, en consecuencia, la mayoría de las soluciones que existen en la actualidad ponen su foco en paliar los efectos que estos conjuntos tienen sobre los clasificadores estándar [24] [25].

Los resultados de estos estudios justifican el uso de métodos de remuestreo para conjunto de datos desbalanceados. Sin embargo no significa que un clasificador no pueda aprender de un conjunto de datos desequilibrado. Análogamente existen algunos experimentos que demuestran que el desequilibrio atribuible a un conjunto de datos no obstaculiza totalmente el rendimiento de un clasificador, y es comparable de un modo u otro a los resultados que se puedan conseguir con el mismo conjunto equilibrado mediante alguna técnica [12].

Este fenómeno se relaciona directamente con el problema de las instancias raras comentado anteriormente. No obstante, si está demostrado que para la mayoría de los conjuntos desequilibrados la aplicación de técnicas de remuestreo ayudan positivamente a la precisión del clasificador, por lo que se ha optado por aplicar una solución nivel de datos, como es un algoritmo de over-sampling. En los siguientes apartados se comentarán algunas técnicas utilizadas.

7.2. Técnicas de remuestreo para conjuntos desbalanceados

Las técnicas de remuestreo sobre los conjuntos de datos desbalanceados consisten en realizar una modificación sobre dicho conjunto de manera que al final de este procedimiento se encuentre en un estado relativamente balanceado, y presente una distribución de datos equilibrada.

Las principales técnicas de remuestreo se enumeran en los siguientes apartados.

7.2.1. Técnicas de under-sampling

El funcionamiento de las técnicas de under-sampling fue introducido en el apartado (3.1. Técnicas de Under-Sampling). En él se dijo que muchos autores han rechazado la idea de eliminar patrones debido a la perdida potencial de información que podía provocarse al suprimir cualquier cantidad de patrones del conjunto original, más aún cuando el grado de desbalanceo es extremadamente grande. Si la pérdida de información es notoria podríamos estar influyendo negativamente al clasificador y conseguir justo el efecto contrario al deseado. A lo largo del tiempo se han diseñado algunos algoritmos que intentan suavizar o paliar el efecto negativo de este tipo de algoritmos pero sin llegar a eliminarlo del todo ya que la propia naturaleza de este tipo de técnica es eliminar patrones del conjunto original.

Teniendo en cuenta que en multitud de ocasiones el desequilibrio de la distribución no es el único problema que puede afectar a un algoritmo de aprendizaje (como se ha comentado en el apartado 7.1), la posibilidad de crear una nueva dificultad para el clasificador a través de la pérdida de información útil, aun si se consiguiera un conjunto equilibrado, hace que a nivel general no podamos recomendar el uso de las técnicas de under-sampling.

7.2.2. Técnicas de over-sampling

Las técnicas de over-sampling son capaces de equilibrar la distribución de las clases sin eliminar ningún tipo de información, por lo que no se perdería información útil como sí pasa con las técnicas de under-sampling. Como ya se introdujo en el apartado (3.2.1. Random Over Sampling (ROS)), la metodología más básica funciona duplicando los patrones de la clase minoritaria hasta conseguir un conjunto equilibrado. ROS crea una enorme duplicidad en la base de datos, lo que provocaría un gran sobrentrenamiento y un aprendizaje sesgado y centrado en aprender los valores locales que existen en el conjunto original. Sin duda, esta práctica hará que durante la fase de entrenamiento se alcancen valores de precisión muy elevados, pero estos no guardaran ninguna relación con los valores que se alcanzarían en la fase de test, pudiendo llegar incluso a ser nefastos. Tal y como se especifica en el requisito FRQ-02 (Tabla 15), no es conveniente que existan valores repetidos en un conjunto de datos, de manera que podemos afirmar que esta técnica no nos conviene y no es recomendada a nivel general.

Desde la creación de SMOTE, se han diseñado multitud de algoritmos que consiguen generar un conjunto de patrones sintéticos para apoyar a la clase minoritaria sin duplicar los individuos originales, evitando de este modo todas las desventajas que esto suponía. La gran mayoría de ellos utilizan un método de selección de patrones desde los cuales se generan los sintéticos, utilizando la interpolación comentada en el capítulo de antecedentes cuando se introducía el algoritmo SMOTE (3.2.2. Synthetic Minority Over-sampling Technique (SMOTE)).

Utilizando los controles adecuados se puede focalizar la creación de estos patrones en las zonas cercanas al borde que separa ambas clases, de modo que se fortalece una zona conflictiva y de elevada dificultad para el clasificador. Análogamente también se podría intentar evitar que se generen patrones sintéticos en zonas conflictivas dando lugar a una superposición de las clases.

Si esto se consigue hacer de forma satisfactoria podríamos obtener un método capaz de generar un conjunto de patrones que ayuden al clasificador a aprender en mayor medida las peculiaridades de la clase minoritaria consiguiendo una mejor clasificación. Normalmente bastará con equilibrar el conjunto de *Train* y no el conjunto completo dado que este será sobre el que se aplicara el entrenamiento.

En conclusión, las técnicas de remuestreo mediante creación de nuevos patrones presentan más ventajas, o al menos no poseen tantas desventajas como los algoritmos de under-sampling.

7.2.3. Optimización mediante Iterated Greedy

Por todo lo anteriormente expuesto se ha llegado a la decisión de realizar un remuestreo del conjunto desbalanceado mediante técnicas de over-sampling. La selección de un buen conjunto de patrones originales a partir de los cuales generar los sintéticos es altamente importante, tanto para tener una mayor certeza de que los sintéticos se generen en las zonas

adecuadas, como para conseguir que estos aporten una información más útil y se consigan mejores resultados. Consecuentemente, deseamos guiar la búsqueda de los patrones originales como un problema de optimización aplicando la metaheurística de *Iterated Greedy* (para más información consultar el apartado 3.3. Metaheurísticas).

Dado que la metaheurística utilizada tiene un componente voraz, es preciso diseñar adecuadamente un proceso de destrucción y construcción; de lo contrario, podríamos quedarnos confinados en un óptimo local o incluso en la solución inicial, reconstruyendo una y otra vez la misma solución e impidiendo que Iterated-Greedy pueda evolucionar la solución hacia un mejor conjunto. Al mismo tiempo, es posible que ocurra sobreentrenamiento debido al fitness utilizado durante las continuas iteraciones de la metaheurística.

Durante la reconstrucción de la solución se utiliza el mismo algoritmo de over-sampling utilizado en la fase inicial, solo que en esta ocasión, la selección de patrones originales estará condicionada por la parte sobreviviente al procedo de destrucción y solo se generaran el número de sintéticos necesarios para completar la solución parcial.

7.3. Entrenamiento del clasificador

Una vez se han generado todos los sintéticos necesarios, podremos unirlo al conjunto de *Train* y de esa forma conseguir un nuevo conjunto con una distribución equilibrada. Con este nuevo conjunto podremos entrenar un clasificador y esperar que sus resultados sean mejores que si se hubiera entrenado simplemente con el conjunto de *Train*, como ya se comentó en el apartado 7.1. Seguidamente, se empleará dicho clasificador entrenado para evaluar su eficacia con el conjunto de *Test*.

Capítulo 8:
Especificación de requisitos.

En esta parte de la documentación de nuestro Trabajo, se ofrecerá una visión más detallada de la aplicación que vamos a desarrollar.

8.1. Catálogo de objetivos del sistema

Para este trabajo, no se han utilizado casos de uso para describir los requisitos funcionales del sistema, debido a que este tipo de descripción no se ajusta demasiado a un proyecto de investigación. Los objetivos que debemos desarrollar en la implementación de la aplicación son los que se mostrarán a continuación.

- **Objetivo 01:** Desarrollo de un algoritmo de over-sampling capaz de generar patrones sintéticos de forma satisfactoria.
    - **Objetivo 01.1:** Diseño e implementación de un algoritmo de over-sampling.
    - **Objetivo 01.2:** Aplicación de la metaheurística voraz iterativa al proceso de creación de patrones sintéticos.

- **Objetivo 02:** El sistema debe permitir que el usuario configure algunos aspectos de su funcionamiento para adaptarlo a sus necesidades.

- **Objetivo 03:** Realizar un proceso de clasificación sobre el conjunto de test.
    - **Objetivo 03.1:** Realizar un entrenamiento de un clasificador utilizando la unión del conjunto de *Train* con el conjunto de patrones sintéticos.
    - **Objetivo 03.2:** Utilizar el clasificador entrenado para clasificar el conjunto de *Test*.

- **Objetivo 04:** Calcular unos evaluadores de la clasificación realizada.

Objetivo 01	**Sobremuestreo**
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	El sistema será capaz de crear un conjunto de sintéticos que ayuden en el proceso de entrenamiento de algún clasificador para posteriormente evaluar la clasificación sobre el conjunto de *Test* y nuevas instancias.
Subobjetivos	• Objetivo 01.1: Algoritmo de over-sampling. • Objetivo 01.2: Metaheurística voraz iterativa

Tabla 4: Objetivo relativo al proceso de sobremuestreo de la clase minoritaria

Objetivo 01.1	Algoritmo de over-sampling
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	Este algoritmo será capaz de generar de forma eficiente y eficaz un conjunto de patrones sintéticos, poniendo énfasis en potenciar el borde que separa ambas clases y evitando en la medida de lo posible generar sintéticos en zonas poco deseadas.
Subobjetivos	Ninguno.
Comentarios	Conviene que el proceso sea lo más sencillo posible ya que será evolucionado mediante una metaheurística y por tanto se ejecutará en multitud de ocasiones.

Tabla 5: Objetivo relativo al procedimiento de creación de patrones sintéticos

Objetivo 01.2	Metaheurística voraz iterativa
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	El proceso de creación de patrones sintéticos será optimizado mediante una metaheurística voraz iterativa diseñada exclusivamente para tal fin.
Subobjetivos	Ninguno.

Tabla 6: Objetivo relativo a la optimización del algoritmo de over-sampling mediante una metaheurística

Objetivo 02	Opciones de configuración
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	Podrán cambiarse algunos parámetros que afectarán tanto a los resultados de los algoritmos como al funcionamiento general de la aplicación.
Subobjetivos	Ninguno
Comentarios	Para configurar algunos parámetros es recomendable tener ciertos conocimientos previos sobre algoritmos de aprendizaje automático.

Tabla 7: Objetivo relativo a la configuración del algoritmo

Objetivo 03	Clasificación
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	La aplicación debe contar con un clasificador y realizar una fase de entrenamiento con su posterior test.
Subobjetivos	• Objetivo 03.1: Entrenamiento con sintéticos • Objetivo 03.2: Test del clasificador
Comentarios	El clasificador a utilizar será un árbol de decisión, pero podría ser cualquier otro que se crea conveniente.

Tabla 8: Objetivo relativo a la utilización de un clasificador

Objetivo 03.1	Entrenamiento con sintéticos
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	El clasificador entrenará con la unión de los patrones de *Train* y el conjunto de sintéticos generados.
Subobjetivos	Ninguno.
Comentarios	La unión de conjunto da como resultado un conjunto balanceado con una distribución de las clases similar.

Tabla 9: Objetivo relativo al entrenamiento del clasificador

Objetivo 03.2	Test del clasificador
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	La aplicación utilizara el clasificador entrenado en [Tabla 9] para intentar predecir las etiquetas del conjunto de *test*.
Subobjetivos	Ninguno.
Comentarios	-

Tabla 10: Objetivo relativo al test del clasificador

Objetivo 04	Evaluación de la clasificación
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Descripción	El sistema debe ser capaz de calcular diferentes medidores de precisión para poder evaluar el resultado de la clasificación final.
Subobjetivos	Ninguno.
Comentarios	-

Tabla 11: Objetivo relativo a los evaluadores de la clasificación

8.2. Catálogo de requisitos del sistema

8.2.1. Requisitos de la información

Dado que la aplicación a desarrollar no se trata de un sistema de información, los requisitos de información son escasos. En concreto son dos y se muestran en las siguientes tablas.

- IRQ-01: Gestión de los datos (Tabla 12)
- IRQ-02: Gestión de los resultados (Tabla 13)

IRQ-01	**Gestión de los datos**
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 02: Opciones de configuración (Tabla 7).
Descripción	El sistema deberá ser capaz de trabajar con conjunto con gran cantidad de patrones o instancias donde cada una posee una serie de atributos. Las instancias se repartirán por las diferentes filas del conjunto, mientras que los atributos lo harán por las columnas. La última de ellas se reserva para la etiqueta de clase.
Información	El conjunto será introducido en el sistema utilizando un archivo externo

Tabla 12: Requisito de la información relativo al formato de entrada de datos

IRQ-02	**Gestión de los resultados**
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 02: Opciones de configuración (Tabla 7). • Objetivo 03: Clasificación (Tabla 8). • Objetivo 03.2: Test del clasificador (Tabla 10). • Objetivo 04: Evaluación de la clasificación (Tabla 11).
Descripción	La aplicación deberá dar la opción al usuario de mostrar por pantalla o almacenar la información obtenida al final del proceso. La información estará contenida en una tabla donde los evaluadores de precisión se dividirán entre las diferentes filas.
Información	El archivo se almacenara en la ruta de ejecución.
Comentarios	La información debe estar ordenada y clara para facilitar su estudio e interpretación.

Tabla 13: Requisito de la información relativo a los resultados

8.2.2. Requisitos funcionales

En esta sección se mostraran los requisitos funcionales que se han identificado.

- **FRQ 01:** Segmentación.
- **FRQ-02:** Única aparición.
- **FRQ-03:** Cambio de dimensión.
- **FRQ-04:** Parámetros por defecto.
- **FRQ-05:** Ejecución del algoritmo.

FRQ-01	**Segmentación.**
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5Tabla 4) • Objetivo 02: Opciones de configuración (Tabla 7).
Descripción	El sistema deberá ser capaz de leer la base de datos y prepararla para poder aplicar el algoritmo de over-sampling. El sistema distinguirá perfectamente entre las entradas (características de los patrones) y las salidas (etiquetas de clase). Además dividirá el conjunto de datos de forma estratificada en 3 subconjuntos *Train*, *Test* y *Validation*, manteniendo la misma proporción de las clases en cada subconjunto y sin repetir elementos entre ellos
Comentarios	Todos los cambios realizados al conjunto original deben ser a nivel local y en ningún caso comprometer la integridad del conjunto original.

Tabla 14: Objetivo relativo a la preparación de los datos de entrada

FRQ-02	**Única aparición**
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01.2: Metaheurística voraz iterativa (Tabla 6) • Objetivo 03: Clasificación (Tabla 8). • Objetivo 03.1: Entrenamiento con sintéticos (Tabla 9). • Objetivo 04: Evaluación de la clasificación (Tabla 11).
Descripción	El conjunto de datos original no podrá tener instancias duplicadas que puedan influir negativamente en el aprendizaje.
Comentarios	La existencia de patrones duplicados puede afectar negativamente al proceso de clasificación, más aun si durante la segmentación estos se reparten por los diferentes conjuntos de *Train*, *Test* o *Validation*. En tal caso los resultados conseguidos perderían fiabilidad.

Tabla 15: Requisito funcional relativo a la duplicidad en los datos

FRQ-03	Cambio de dimensión
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 02: Opciones de configuración (Tabla 7).
Descripción	En el caso de que el conjunto original tenga más de dos clases, será necesario realizar una modificación en las etiquetas de clase (*outputs*). Estas etiquetas pasarán a identificarse como pertenencia a la "clase minoritaria" (positiva) o a la "clase mayoritaria" (negativa), de modo que al final del proceso solo existan dos clases**.**
Comentarios	▪ Este cambio no debe alterar de ningún modo a las entradas del conjunto (inputs). ▪ Para realizar esta modificación de forma satisfactoria se requiere una configuración por parte del usuario.

Tabla 16: Requisito funcional relativo al cambio de la dimensión del problema

FRQ-04	Parámetros por defecto
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 01.2: Metaheurística voraz iterativa (Tabla 6) • Objetivo 02: Opciones de configuración (Tabla 7). • Objetivo 03: Clasificación (Tabla 8).
Descripción	La aplicación tendrá preestablecidos unos parámetros por defecto para todo el algoritmo. No obstante, dichos valores podrán ser modificados por el usuario para afectar tanto al funcionamiento del algoritmo como a los resultados obtenidos.
Comentarios	Existirá una configuración por defecto que si bien no es prefecta, funcionara decentemente en la mayoría de los casos.

Tabla 17: Requisito funcional relativo a poder configurar el algoritmo

FRQ-05	Ejecución del algoritmo
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 01.2: Metaheurística voraz iterativa (Tabla 6) • Objetivo 02: Opciones de configuración (Tabla 7). • Objetivo 03: Clasificación (Tabla 8). • Objetivo 03.1: Entrenamiento con sintéticos (Tabla 9). • Objetivo 03.2: Test del clasificador (Tabla 10). • Objetivo 04: Evaluación de la clasificación (Tabla 11).
Descripción	El sistema debe ejecutarse correctamente y realizar su cometido según los parámetros establecidos.

Tabla 18: Requisito funcional relativo a la ejecución del algoritmo

8.2.3. Requisitos no funcionales

En esta sección se van a detallar los requisitos no funcionales que se exigen en el sistema que se está desarrollando.

- **NFRQ-01:** Entorno de programación *Python.*
- **NFRQ-02:** Fiabilidad.
- **NFRQ-03:** Formatos de entrada de los datos.
- **NFRQ-04:** Formato de almacenamiento de los resultados.
- **NFRQ-05:** Optimización.

NFRQ-01	Entorno de programación *Python*.
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 01.2: Metaheurística voraz iterativa (Tabla 6) • Objetivo 02: Opciones de configuración (Tabla 7). • Objetivo 03: Clasificación (Tabla 8). • Objetivo 03.1: Entrenamiento con sintéticos (Tabla 9). • Objetivo 03.2: Test del clasificador (Tabla 10). • Objetivo 04: Evaluación de la clasificación (Tabla 11).
Descripción	Implementación del sistema utilizando *Python*.
Comentarios	Se ha elegido *Python* como lenguaje de programación por ser lo suficientemente flexible para implementar algoritmos de inteligencia artificial y aprendizaje automático, y por tener una gran de librerías existentes y en continuo desarrollo. (5.2 Factores estratégicos)

Tabla 19: Requisito no funcional relativo al lenguaje de programación

NFRQ-02	Fiabilidad
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 01.2: Metaheurística voraz iterativa (Tabla 6) • Objetivo 03: Clasificación (Tabla 8). • Objetivo 03.2: Test del clasificador (Tabla 10). • Objetivo 04: Evaluación de la clasificación (Tabla 11).
Descripción	La aplicación no deberá comprometer en ningún caso los archivos originales y deberá asegurar una salida correcta.

Tabla 20: Requisito funcional relativo a la fiabilidad

NFRQ-03	Formatos de entrada de los datos
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 02: Opciones de configuración (Tabla 7).
Descripción	La aplicación debe recibir una base de datos en formato CSV.
Comentarios	El usuario será quien decida el archivo que se le pase al sistema.

Tabla 21: Requisito no funcional relativo al formato de entrada de los datos

NFRQ-04	Formato de almacenamiento de los resultados
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 02: Opciones de configuración (Tabla 7). • Objetivo 04: Evaluación de la clasificación (Tabla 11).
Descripción	La aplicación podrá almacenar los resultados obtenidos en un archivo CSV para su posterior análisis.
Comentarios	El usuario podrá elegir el nombre de dicho archivo.

Tabla 22: Requisito no funcional relativo al formato de salida de los datos

NFRQ-05	Optimización
Versión	1.0
Autores	Francisco Javier Maestre García
Fuentes	Pedro Antonio Gutiérrez Peña Carlos García Martínez
Dependencias	• Objetivo 01: Sobremuestreo (Tabla 4) • Objetivo 01.1: Algoritmo de over-sampling (Tabla 5) • Objetivo 01.2: Metaheurística voraz iterativa (Tabla 6) • Objetivo 03: Clasificación (Tabla 8). • Objetivo 03.1: Entrenamiento con sintéticos (Tabla 9).
Descripción	El algoritmo de sobremuestreo y la clasificación deben estar optimizados en coste computacional y resultados.

Tabla 23: Requisito no funcional relativo a la optimización

8.3. Matriz de trazabilidad de requisitos

Esta sección presenta la matriz de trazabilidad de requisitos (**¡Error! No se encuentra el origen de la referencia.**), que se ha considerado oportuna para así relacionar los objetivos y los distintos requisitos de la información, funcionales y no funcionales.

TRM-0001	OBJ-01	OBJ-01.1	OBJ-01.2	OBJ-02	OBJ-03	OBJ-03.1	OBJ-03.2	OBJ-04
IRQ-01	X	X	X					
IRQ-02				X	X		X	X
FRQ-01	X	X		X				
FRQ-02			X		X	X		X
FRQ-03	X	X		X				
FRQ-04		X	X	X	X			
FRQ-05	X	X	X	X	X	X	X	X
NFRQ-01	X	X	X	X	X	X	X	X
NFRQ-02	X	X	X		X		X	X
FRQ-03				X				
NFRQ-04				X				X
NFRQ-05	X	X	X		X	X		

Tabla 24: Matriz de trazabilidad de requisitos y objetivos

Capítulo 9:
Herramientas utilizadas

Tras haber realizado un estudio previo de las principales tecnologías existentes, y evaluado las ventajas y desventajas, en el presente capítulo se describirán cuáles de ellas se utilizarán en el desarrollo de nuestra aplicación.

9.1. Lenguaje de programación: *Python*

Python es uno de los principales lenguajes de programación más conocidos en la actualidad. Se encuentra entre los más populares y demandados junto a Perl, Tcl, PHP o Ruby. Este lenguaje, a menudo, suele ser considerado como lenguaje de scripting[4] a pesar de que realmente es un lenguaje de propósito general.

En la actualidad *Python* tiene un amplio uso: desde crear sencillos *scripts* hasta programación para grandes servidores que proveen servicios web de forma continuada e ininterrumpida.

Científicos e ingenieros del todo el mundo diseñan y programan cada día aplicaciones con un alto nivel de complejidad computacional para ser ejecutados en supercomputadoras, y lo hacen en *Python* debido a su sencillez [26]. De hecho, este es uno de los lenguajes que se emplean en la enseñanza de los más pequeños que se inician desde su infancia en el mundo de la programación [27].

Python cuenta con una gran comunidad, por lo que es fácil recurrir a internet en busca de los conocimientos necesarios para poder saciar las necesidades de los usuarios. Esta comunidad también ayuda a mantener *Python* actualizado a través de la creación de librerías muy variadas con todo tipo de funciones y que pueden ser utilizadas en diversos ámbitos.

9.1.1. Historia

El inicio de este lenguaje de programación se remonta a finales de los años ochenta y principios de los noventa. Su implementación comenzó en el mes de diciembre de 1989 (Holanda), cuando Guido Van Rossum decidió comenzar un proyecto, en principio por entretenimiento, que consistía en darle continuidad a un lenguaje llamado ABC, que utilizaba en su trabajo en el CWI (*Centrum Wiskunde & informática*) [28] en Ámsterdam.

[4] Un lenguaje de scripting o lenguaje interpretado, es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete. No utiliza un compilador, por lo que se genera ningún archivo compilado.

Dicho lenguaje era muy fácil de aprender y de usar, por lo tanto, el proyecto que inicio Rossum también debía serlo, pero la tecnología hardware de la época no era muy apta para poder continuar con ABC de forma sencilla.

Van Rossum, por consiguiente, creo un nuevo lenguaje de programación y lo bautizo como *Python* en honor a su afición por el grupo de humoristas "Los Monty Python". En su origen, *Python* fue concebido para ejecutarse bajo el sistema operativo llamado *Amoeba*, pero en la actualidad, cualquier sistema operativo es capaz de ejecutar un código escrito en *Python*.

9.1.2. Filosofía

Los Usuarios de *Python*, a menudo hacen referencia a la "Filosofía de Python" que es bastante similar a la de *Unix*. A un código que siga los principios de legibilidad y transparencia se dice que es "*Pythonico*", y al contrario, cuando el código es opaco y ofuscado se le denomina "*no Pythonico*" ("*unpythonic*" en inglés).

Los siguientes principios fueron popularmente descritos por el desarrollador de *Python* Tim Peters en "El Zen de *Python*" [29]:

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Plano es mejor que anidado.
- Disperso es mejor que denso.
- La legibilidad cuenta.
- Los casos especiales no son tan especiales como para quebrantar las reglas.
- Lo práctico gana a lo puro.
- Los errores nunca deberían dejarse pasar silenciosamente.
- A menos que hayan sido silenciados explícitamente.
- Frente a la ambigüedad, rechaza la tentación de adivinar.
- Debería haber una (y preferiblemente sólo una) manera obvia de hacerlo.
- Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
- Ahora es mejor que nunca.
- Aunque nunca es a menudo mejor que ya mismo.
- Si la implementación es difícil de explicar, es una mala idea.
- Si la implementación es fácil de explicar, puede que sea una buena idea.
- Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

9.1.3. Características

Entre las características principales atribuibles al lenguaje de programación *Python*, podemos comentar las siguientes:

- **Simple:** *Python* es un lenguaje simple y minimalista. La lectura de un buen código en *Python* se debe sentir casi como si de un texto en ingles se tratara, aunque un inglés especialmente estricto. El Pseudocódigo natural de *Python* es una de sus mayores fortalezas, ya que permite concentrarse en la implementación de la solución, en lugar de aspectos relacionados con el propio lenguaje.

- **Fácil de entender:** *Python* tiene una sintaxis realmente simple que facilita su aprendizaje y su uso.

- **Libre y de código abierto:** *Python* es un ejemplo de un FLOSS (*Free / Libre y Open Source Software*), es decir, que es posible distribuir libremente copias de este *software*, leer el código fuente, modificarlo y usar fragmentos de él en nuevos programas libres. FLOSS está basado en el concepto de una comunidad que comparte conocimiento libremente. Esta es una de las razones por las que *Python* es uno de los lenguajes más queridos y usados, una gran comunidad de desarrolladores que ayudan día a día a que este mejore y se expanda.

- **Portable:** Debido a su naturaleza de código abierto y a su gran comunidad, *Python* ha podido ser portado a muchas plataformas como*: Linux, Windows, Macintosh, Solaris, Amiga; AROS, OS/2, AS/400, BeOS, OS/390, z / OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS, VxWorks, PlayStation, Sharp Zaurus, Windows CE*, e incluso *PocketPC*. Aun así, cabe destacar que podrían crearse dependencias al usar librerías propias de un sistema determinado, en ese caso, la portabilidad podría verse comprometida.

- **Interpretado:** Este tipo de lenguajes, no requieren crear un archivo de compilación para ser ejecutado por el sistema, ya que consisten en "*scripts*" que son interpretados en tiempo real por un intérprete desde el propio código fuente. *Python* internamente convierte el código fuente en un formato intermedio llamada bytecodes, luego traduce este en el lenguaje nativo de la computadora y por último lo ejecuta. Todo el proceso de ejecución es totalmente transparente tanto para el usuario como para el programador.

- **Orientado a objetos:** *Python* permite la programación orientada a procedimientos así como la programación orientada a objetos. En los lenguajes orientados a procedimientos, el código está construido sobre "procedimientos" o "funciones" que se describen como fragmentos reutilizables del código. En los lenguajes orientados a objetos, el código es construido utilizando "objetos", capaces de compartir y combinar datos y funcionalidades. *Python* es muy

poderoso y simple a la hora de hacer programas orientados a objetos, especialmente cuando se compara con grandes lenguajes como C++ o Java.

- **Extensible:** Si para el funcionamiento de un software es fundamental codificar parte de él en otro lenguaje como C o C++, es posible utilizar esa codificación desde el propio programa en *Python*.

- **Bibliotecas extendidas:** La biblioteca estándar de *Python* es muy amplia. Puede realizar funciones muy variadas relacionados con expresiones regulares, generación de documentos, pruebas, procesos, bases de datos, navegadores web, CGI, FTP, correo electrónico, XML, XML-RPC, HTML, archivos WAV, criptografía, GUI (interfaz gráfica de usuario) utilizando TK, y algunas otras funcionalidades que dependen del sistema. Todo esto está disponible donde quiera que *Python* está instalado, dado que forma parte de la filosofía del lenguaje, "*batteries included*" (pilas incluidas).

- **Sintaxis clara:** Cualquier código escrito en *Python* posee una sintaxis muy clara y visual, debido en gran parte a que posee una sintaxis indexada de carácter obligatorio. Está indexación es muy agradable visualmente y favorece el hecho de que todos los códigos escritos en *Python* tengan un aspecto visual similar, lo que facilita su lectura a todos aquellos que estén familiarizados con *Python*. Se pone mucho énfasis en la facilidad de uso, documentación y la consistencia de la API.

9.2. Librerías de *Python*

Como se ha comentado anteriormente, *Python* dispone de una gran variedad de librerías o bibliotecas con multitud de funciones, creadas en la mayoría de los casos por la propia comunidad.

Algunas de las librerías usadas en este Trabajo Fin de Grado son las siguientes:

9.2.1. Scikit-learn

Según Fabian Pedrejosa [8], *Scikit-learn* es un módulo de *Python* que integra una amplia gama de los algoritmos de aprendizaje automático más actuales, ya sea para problemas supervisados como no supervisados.

La librería tiene un número mínimo de dependencias y se distribuye con una licencia BSD[5] (Berkeley Software Distribution), poniendo interés tanto en el mundo académico como en el comercial. Se puede acceder al código fuente, archivos binarios y toda su documentación a través de la dirección*: http://scikit-learn.org.*

[5] La licencia BSD permite el uso del código fuente en software no libre, mientras que la GLP (General Public License) no lo permite [30].

Los algoritmos incluidos en *Scikit-learn* poseen un alto nivel de abstracción, por lo que su utilización en diversos problemas es sencilla y al alcance de muchos desarrolladores.

9.2.2. Scipy

En los años 90 *Python* fue ampliado mediante la creación de múltiples funciones destinamos a los modelos computacionales y algoritmos matemáticos mediante una librería llamada *Numeric*. En el año 2006, Travis Oliphant creó *NumPy* basándose en sus antecesores *Numeric* y *Numarray*, proyecto que comenzó varios años antes, en 2001 [9].

En 2001 Travis Oliphant, Eric Jones y Pearu Peterson fusionaron el código que habían escrito y llamaron al paquete resultante *SciPy*. El paquete recién creado proporcionaba una colección estándar de operaciones para el tratamiento y la utilización de conjunto de datos. Poco después John Hunter lanzó la primera versión de *Matplotlib*, la biblioteca para trazado de gráficas 2D [11]. Desde entonces, las herramientas pertenecientes al entorno de *SciPy* han seguido creciendo de forma constante con la inclusión de más paquetes y actualizaciones. Se puede acceder a la información de este proyecto a través del enlace https://www.scipy.org/

9.2.3. Pandas:

Pandas es una librería open source (BSD) para la manipulación y análisis de grandes estructuras de datos [10]. Wes McKinney comenzó a trabajar en *Pandas* en 2008, mientras trabajaba en *AQR Capital*[6]. La entidad necesitaba una herramienta de alto rendimiento y flexible para utilizarlo en análisis financiero. Antes de abandonar la entidad, McKinney consiguió convencer a la gerencia para que le permitiera abrir la biblioteca.

Otro empleado de AQR, Chang She, se unió al proyecto en 2012 como el segundo mejor contribuyente a la biblioteca. Sobre esa misma época la biblioteca se hizo popular en la comunidad de *Python*, y muchos más colaboradores se unieron al proyecto. Se puede acceder a la información de este proyecto a través del enlace http://pandas.pydata.org/

[6] AQR Capital Management es un fondo de inversiones fundado en 1998 por antiguos ejecutivos de Goldman Sachs con sede en Greenwich (Connecticut).

Parte III
Diseño del sistema

Capítulo 10:
Análisis del algoritmo y metaheurística

En este capítulo se describirá cómo ha sido diseñado tanto el algoritmo de over-sampling como la metaheurística voraz iterativa. Además se explicará cómo se realiza el proceso de clasificación y su evaluación. De manera adicional, al final del capítulo se incluirá un pseudocódigo reflejando la funcionalidad a desarrollar.

10.1. Algoritmo de over-sampling

Como ya se dijo anteriormente, para nuestro algoritmo de over-sampling se ha elegido como base ADASYN [3], ya que este presenta una serie de características que deseamos explotar y mejorar en la creación de nuestro sistema (consultar el apartado 5.2. Factores estratégicos.). Resulta muy interesante la forma que ADASYN gestiona la importancia de que un patrón esté localizado cerca del borde, y como le asigna un numero diferente de patrones sintéticos a generar dependiendo de dicha importancia.

A lo largo de este apartado, se explicará cómo se desarrolla el algoritmo que se ha diseñado en este TFG, y cuales han sido las consideraciones tenidas en cuenta a lo largo de su implementación para llegar a ciertas conclusiones. Se ha decidido definir su funcionamiento utilizando una explicación estructurada, dividiéndolo en los diferentes pasos más importantes que se pueden extraer de su desarrollo.

Paso 1

Se calcula la vecindad de cada patrón de la clase minoritaria. Es importante saber tanto la etiqueta y el índice de los *k* vecinos más cercanos, como la distancia a la que está cada uno.

1º. El número de patrones que se analizan dentro de la vecindad es configurable por el usuario. No obstante el sistema cuenta con un método para calcular cuántos patrones deben der tomados en cuenta según la distribución de la base de datos. Este método funciona del siguiente modo:

   A.1. Por cada $i \in m$ (donde m son todos los patrones originales de la clase minoritaria) se busca el vecino de la clase mayoritaria más cercano, y se almacena que posición representa dicho vecino dentro los vecinos más cercanos de i (j_{iM}). Por ejemplo, para un patrón cualquiera i donde su vecino de la clase mayoritaria más cercano es el 4º vecino más cercano de i, la asignación sería $j_{iM} = 4$.

A.2. Una vez obtenido el valor de j_{iM} para todo $i \in m$ (el vector j_{mM}) *k* sería igual al máximo de ese vector ($k = \max j_{mM}$). De forma que cada patrón i tendrá al menos un vecino de la clase mayoritaria dentro de sus *k* vecinos más cercanos.

Posteriormente, será la función de asignación de importancia la que decida que patrones son más aptos para generar sintéticos.

Paso 2

Una vez obtenido *k,* y las etiquetas de los *k* vecinos más cercanos, es posible calcular su ratio de cercanía al borde de la siguiente forma:

$$r_i = \frac{majority\ neighbors}{k}$$

donde r_i es el ratio de cercanía al borde, *k* es el número de patrones tenidos en cuenta y $majority\ neighbors$ es el número de vecinos de la clase mayoritaria dentro de los *k* vecinos más cercanos ($0 \leq majority\ neighbors \leq k$).

Paso 3

Se calcula cuantos sintéticos se deben generar para cada patrón minoritario del conjunto original utilizando las definiciones:

$$\hat{r}_i = {}^{r_i}/_{\sum_z^m r_z}$$

donde m hace referencia a todos los patrones originales de la clase minoritaria, $\hat{r}_i$ es el ratio de cercanía al borde normalizado del patrón i ($i \in m$), y $\sum_z^m r_z$ es la suma del ratio de cercanía al borde de todos los patrones de m. Después se utiliza:

$$g_i = \hat{r}_i \cdot G$$

donde g_i es el número de sintéticos que debe generar el patrón i, y *G* es el número total de sintéticos a generar (definido por el usuario). Si bien esta fórmula es un buen método para calcular el valor de g_i, su resultado solo puede ser tomado como un valor teórico o de referencia, ya que probablemente aparezcan decimales en el número de patrones a generar. Si existiesen decimales para algunos valores de g_i, debe procederse del siguiente modo:

1º. La parte entera de cada valor de g_i es asignada inmediatamente.

2º. La parte decimal es descartada.

3º. Se calcula la diferencia entre todas las partes enteras asignadas y el número total de sintéticos que deben ser generados (G). Este valor debe corresponderse con la suma de las diferentes partes decimales descartadas.

4º. Se asignan los sintéticos restantes calculados en el paso anterior utilizando un modelo probabilístico, de forma que un patrón original i tenga más posibilidades de recibir una nueva asignación cuanto más alta sea su parte decimal calculada en g_i.

Al finalizar este proceso de reasignación, podemos obtener un nuevo valor de g_i al que llamaremos $\bar{g}_i$ para todo $i \in m$ sin que existan decimales.

Paso 4

Una vez se han calculado $\bar{g}_i$, es hora de generar los patrones sintéticos. Como ya se ha comentado en varios puntos de este documento, algunas técnicas de over-sampling pueden llegar a generar patrones sintéticos en zonas poco adecuadas dando lugar a una superposición de las clases. En la Figura 2, podemos observar una imagen gráfica donde se da superposición de clases.

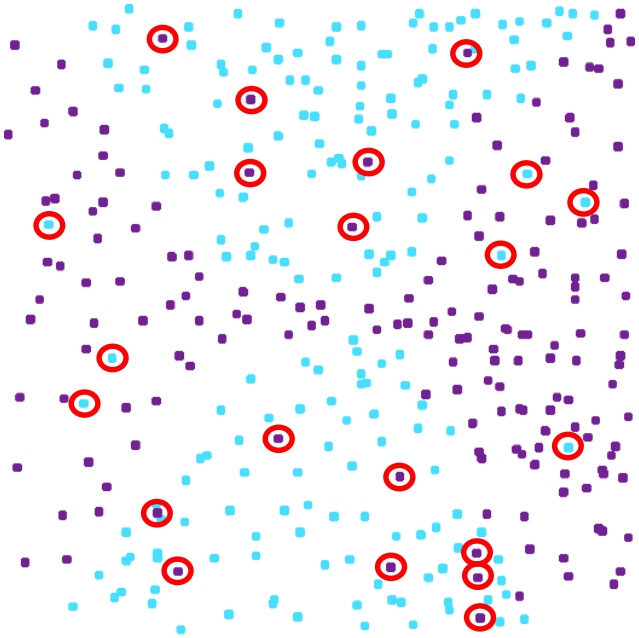

Figura 2: Superposición de clases

Esto se provoca debido a que la elección del vecino con el que se genera un sintético sigue un procedimiento puramente aleatorio sin tener en cuenta la distribución del espacio. Nuestro algoritmo intentara evitar esta situación generando sintéticos como se explica a continuación.

Utilizando $\bar{g}_i$, seleccionamos aquellos patrones con $\bar{g}_i$ diferente de 0, y se proceden a elegir los vecinos utilizando el siguiente procedimiento:

$$\varepsilon_j = \left(n \cdot \hat{f}_{j1}\right) + \left((1-n) \cdot \hat{f}_{j2}\right)$$

donde ε_j representa la probabilidad de elegir al vecino j. Los factores $\hat{f}_{j1}$y $\hat{f}_{j2}$ hacen referencia a dos factores distintos de importancia y n es una variable de ponderación que equilibra la influencia entre $\hat{f}_{j1}$y $\hat{f}_{j2}$ (normalmente n = 0.5, aunque es un valor a configurar por el usuario). Concretamente, f_{j1} es el ratio de cercanía al borde normalizado del vecino j ($\hat{r}_j$) y f_{j2} es el inverso de la distancia que separa al patrón i de su vecino j:

$$f_{j1} = \hat{r}_j$$
$$f_{j2} = {}^{1}/_{dist_{j-i}}$$

donde j es uno de los *k* vecinos más cercanos del patrón i ($j \in k$) y $dist_{j-i}$ es la distancia entre el patrón i y el vecino j. Una vez calculados f_{j1} y f_{j2}, es muy importante normalizarlos con respecto a la suma de estos factores de los *k* vecinos, para que se encuentren en la misma escala y tengan la misma influencia en el cálculo de la importancia general del patrón. De esta forma, $\hat{f}_{j1}$y $\hat{f}_{j2}$ quedan definidos:

$$\hat{f}_{j1} = {}^{f_{j1}}\Big/_{\sum_z^k f_{z1}}$$
$$\hat{f}_{j2} = {}^{f_{j2}}\Big/_{\sum_z^k f_{z2}}$$

De este modo, se busca que el vecino con el que se genera el sintético esté cerca del borde que separa ambas clases para reforzar fuertemente esta zona gracias a $\hat{f}_{j1}$, pero al mismo tiempo, a través de $\hat{f}_{j2}$ se orienta a que el vecino j esté lo más cerca posible del patrón i, para así evitar que se genere en una zona de conflicto. De esta forma, utilizando la ponderación de n, se desea generar patrones sintéticos en zonas adecuadas, pero sin recurrir a técnicas más costosas y que limitan la zona de creación de sintéticos como pasa con MWMOTE[4].

Cuando se han seleccionado los $\bar{g}_i$ vecinos del patrón i, es hora de generar los sintéticos. Utilizaremos la misma interpolación definida en SMOTE [2]:

$$S_{ig} = x_i + (x_{ji} - x_i) \cdot \lambda$$

donde recordamos que S_{ig} es el sintético que se genera en algún punto de la recta que separa a los patrones originales x_{ji} y x_i. Se generará un sintético por cada $\bar{g}_i$ calculado anteriormente. Al conjunto de sintéticos de S_{ig} que se forman para todo $i \in m$ lo llamaremos S. Posteriormente se etiquetará a todo el conjunto S como perteneciente a la clase minoritaria.

Paso 5

El siguiente paso será unir todos los patrones sintéticos S al conjunto de *Train* y realizar una validación, considerando ya sea el G-mean del conjunto o el AUC. Según el resultado obtenido, y teniendo en cuenta que este paso se ejecutará repetidas veces debido a la

metaheurística, se almacenará el conjunto de sintéticos capaz de conseguir el mejor resultado en la validación, y también el árbol entrenado con el que se consiguió dicho resultado.

10.2. Aplicación de Iterated Greedy

Iterated Greedy ha sido la metaheurística elegida para optimizar el proceso de generación de patrones sintéticos. Inmediatamente, pasaremos a explicar cómo se ha implementado la metaheurística voraz iterativa a nuestro sistema apoyándonos en los pasos explicados en el apartado 10.1. Algoritmo de over-sampling.

Paso 6

Después de realizar la validación, es turno de ejecutar el destructor de *Iterated Greedy*. El destructor eliminará un 15% de todos los patrones sintéticos pertenecientes a S de una forma completamente aleatoria sin utilizar ningún método de selección. Llamaremos a al conjunto de sintéticos supervivientes S_b de modo que $S_b \subseteq S$.

Paso 7

Finalmente, se calcula unos nuevos valores de $\hat{r}_i$ para todo $i \in m$ tal y como se explica en los siguientes párrafos, y se vuelve a ejecutar todo a partir del **Paso 3**. De esta forma, al calcular los diferentes valores de g mediante la utilización de los nuevos valores de $\hat{r}$, se llegará a un nuevo resultado y un conjunto diferente de sintéticos S.

Antes de volver a ejecutar el constructor, debemos asegurarnos de implementar un método que haga que la reconstrucción avance hacía una solución diferente a la destruida anteriormente, para de esta forma explorar zonas desconocidas del espacio de soluciones del problema, que en este caso son nuevos conjuntos de sintéticos. Dicho esto, nuestro objetivo será llegar a calcular un valor diferente de $\bar{g}_i$, de manera que las asignaciones de los patrones sintéticos sean distintas en las continuas iteraciones de la metaheurística.

Para conseguirlo se ha optado por utilizar los sintéticos S_b para recalcular del ratio de cercanía al borde (número de vecinos mayoritarios / *k*) de los patrones minoritarios originales. De modo que ahora los sintéticos que sobreviven a la fase de destrucción (S_b) son tenidos en cuenta en el cálculo de la vecindad. En esta nueva exploración de la vecindad se calculan los vecinos de los patrones de m (clase minoritaria) con respecto a la unión de los conjuntos de $m \cup S_b$, utilizando el método automático explicado en el **Paso 2**.

De esta forma, los patrones cercanos a las zonas donde están localizados estos sintéticos perderían importancia debido a que en su vecindad ahora podrían aparecer estos sintéticos, y disminuiría las apariciones de la clase mayoritaria. Al normalizar el nuevo ratio de cercanía al borde y obtener $\hat{r}_i$, se distribuye la asignación de sintéticos hacia zonas donde todavía no han sido creados patrones sintéticos, es decir, se potencian zonas donde aun después de la

generación de los sintéticos la densidad de patrones minoritarios sigue siendo escasa. Cabe destacar que los patrones del conjunto de S_b, no son utilizados para generar nuevos sintéticos, sino que solo son tenidos en cuenta en la vecindad de los patrones originales para recalcular su cercanía al borde. Esto es debido a que la información contenida en estos patrones sintéticos no se considera lo suficientemente fiable para la generación de nuevos patrones en base a los mismos. Finalmente, una vez explicado cómo se recalcula $\hat{r}_i$, se vuelve al **Paso 3**, para recalcular una asignación distinta de $\bar{g}_i$ y reconstruir una solución diferente.

En este caso el criterio de aceptación implementado ha sido almacenar siempre el mejor conjunto de sintéticos S, con el que se consigue la mejor validación, así como el árbol entrenado con dicho conjunto. Posteriormente se ejecutará el módulo encargado de la evaluación.

10.3. Evaluador

La parte final del sistema se corresponde con el sistema de evaluación implementado. En nuestro caso, utilizaremos un árbol de decisión que usa el criterio Gini para entrenar el árbol. El índice de diversidad de Gini trata de minimizar la impureza existente en los subconjuntos de casos de entrenamiento generados al ramificar por un atributo, por tanto, se busca minimizar este índice. El árbol construido con el indice de Gini tiende a seleccionar *splits* que aíslan una clase mayoritaria en un nodo y el resto en otros nodos. También tiende a crear subconjuntos desbalanceados y a favorecer *splits* con muchos nodos hijos. Puede funcionar de una forma menos beneficiosa con bases de datos con muchas clases debido a la gran cantidad de nodos que se crean, pero en nuestro caso solo trabajaremos con dos clases, por lo que no supondrá un problema.

Nuestro sistema unirá el conjunto de sintéticos S al conjunto de *Train,* para realizar la validación y posteriormente la evaluación utilizando el conjunto de *Test* sin modificación alguna. El sistema calculará distintos evaluadores para poder realizar un análisis más completo al finalizar la ejecución del sistema.

10.3.1. Evaluadores utilizados

Antes de pasar a comentar los evaluadores en sí mismos, es necesario aclarar en qué consiste una *Matriz de confusión*. La *Matriz de confusión* se utiliza en el aprendizaje automático para representar la calidad de la clasificación de forma gráfica y clara, y posee una estructura simple y muy sencilla. Se trata de una matriz cuadrada donde en cada columna se colocan las predicciones de cada clase calculadas durante el test o la validación del clasificador, mientras que en cada fila se colocan las instancias de las clases reales como se muestra en la Figura 3. Una de las ventajas de la matriz de confusión es que se puede ver claramente si el clasificador produce un sesgo en la clasificación hacia alguna de las clases.

		Hipótesis de salida	
		Clase positiva	Clase negativa
Clase real	Clase positiva	True positive (TP)	False Negative (FN)
	Clase negativa	False positive (FP)	True Negative (TN)

Figura 3: Matriz de confusión

Aunque la *Matriz de confusión* nos aporta una excelente visión de cómo ha ido la clasificación, no puede ser utilizada para comprar directamente resultados provenientes de conjunto de datos diferentes, ya que estos presentaran una distribución distinta de las clases que lo componen, y esto hace que la lectura a simple vista de la *Matriz de confusión* no coincida. Por esta razón, existen multitud de evaluadores que se calculan utilizando los datos que aporta la *Matriz de confusió*n, para que de este modo podamos comparar la precisión alcanzada en la clasificación. Cada evaluador pone su interés en medir la precisión de forma diferente. Así pues, se ha decidido utilizar varios de ellos para que el análisis final sea más completo. A continuación se describirán las principales características de los evaluadores utilizados.

- **Overall Accuracy (OA)**: Esta medida hace referencia a la precisión general de la clasificación. Puede ser un buen evaluador si el conjunto está balanceado, pero a medida que el conjunto presenta una distribución más desigual, este medidor pierde cada vez más fiabilidad. Un ejemplo claro son los datos estadísticos que podemos sacar acerca del Síndrome de Down. De cada 10.000 nacimientos en España en el periodo de 2011-2012, solo 5,51 nacieron con Síndrome de Down. Un método que simplemente clasifique todos los patrones como negativos (clase mayoritaria) podría conseguir un OA de 0.999, cuando realmente ha clasificado mal el 100% de la clase minoritaria (un árbol de decisión como es C4.5 podría crear solo un nodo hoja). Esta gran desventaja hace que no sea muy útil a la hora de evaluar un clasificador aplicado a un conjunto desbalanceado, como es el caso de los conjuntos utilizados en este proyecto. Su fórmula es la siguiente:

$$OA = \frac{TP + FN}{TP + FP + FN + TN}$$

- **Precisión**: Es una medida que calcula la relación que existe la clase positiva real y las instancias consideradas como positivas. Para ello, analiza de entre todos las instancias clasificadas como positivas, cuales son realmente de la clase positiva. Si se obtiene una *Precisión* alta quiere decir que el clasificador tiene una gran fiabilidad a la hora de etiquetar patrones de la positiva, de tal modo que rara vez etiqueta un patrón de la clase negativa como positiva. No obstante, puede confundirse etiquetando un patrón de la clase positiva como negativa. Se calcula de la siguiente forma:

$$\text{Precisión} = \frac{TP}{TP + FP}$$

- **Recall**: Al igual que *Precisión*, tiene en cuenta la relación existente entre la clase positiva real y las instancias consideradas como positivas, aunque de forma diferente. *Recall* calcula de todas las instancias positivas existentes, cuantas han sido clasificadas como positivas. Los valores altos de *Recall* significan que el clasificador detecta fácilmente los patrones de la clase positiva, y los etiqueta correctamente. Sin embargo, pueden darse muchos casos en los que se etiquetan instancias de la clase mayoritaria como minoritaria creando así falsos positivos. La fórmula para calcular este evaluador es como sigue:

$$Recall = \frac{TP}{TP + FN}$$

- **F-Measure**: Realmente no existe una interpretación lógica para esta medida, ya que se trata de una métrica que combina una ponderación de *Recall* y de *Precisión*. Estas medidas suelen tener una relación inversa, y buscar incrementar una de ellas supone a menudo un detrimento de la otra. Para ambas, conviene obtener un valor considerablemente alto, pero ya que presentan una naturaleza comúnmente inversa, puede interesarnos más una que otra según el escenario de aplicación. Por esta razón propuso Rijsbergen *F-Measure* en [31]. La forma de calcular esta métrica es la siguiente:

$$F = \frac{(1 + \beta^2) \cdot recall \cdot precisión}{\beta^2 \cdot recall \cdot precisión}$$

  donde β es un coeficiente para ajustar la importancia relativa de *Precisión* con respecto a *Recall* (normalmente $\beta = 1$). En este caso, elegir el valor de β es responsabilidad del usuario. Cabe destacar que una vez elegido el valor de β deseado, lo valores altos de *F-Measure* determinaran una mejor clasificación.

- **G-mean**: Otra medida que se utiliza comúnmente con conjuntos no balanceados propuesta en [32], es la media geométrica (*G-mean*), que evalúa el rendimiento en términos de *Recall* de la clase positiva y *Recall* de la clase negativa. Como ya se ha dicho antes, en los conjuntos de datos desequilibrados el clasificador tiende a clasificar bien la clase mayoritaria sobre la minoritaria si se utiliza una precisión general, pero *G-mean* utiliza una media geométrica que combina la precisión de la clasificación para ambas clases. Es por eso que se comporta significativamente bien ante conjuntos desbalanceados y es la medida más utilizada en este tipo de problemas. La forma de calcularla es mostrada a continuación:

$$G - mean = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{TN + FP}}$$

$$G - mean = \sqrt{PositiveAccuracy \cdot NegativeAccuracy}$$

- **AUC**: Otra metodología gráfica (y numérica) de evaluar cómo de bueno es un clasificador, es mediante la curva ROC (Receiver Operating Characteristic) [33]. Las curvas ROC para dos clases están basadas en una representación visual entre dos parámetros, la *sensibilidad* y la *especificidad*.

$$Sensibilidad = Recall\ clase\ positiva\ \frac{TP}{TP + FN}$$
$$Especificidad = \ Recall\ clase\ negativa\ \frac{TN}{TN+FP}$$

donde en el eje X se coloca *1-especificidad*, y en eje Y se coloca la *sensibilidad*. Esto proporciona un punto para cada clasificador o para cada medición. Una mayor exactitud en la clasificación se traduce en un desplazamiento hacia arriba y a la izquierda de la curva ROC. Basándose en esto, se propuso utilizar el área bajo la curva ROC *AUC* como índice conveniente de la exactitud global, de forma que, será mejor cuanto más cerca esté de 1 y empeorará conforme se acerca a 0.5 [34], tal y como se muestra en la Figura 4. *AUC* no se ve afectada por el desequilibrio atribuible a un conjunto, por este motivo es un buen medidor para evaluar la clasificación en conjunto de datos desbalanceados.

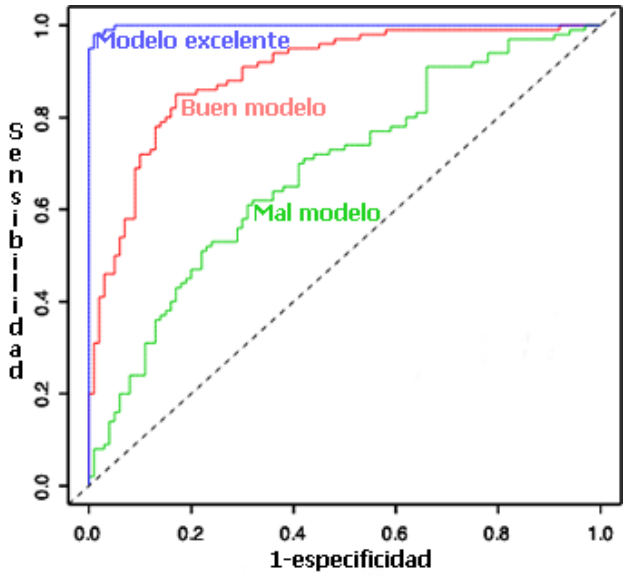

Figura 4: Ejemplo de curva ROC

10.4. Pseudocódigo

Para dar fin a esta sección realizaremos un resumen del algoritmo mostrando su pseudocódigo:

Algoritmo de over-sampling

Entrada*:*

Train inputs (Variables de entrada del conjunto de entrenamiento)
Train outputs (Etiqueta a predecir entrada del conjunto de entrenamiento)
Validation inputs (Variables de entrada del conjunto de validación)
Validation outputs (Etiqueta a predecir entrada del conjunto de validación)

Salida:

NewData inputs (Entradas del conjunto de sintéticos + *Train*)
NewData outputs (Etiquetas del conjunto de sintéticos + *Train*)
Árbol entrenado (Opcional) (Entrenado durante la validación)

1. **Para** Cada i perteneciente a m ($i \in m$)
2. | Calcular su ratio del cercanía al borde (número de vecinos mayoritarios / *k*)
3. | Normalizar su ratio del cercanía al borde ($\hat{r}_i$)
4. **Fin para**
5. **Mientras** contador **sea menor que** 100
6. **Para** Cada i perteneciente a m ($i \in m$)
7. | Asignarle un numero de sintéticos a generar (g_i) en función de $\hat{r}_i$
8. | Redistribuir la asignación de sintéticos (g_i) para eliminar los decimales ($\bar{g}_i$)
9. **Fin para**
10. **Para** cada i con $\bar{g}_i$ diferente de 0
11. | Seleccionar $\bar{g}_i$ vecinos mediante un modelo probabilístico basado en ε_j ($j \in k$)
12. | Generar un sintético S_{ij} por cada $\bar{g}_i$ utilizando la interpolación de SMOTE.
13. **Fin para**
14. Unir sintéticos S y *Validation*
15. Realizar validación con la unión de S y *Validation*
16. **Si** validación **es mejor que** la mejor validación encontrada antes
17. | Almacenar la mejor validación
18. | Almacenar árbol entrenado en la validación
19. **Sino**
20. | Contador se incrementa en 1
21. **Fin si**
22. Destruir un 15% de S (S_b, sabiendo que $S_b \subseteq S$)
23. Unir *Train* y S_b
24. Recalcular $\hat{r}_i$ para todo m
25. **Repetir**
26. **Fin**

Capítulo 11:
Arquitectura del sistema

En este capítulo vamos a desarrollar una estructura modular y a representar las relaciones de control que existen entre los distintos componentes o módulos de nuestro sistema. En la Figura 5 se mostrará un diagrama de entidad relación para poder entender rápidamente y de forma gráfica la distribución modular del sistema.

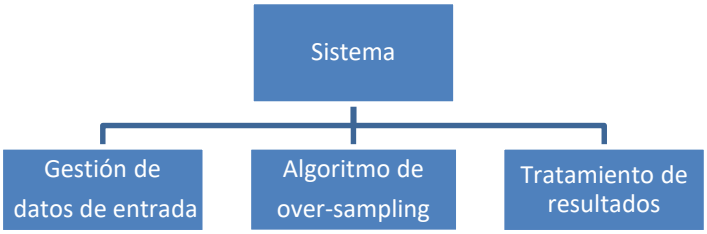

Figura 5: Diagrama de módulos del sistema

En los sucesivos apartados se desarrollarán los diferentes módulos que conforman el sistema, indicando que componentes están incluidos en cada uno y cómo han sido diseñados.

11.1. Gestión de datos de entrada

Este módulo es el encargado, no solo de leer los datos de entrada, sino también de realizar todo tratamiento necesario al conjunto para que el siguiente módulo pueda ejecutarse correctamente. Además, debe identificar e intentar corregir algunas de las incoherencias más importantes que no deben aparecer en ningún conjunto de datos. Este módulo cumple con las siguientes funcionalidades.

11.1.1. Definición de funciones del modulo

Función	Lectura de datos
Objetivo	Esta función debe ser capaz de leer correctamente la base de datos proporcionada por el usuario, identificar claramente las entradas y las salidas y separarlas en dos estructuras diferentes para que sea más fácil trabajar con ellas a lo largo de la ejecución del algoritmo.

Tabla 25: Función referente a la lectura de datos

Función	Conteo de clases
Objetivo	Este método identificaría el número de clases que existen en el *dasaset*, y el número de veces que aparece cada una de ellas, de esta forma se puede analizar el grado de desbalanceo que existe y proceder en consecuencia.

Tabla 26: Función referente al conteo e identificación de las clases

Función	Cambio de dimensión
Objetivo	El sistema estará diseñado para trabajar con problemas de solo dos clases. No obstante, es posible realizar una transformación de un problema de más clases en uno de tan solo dos de ellas, de forma que el sistema pueda ejecutarse correctamente y conseguir resultados satisfactorios. El sistema analiza el grado de presencia de cada clase en la totalidad del conjunto original, y de menor a mayor, todas aquellas clases con una representación acumulada menor que un determinado grado introducido por el usuario, serán incluidas dentro de la definición de clase minoritaria. Las clases restantes formaran la clase mayoritaria. Para configurar este parámetro correctamente es muy importante que el usuario conozca la distribución de las clases dentro del conjunto original. Sin embargo, si no se desea introducir ningún grado de corte, el sistema considerará como clase minoritaria solo la clase más minoritaria de todas.

Tabla 27: Función referente al cambio de dimensión

Función	Partición de los datos
Objetivo	El sistema debe segmentar tanto las variables de entrada como las de salida en 3 subconjuntos que llamaremos *Train*, *Test* y *Validation*, o en dos si no desea en conjunto de validación. La partición será estratificada, por lo que la distribución de las clases deberá seguir el mismo grado en cada uno de los conjuntos así como en el conjunto original. En esta partición no deben crearse incoherencias, y la etiqueta asociada a cada grupo de entradas debe mantenerse igual en todo momento.

Tabla 28: Función referente a la partición de los datos

Función	Índices de clase
Objetivo	La aplicación contará con un método para identificar de forma sencilla los índices de los parones de cada clase.

Tabla 29: Función referente a la identificación de índices de una clase

Función	búsqueda de duplicados
Objetivo	Esta función buscará los posibles duplicados que existan en el conjunto original y los eliminará de forma automática. Desde este documento se recomienda al usuario la eliminación de toda aquella duplicidad de patrones que pueda existir en el conjunto.

Tabla 30: Función referente a la identificación de duplicados

11.1.2. Análisis del módulo

A continuación, veremos un diagrama de secuencia que mostrará la interacción entre los componentes que forman el módulo de gestión de datos de entrada.

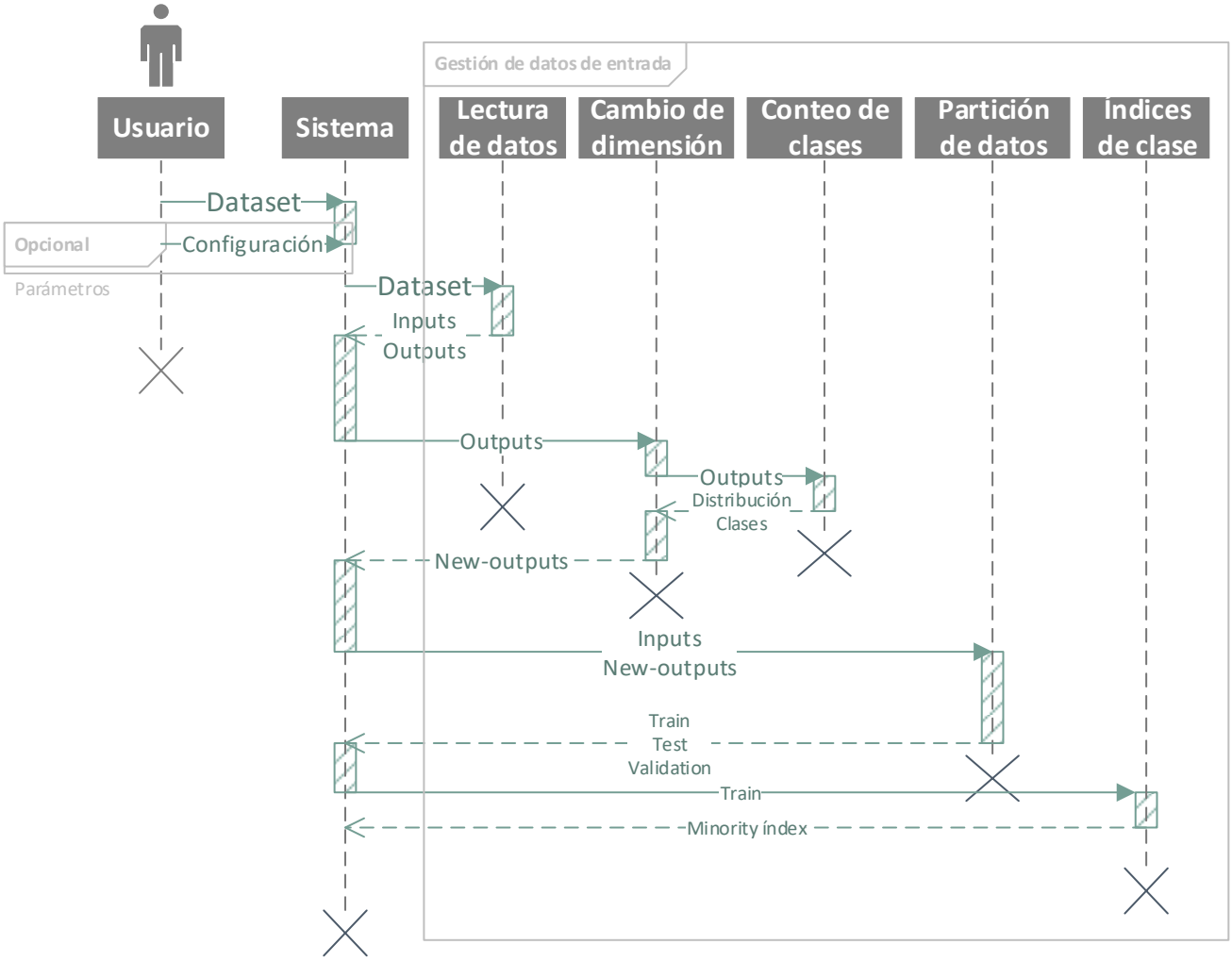

Figura 6: Diagrama de secuencia referente a la gestión de datos

11.2. Algoritmo de over-sampling

Este es el modulo más importante del sistema ya que es el que incluye el algoritmo de over-sampling diseñado y la metaheurística que lo optimiza. En él se recogen los datos preparados en el módulo anterior, se generan los patrones sintéticos utilizando todos los métodos necesarios, se unen los sintéticos al conjunto de *Train* para que quede balanceado, y se le pasan los conjunto de *Train* y *Test* al siguiente modulo para que sean evaluados. Las funcionalidades que se incluyen en este módulo se definen en las siguientes tablas.

Función	Ratio de cercanía al borde
Objetivo	Esta función calculará el ratio de cercanía al borde (vecinos clase mayoritaria / k vecinos) utilizando el supervisor de vecindad (Tabla 33) y normalizará dicho ratio con respecto a suma total del ratio de cada patrón. Este ratio normalizado simbolizará la importancia de cada patrón, y será utilizado posteriormente para asignarle un numero de sintéticos a generar.

Tabla 31: Función referente al cálculo de cercanía al borde

Función	Calculador de valor de *g*
Objetivo	Esta función calcula y asigna a cada patrón cuantos sintéticos debe generar (*g*), para ello utilizará el ratio de cercanía al borde normalizado y multiplicará cada valor por *G* (número total de sintéticos a generar). Posteriormente redistribuye aquellas asignaciones que posean una parte decimal.

Tabla 32: Función referente al cálculo de g

Función	Supervisor de vecindad
Objetivo	Esta función buscará la etiqueta de clase de los *k* vecinos más cercanos para cada patrón de un conjunto dado (Conjunto A). Si se facilita otro conjunto diferente en la llamada de la función (conjunto B), podrá calcularse los vecinos para cada patrón del conjunto A, con respecto a dicho conjunto B. Si no se establece un valor para *k,* se utilizará el método definido en la Tabla 34, para calcular automáticamente un valor.

Tabla 33: Función referente al supervisor de vecindad

Función	Calculador de valor de *k*
Objetivo	Esta función calculará un valor automáticamente para *k* (número de vecinos a tener en cuenta), de modo que todos los patrones de la clase minoritaria puedan tener al menos un vecino de la clase mayoritaria. De este modo, siempre podrá calcularse un ratio de cercanía al borde sin importar la densidad de patrones en la distribución, y el sistema tendrá la capacidad de adaptarse a cualquier base de datos sin necesidad de que el usuario introduzca un valor adecuado en la configuración.

Tabla 34: Función referente al cálculo de k

Función	Eliminación de ruido
Objetivo	Si se decide activarla, se eliminará todo aquél patrón de la clase minoritaria que tenga todos sus *k* vecinos pertenecientes a la clase mayoritaria, ya que serán considerados como ruido. Esto se realiza con el objetivo de perfeccionar más el borde y que quede mejor definido.

Tabla 35: Función referente a la eliminación del ruido

Función	Generador de sintéticos
Objetivo	Esta función encierra el núcleo más importante del sistema. En ella se generarán los sintéticos y se evolucionarán mediante la metaheurística voraz iterativa. Su diseño y funcionamiento ya ha sido explicado los apartados 10.1 y 10.2.

Tabla 36: Función referente a la generación de sintéticos

Función	Mejor segundo padre
Objetivo	Este método se encargará de elegir los mejores vecinos de cada patrón con el que generar los sintéticos que le corresponda, en lugar de utilizar un método puramente aleatorio como utilizan otros algoritmos explicados en el Capítulo 3: Antecedentes. Este método ha sido explicado más en detalle en el Capítulo 10:.

Tabla 37: Función referente a la elección del mejor vecino

Función	Validación
Objetivo	Se encarga de realizar el proceso de validación para guiar la evolución de la metaheurística. La validación puede guiarse en función del AUC o del G-mean.

Tabla 38: Función referente al proceso de validación

11.2.1. Análisis del módulo

A continuación, la Figura 7 y la Figura 8 muestran un diagrama de secuencia que especifica la interacción entre los componentes que forman el módulo de generación de patrones sintéticos.

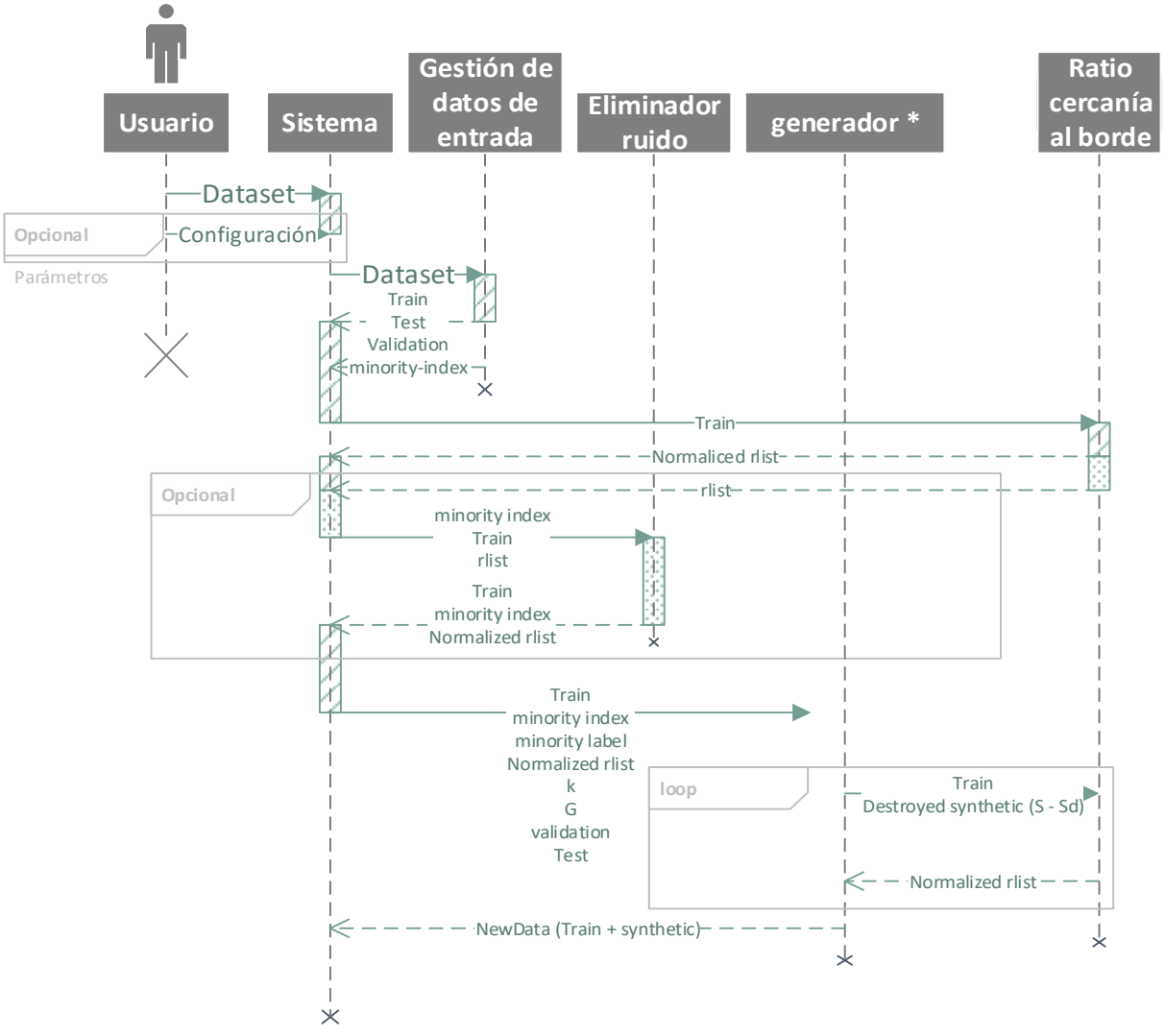

Figura 7: Diagrama de secuencia referente a la generación de sintéticos

La funcionalidad de generador * se especifica en la Figura 8.

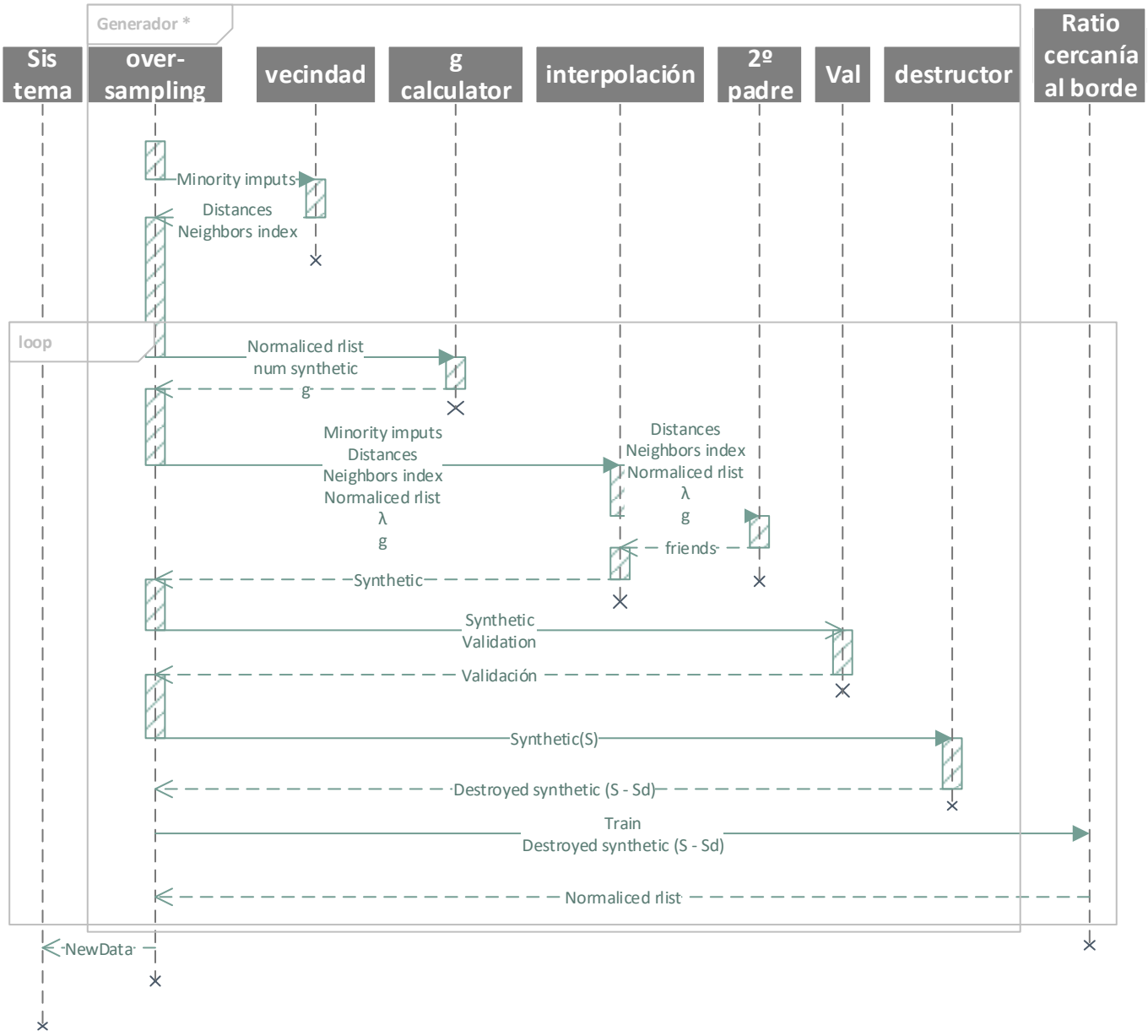

*Figura 8: Diagrama de secuencia correspondiente a la funcionalidad de generador * (complementa a la Figura 4).*

11.3. Tratamiento de resultados

Este módulo es el encargado tanto de calcular los resultados de la clasificación final, como de ordenarlos, mostrarlos y almacenarlos. Las funcionalidades que cumple este módulo se definen a continuación.

11.3.1. Definición de funciones del modulo

Función	Recolección de resultados
Objetivo	Esta función almacena y recopila en un conjunto de datos, todos los resultados obtenidos en las sucesivas iteraciones del algoritmo.

Tabla 39: Función referente a la recolección de resultados

Función	Evaluador
Objetivo	Es la función encargada de calcular algunos medidores de precisión de la clasificación.

Tabla 40: Función referente al cálculo de evaluadores

Función	Gestión de resultados
Objetivo	Es la función encargada de mostrar por pantalla la media de los resultados o de almacenarlos en un archivo.

Tabla 41: Función referente a la gestión de los resultados

Función	Representar gráfica
Objetivo	Es una función que puede representar una gráfica de varios evaluadores diferentes para el mismo conjunto, o imprimir el mismo evaluador para dos conjuntos diferentes, como por ejemplo *Validation* y *Test*, para ver si se produce sobre-entrenamiento.

Tabla 42: Función referente a la creación de gráficas

Función	Árbol de decisión
Objetivo	Es la función que contiene el árbol de decisión utilizado como clasificador. De este modo, tendremos centralizado el árbol que se utiliza en varios puntos como la validación o el test final, y solo requerirá de una configuración.

Tabla 43: Función referente a la creación de gráficas

11.3.2. Análisis del módulo

A continuación, veremos un diagrama de secuencia que mostrará la interacción entre los componentes que forman este módulo.

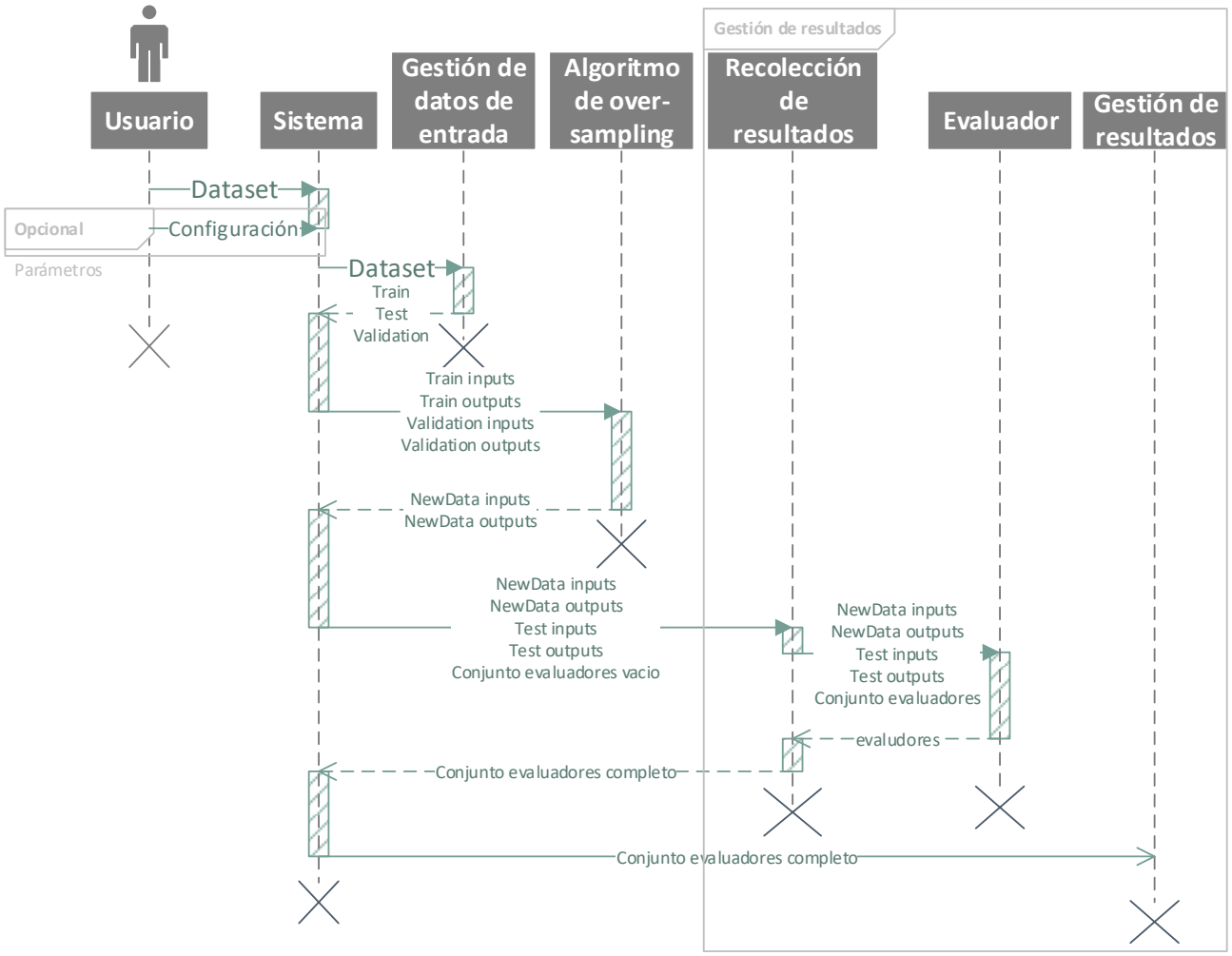

Figura 9: Diagrama de secuencia referente al tratamiento de resultados

11.4. Diagrama de módulos del sistema

Una vez desarrollados los módulos que conforman el sistema, la Figura 10 muestra de forma completa un diagrama de entidad relación que refleja la distribución modular.

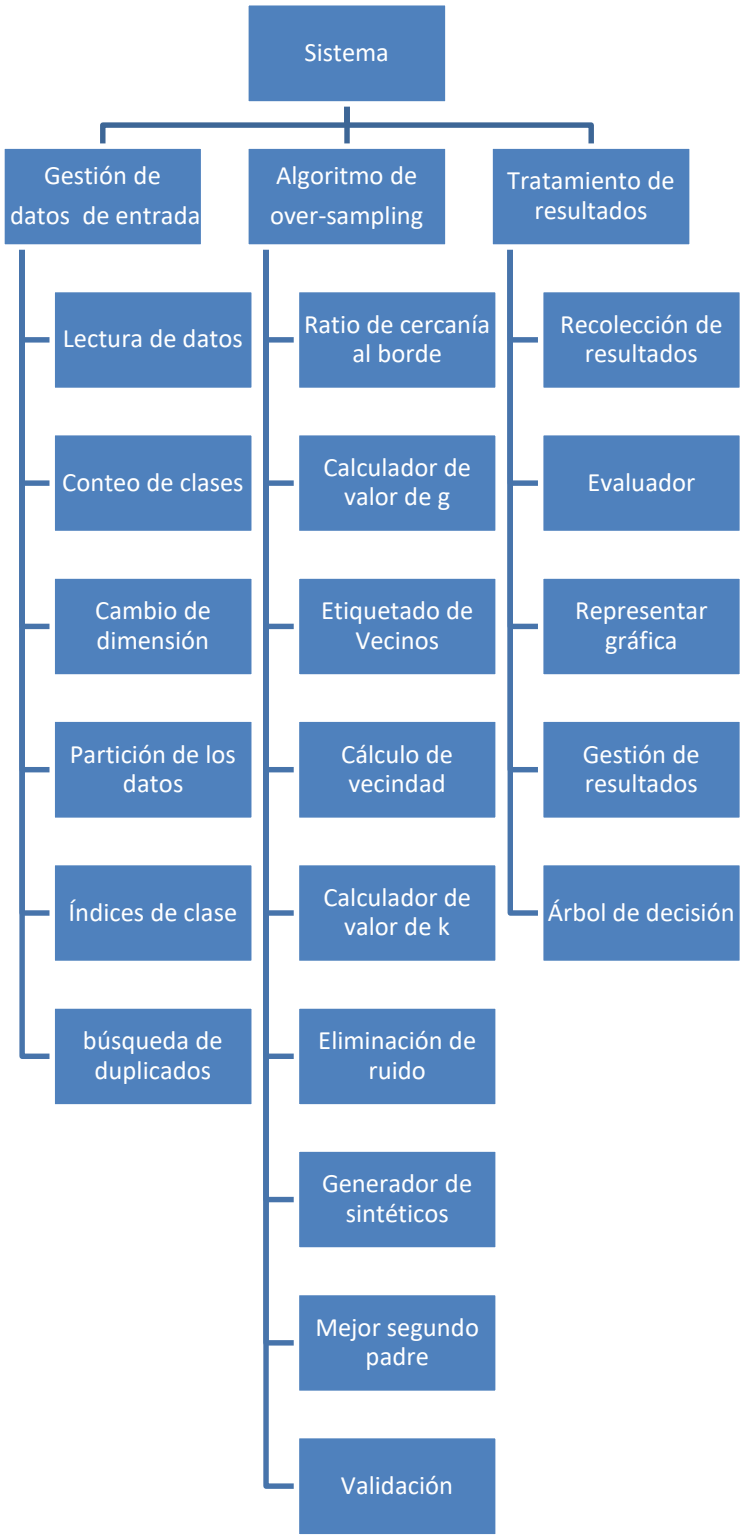

Figura 10: Diagrama de módulos del sistema completo

Parte IV
Pruebas

Capítulo 12:
Pruebas

La fase de pruebas es una parte fundamental y muy importante en el desarrollo de cualquier sistema, ya que permitirá evaluar la calidad de este. A lo largo del desarrollo de la aplicación se llevaran a cabo una serie de controles que determinaran su calidad. Además, permitirán detectar errores en las fases tempranas evitando que este problema subsista durante la evolución del sistema, ya que a medida que el desarrollo avanza, la dificultad de realizar una corrección puede llegar a crecer de forma exponencial tanto en tiempo como en esfuerzo.

Considerando un sistema como un conjunto de componentes que interactuaba entre sí para un fin común, es recomendable y necesario probar estos componentes por separado y tener la seguridad de que cada uno cumple su función correctamente y produce unos resultados satisfactorios antes de poder realizar pruebas a todo el conjunto.

A continuación, se va a realizar una breve introducción teórica sobre el fundamento y la finalidad de las pruebas. Para una explosión más profunda y detallada se puede consultar el libro Ingeniaría del Software de R. S. Pressman [35]. Posteriormente se explicará algunas de las pruebas realizadas al sistema.

12.1. Estrategia de pruebas

El objetivo es conseguir diseñar un conjunto de casos de prueba que lleve asociado un nivel de confianza lo suficientemente aceptable para poder localizar los errores existentes, y a la vez analizar si los resultados que se obtengan son aceptablemente buenos con respecto a un margen establecido y optimizarlo en caso contrario.

Como estrategia de pruebas para este sistema se utilizaran una serie de comprobaciones denominadas *pruebas de caja blanca* y *pruebas de caja negra*. Al aplicar estas pruebas se consigue detectar los errores, que implican volver sobre las fases anteriores para su resolución. Tras esto, de forma cíclica, se vuelven a realizar las pruebas, hasta obtener un resultado satisfactorio.

El procedimiento utilizado para eliminar errores es por tanto la *vuelta atrás,* localizando el síntoma que indujo a pensar que había un error y retroceder hasta llegar a la causa. A su vez, el conjunto de casos de prueba realizados se pueden clasificar en cuatro tipos diferentes, que serán explicados a continuación.

12.2. Pruebas unitarias

El conjunto de comprobaciones que forman esta parte, se corresponden con un conjunto de pruebas unitarias de los diferentes componentes del sistema, y por tanto, la verificación de la menor unidad del diseño del software, el módulo. Se conoce como módulo a un bloque o porción de código perteneciente al sistema que se encarga de realizar una tarea (o varias, en algún caso). En general, un módulo recibe unas entradas que pueden corresponderse con las entradas al sistema, o incluso con la salida que haya proporcionado otro módulo, entonces, tras realizar la función por la que ha sido diseñado, ofrece unas salidas que serán gestionadas de la forma que sea necesaria. En la programación, los módulos suelen estar organizados jerárquicamente (aunque no necesariamente), de manera que un módulo principal se encarga de realizar llamadas a módulos de nivel inferior, y estos a su vez pueden llegar a hacer lo mismo dependiendo de la magnitud del sistema. Este conjunto de pruebas se compone de dos tipos de ellas, como ya se ha mencionado anteriormente, las pruebas estructurales o *pruebas de caja blanca* (12.2.1. Pruebas de caja blanca) o *pruebas de caja negra* (12.2.2 Pruebas de caja negra).

12.2.1. Pruebas de caja blanca

Las pruebas de caja blanca están relacionadas con la fase de diseño. Estas intentan mostrar la validación de los componentes que conforman el sistema, por lo que se precisa probar la funcionalidad de cada uno de los componentes por separado y demostrar que se ejecutan de forma adecuada y no provocan ningún error. Resumiendo, consisten en probar cada una de las líneas de código que forman el sistema. Aunque normalmente las distintas combinaciones que se podrían llevar a cabo son muy numerosas, debido a la existencia de estructuras de control como bucles, sentencias condicionales, etc. Sin embargo, es necesario probar que cada una de esas sentencias realiza una acción adecuada a su objetivo, es decir, será necesario realizar una cobertura de sentencias que demuestre que cada trozo del código se ejecuta de forma correcta.

Los resultados de estas pruebas han llevado a la detección de errores solucionados en la fase de codificación, buscando la causa a partir de los síntomas que delataban al error. Así pues, estas pruebas se han llevado a cabo durante la etapa de codificación durante el desarrollo de los diferentes módulos existentes, buscando posibles errores realizando las acciones oportunas para depurarlos.

De esta manera, a medida que se iban implementando las distintas rutinas de ejecución, se iba comprobando su eficiencia y estructura para asegurar que no se producía ningún tipo de error.

Las pruebas mencionadas se han realizado de la siguiente manera:

1. Se ha comprobado que en cada módulo, se almacenaba y se trataba correctamente toda la información contenida en las variables y estructuras de datos.

2. Se ha comprobado que el flujo de información se realiza correctamente.

3. Se han realizado pruebas en los límites de los bucles.

Se ha comprobado que la ejecución de cada una de las sentencias que forman el módulo cumple su cometido.

Un ejemplo de este tipo de prueba podría ser la lectura de los argumentos de ejecución del mismo script que engloba al sistema. Inicialmente el *script* se ejecutaría de la siguiente forma:

./ANEIGSYN.py RutaBaseDeDatos Semilla

Y los argumentos se leen del siguiente modo:

```
fichero = sys.argv[1]
semilla = int(sys.argv[2])
```

Código 1: Lectura de argumentos

Pero cuando no se proporciona los argumentos necesarios, se obtiene un error al intentar almacenar dichos argumentos, ya que estos no han sido introducidos por el usuario. Se optó por la solución que se muestra en el Código 2.

```
ejecucion = "Para ejecutar: ./ 'nombre fichero' 'Ruta de una BBDD en .CSV'
'Semilla para aleatoriedad'"
    try:
        fichero = sys.argv[1]
    except IndexError:
        print "Falta la ruta de una BBDD\n", ejecucion
        sys.exit("Parando ejecución")

    try:
        semilla = int(sys.argv[2])
    except IndexError:
        semilla = np.random.randint(low=10000000, high=99999999)
        print 'semilla aleatoria:', semilla
```

Código 2: Lectura de argumentos supervisada

12.2.2. Pruebas de caja negra

Las de caja negra, por su parte, se centran en comprobar las salidas que ofrece el sistema a partir de unas entradas que se le han proporcionado, es decir, lo que se espera de un módulo. Estas intentan encontrar casos en los que el módulo no se atiene a su especiación. Por ello se denominan pruebas funcionales, limitándose a suministrar datos de entrada y estudiar su salida, sin analizar o estudiar lo que pueda estar haciendo el módulo de forma interna.

Las pruebas de caja negra se basan en la especificación de requisitos del módulo. De hecho, se habla de *cobertura de especificación* para dar una medida del número de requisitos que se han probado.

Estas pruebas se han realizado para cada módulo de la siguiente forma:

1. Se realizaron comprobaciones para determinar que los valores eran correctos en cuanto al tipo de dato.

2. Se analizaron los resultados obtenidos y en que situaciones se habían dado, y se comprobaba si había concordancia con lo esperado.

3. Para realizar dichas pruebas se utilizaron pequeños programas de prueba, en los que se cargaban los objetos de entraba para cubrir el caso de prueba a realizar, se ejecutaba el método y se imprimía el resultado por pantalla.

Con la estrategia seguida cabe destacar la prueba del algoritmo con distintas bases de datos y distintas configuraciones, comprobando que los datos sean correctos en todo momento. Además, se han probado todas las funciones relacionadas con la lectura y escritura de ficheros, analizando por ejemplo, que en el cambio de dimensión del problema (transformar a un problema bi-clase) se realice acorde a lo diseñado en el apartado 13.1.1. Bases de datos utilizadas, y se obtenga la distribución de las clases esperada.

12.3. Pruebas de integración

Para demostrar la integridad y robustez del sistema, es necesario realizar un conjunto de pruebas de integración que permita comprobar que el flujo de datos entre módulos es el adecuado. Para poder realizar todas estas pruebas es necesario contar con el sistema al completo, con todos los módulos integrados.

Para estas pruebas se ha seguido una metodología ascendente, es decir, que se trata de un proceso incremental en el cual se realizan pruebas a medida que se añade un módulo al grupo de módulos ya evaluados, de esta forma se verifica la integración modulo a modulo hasta formar el sistema al completo.

Así estas pruebas se han realizado de la siguiente forma:

1º. Se ha comprobado en cada paso incremental que la integración entre el nuevo módulo añadido con el conjunto de módulos ya evaluados funcionan correctamente.

2º. Se ha comprobado que el flujo de información entre los módulos se realiza correctamente.

3º. Se ha comprobado que la ejecución de todas las ramas posibles del conjunto de módulos analizados hasta ese momento sea correcta.

En esta fase se prestó especial interés sobre las entradas y las salidas de cada función, comprobando que todas ellas siguieran unos valores lógicos para su cometido. También se ha

asegurado que los datos no fueran modificados de forma indeseada tras su paso de un módulo a otro, y que todas las modificaciones que necesite realizar un módulo determinado no afecten de ninguna forma al exterior del mismo.

En los siguientes párrafos, se expondrá un error que se identificó durante estas pruebas a modo de ejemplo para entender su importancia, y no conformarse con que cada módulo funcione correctamente por separado. Como se ha explicado en 10.2. Aplicación de Iterated Greedy, los patrones sintéticos son utilizados para cambiar el ratio de cercanía al borde de los patrones originales durante la evolución que provoca la metaheurística. Esto induce a que en la reconstrucción de la solución por parte del constructor de *Iterated Greedy* escoja patrones en lugares donde la densidad de patrones minoritarios aun es escasa, después incluso de haber generado la primera generación de sintéticos. Esta técnica provocó varios errores durante las pruebas sobre algunas bases de datos muy desbalanceadas, debido a que al generar muchos patrones sintéticos estos inundaban el espacio de representación de los patrones sintéticos originales, haciendo que la importancia derivada de su ratio de cercanía al borde (vecinos mayoritarios / *k* vecinos) sea 0 para todos ellos al no encontrar vecinos de la clase mayoritaria.

Una vez que la importancia de todos patrones original era 0, el sistema determinaba que ningún patrón original generaba más sintéticos y la evolución de la metaheurística se hacía imposible. Entonces, se descubrió que este problema era debido a que eran necesarios varios valores de *k* a lo largo de la ejecución del algoritmo. En la fase inicial, un valor pequeño de *k* podría funcionar bien, pero durante la evolución de la metaheurística puede ser necesario incrementar este valor si el número de patrones sintéticos generados era muy alto e inundaba el espacio de representación. Por todo esto, se optó por añadir la opción de que las funciones que calculaban la vecindad, integraran un método para calcular cuántos vecinos es necesario analizar en cada momento de forma automática, y sobre todo durante la evolución de la metaheurística.

Estas pruebas han permitido obtener un sistema completo que funciona de forma correcta y en el que se han solucionado numerosos errores gracias a las pruebas anteriores. Por esto, es posible comenzar con una serie de pruebas del software para validar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones adecuadas. Estas se denominan *pruebas de sistema* y se analizarán en la siguiente sección.

12.4. Pruebas del sistema

Esta fase de pruebas se realiza con el objetivo de verificar que el sistema software cumple con todos los requisitos. Dentro de esta fase pueden desarrollarse varios tipos distintos de pruebas en función de los objetivos. De esta forma se cumple con el doble objetivo de ejecutar a fondo el sistema, cumpliendo con las pruebas de sistema y de comprobar si efectivamente el algoritmo cumple con los objetivos anteriormente expuestos (8.1. Catálogo de objetivos del sistema)

12.5. Pruebas de aceptación

Este tipo de prueba es diferente al resto ya que la realiza el usuario final. Son básicamente pruebas funcionales, sobre el sistema completo, y buscan una cobertura de la especificación de requisitos. La peculiaridad de estas pruebas es que no se realizan durante el desarrollo del sistema, sino una vez pasadas todas las pruebas comentadas anteriormente, ya que de otra forma el sistema no estaría disponible para el usuario. Aun cuando el desarrollador ha llevado a cabo el proceso más cuidadoso de pruebas, la experiencia nos dice que aún quedan una serie de errores en el sistema que solo aparecen durante la utilización del usuario.

Para estas pruebas, el usuario se queda a solas con el sistema a evaluar y trata de encontrarle errores, de los que informaría al desarrollador en caso de dar con alguno. En esta parte el usuario es el único con la posibilidad de encontrar cualquier tipo de fallo que haya sido obviado en todo el proceso anterior o también de buscar una mayor comodidad o eficiencia en el sistema.

Hasta aquí se han comprobado todo tipo de errores, pero no se puede afirmar que el sistema esté libre de ellos hasta que el usuario final lo haya comprobado y esté satisfecho con el resultado final de la aplicación.

En este contexto, podríamos ver a los directores del proyecto como los usuarios finales del mismo, ya que ellos están en capacidad de opinar sobre el funcionamiento del sistema y ayudan a dar con nuevos errores, en el caso de que los hubiese.

Capítulo 13:
Resultados experimentales

Una vez concluidas las primeras pruebas, es necesario realizar una serie de comprobaciones experimentales a gran nivel que permitan:

- Conocer el rendimiento del algoritmo.
- Inspeccionar a fondo el sistema, con el objetivo de finalizar un completo proceso de pruebas.
- Analizar diferentes configuraciones de parámetros.
- Valorar la efectividad del estudio teórico explicado en el capítulo donde se describe el problema (Capítulo 7: Descripción general del problema).
- Comparar la eficacia del sistema con distintas bases de datos con el objetivo de comprobar que, en general, los resultados obtenidos con los algoritmos son satisfactorios y acordes a lo esperado.

En este capítulo se expondrán los experimentos finales realizados al sistema así como el entorno y las condiciones de ejecución.

13.1. Condiciones de los experimentos

Para llevar a cabo estos experimentos, es necesario establecer el diseño experimental de los mismos y utilizar un conjunto de bases de datos diferentes que permitan realizar y evaluar multitud de pruebas en diferentes ámbitos. En este apartado comentaremos cual han sido el dominio sobre el que se han ejecutado las pruebas, comenzando por describir las bases de datos utilizadas y posteriormente, comentar los parámetros configurables del sistema para así poder apreciar y calibrar la flexibilidad del sistema ante diferentes situaciones.

13.1.1. Bases de datos utilizadas

Se han utilizado varias bases de datos con formato CSV asociadas a problemas de clasificación para probar el rendimiento del sistema. Algunas de ellas poseen más de dos clases en el conjunto original, por lo que se necesita una recodificación de las mismas para transformarlas en un problema bi-clase. A continuación, se describirán las principales características de todas ellas. Adicionalmente, se comentará si ha sido necesario realizar alguna recodificación y en caso de haber sido necesaria, detallar como se ha llevado a cabo.

Nombre	Abalone
Número de instancias	4177
Numero de clases	29
Distribución de las clases	• Clase 1: 1 Instancia • Clase 2: 1 Instancia • Clase 3: 15 Instancias • Clase 4: 57 Instancias • Clase 5: 115 Instancias • Clase 6: 259 Instancias • Clase 7: 391 Instancias • Clase 8: 568 Instancias • Clase 9: 689 Instancias • Clase 10: 634 Instancias • Clase 11: 487 Instancias • Clase 12: 267 Instancias • Clase 13: 203 Instancias • Clase 14: 126 Instancias • Clase 15: 103 Instancias • Clase 16: 67 Instancias • Clase 17: 58 Instancias • Clase 18: 42 Instancias • Clase 19: 32 Instancias • Clase 20: 26 Instancias • Clase 21: 14 Instancias • Clase 22: 6 Instancias • Clase 23: 9 Instancias • Clase 24: 2 Instancias • Clase 25: 1 Instancia • Clase 26: 1 Instancia • Clase 27: 2 Instancias • Clase 29: 1 Instancia
Numero de atributos	8 + 1(clase)
Tipo de atributos	1 Atributo Nominal (suprimido en la modificación necesaria) 7 atributos continuos 1 atributo discreto (clase)
Valores perdidos	No
Descripción	Esta base de datos contiene información acerca de las características de un abalone (orejas de mar o haliotis) y de su entorno, con el objetivo de predecir su edad de una forma sencilla.
Requiere modificación	Si
Modificación	Se ha seleccionado la clase 18 como la clase minoritaria y la 9 como clase mayoritaria (según [3]). Además se ha suprimido el atributo nominal. Esto genera un conjunto de dos clases con una distribución desigual donde hay 42 patrones en la clase minoritaria y 689 en la mayoritaria y cada instancia posee 7 atributos.
Más información	https://archive.ics.uci.edu/ml/datasets/Abalone

Tabla 44: Descripción dataset Abalone

Nombre	Ionosphere
Número de instancias	351
Numero de clases	2
Distribución de las clases	• Clase 1 (b): 126 Instancias • Clase 2 (g): 225 Instancias
Numero de atributos	34 + 1(clase)
Tipo de atributos	34 atributos continuos 1 atributo nominal (clase)
Valores perdidos	No
Descripción	Este conjunto está formado con los datos recolectados de un radar en Goose Bay, Labrador. El valor "Good radar" era devuelto cuando había evidencias de electrones libres en la ionosfera, y "Bad radar" en caso contrario.
Requiere modificación	No
Modificación	-
Más información	https://archive.ics.uci.edu/ml/datasets/Ionosphere

Tabla 45: Descripción dataset Ionosphere

Nombre	Page-blocks
Número de instancias	5473
Numero de clases	5
Distribución de las clases	• Clase 1 (text): 4913 Instancias • Clase 2 (horiz. line): 329 Instancias • Clase 3 (graphic): 28: 28 Instancias • Clase 4 (vert. line): 88 Instancias • Clase 5 (picture): 115 Instancias
Numero de atributos	34 + 1(clase)
Tipo de atributos	4 atributos continuos 6 atributos enteros 1 atributo nominal (clase)
Valores perdidos	No
Descripción	El problema consiste en clasificar todos los bloques del diseño de una página de un documento que han sido detectados por un proceso de segmentación. Este es un paso esencial en el análisis de documentos para separar el texto de las áreas gráficas.
Requiere modificación	Si
Modificación	Se ha seleccionado "text" como la clase mayoritaria y la unión de las demás como clase minoritaria [36]. Esto genera un conjunto de dos clases con una distribución desigual donde hay 559 patrones en la clase minoritaria y 4913 en la mayoritaria.
Más información	https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification

Tabla 46: Descripción dataset Page-blocks

Nombre	Pima Indians Diabetes
Número de instancias	768
Numero de clases	2
Distribución de las clases	• Clase 1 (1): 268 Instancias • Clase 2 (0): 500 Instancias
Numero de atributos	8 + 1(clase)
Atributos	1. Número de veces que ha estado embarazada 2. Concentración de glucosa en plasma 3. Presión arterial diastólica (mm Hg) 4. Espesor del pliegue cutáneo del tríceps (mm) 5. Insulina sérica de 2 horas (mu U / ml) 6. Índice de masa corporal (IMC) 7. Diabetes pedigree function 8. Edad 9. Clase (0 o 1)
Valores perdidos	Si
Descripción	Esta base de datos contiene información médica sobre pacientes femeninos con al menos 21 años de edad y con antecedentes familiares de diabetes. Se busca predecir si los pacientes presentan diabetes o no lo presentan.
Requiere modificación	No
Modificación	-
Más información	https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

Tabla 47: Descripción dataset Pima Indians Diabetes

Nombre	Vehicle
Número de instancias	846
Numero de clases	4
Distribución de las clases	• Clase 1 (opel): 212 Instancias • Clase 2 (saab): 217 Instancias • Clase 3 (bus): 218 Instancias • Clase 4 (van): 199 Instancias
Numero de atributos	18 + 1(clase)
Tipo de atributos	18 atributos continuos 1 atributo nominal (clase)
Valores perdidos	No
Descripción	Esta base de datos contiene información extraída de imágenes en dos dimensiones sobre cuatro tipos diferentes de vehículos. Se pretende predecir, dadas las características de una imagen, a qué tipo de vehículo pertenece.
Requiere modificación	SI
Modificación	Se ha seleccionado “Van” como la clase minoritaria y la unión de las demás como clase mayoritaria (como en [3]). Esto genera un conjunto de dos clases con una distribución desigual donde hay 199 patrones en la clase minoritaria y 647 en la mayoritaria.
Más información	https://archive.ics.uci.edu/ml/datasets/Statlog+(Vehicle+Silhouettes)

Tabla 48: Descripción dataset Vehicle

Nombre	Vowel
Número de instancias	990
Numero de clases	11
Distribución de las clases	• Clase 1 (1): 90 Instancias • Clase 2 (0): 90 Instancias • Clase 3 (1): 90 Instancias • Clase 4 (0): 90 Instancias • Clase 5 (1): 90 Instancias • Clase 6 (0): 90 Instancias • Clase 7 (1): 90 Instancias • Clase 8 (0): 90 Instancias • Clase 9 (1): 90 Instancias • Clase 10 (0): 90 Instancias • Clase 11 (1): 90 Instancias
Numero de atributos	13 + 1(clase)
Tipo de atributos	3 Atributos discretos (suprimido en la modificación necesaria) 10 atributos continuos 1 atributo nominal (clase)
Valores perdidos	No
Descripción	Esta base de datos contiene información extraída del análisis de señales de audio, donde diferentes personas leen un texto. Este texto consiste en palabras donde se destacan 11 pronunciaciones diferentes de vocales inglesas (del idioma ingles). La predicción que se busca en este ejemplo consiste en determinar qué vocal pronuncia un interlocutor dado el análisis de la señal de audio.
Requiere modificación	Si
Modificación	Se ha seleccionado la primera vocal como la clase minoritaria y la unión de las demás como clase mayoritaria. Además se han suprimido todos los atributos discretos (según [3]). Esto genera un conjunto de dos clases con una distribución desigual donde hay 90 patrones en la clase minoritaria y 900 en la mayoritaria y cada instancia posee 10 atributos.
Más información	http://sci2s.ugr.es/keel/dataset.php?cod=113

Tabla 49: Descripción dataset Vowel

Nombre	Numero de instancias	Instancias minoritarias	Instancias mayoritarias	Atributos
Abalone	731	42	689	7
Ionosphere	351	126	225	34
Page-blocks	5472	559	4913	10
Pima Indians Diabetes	768	268	500	8
Vowel	990	90	900	10
Vehicle	846	199	647	18

Tabla 50: Resumen datasets utilizados

13.1.2. Parámetros de configuración

Como se ha explicado a lo largo de este documento, el sistema cuenta con múltiples parámetros configurables que pueden modificar el rendimiento y funcionamiento del algoritmo. Por ello, es necesario ajustar lo mejor posible dichos parámetros para obtener unos resultados adecuados. En nuestro caso, prácticamente todos los parámetros han sido estimados mediante ensayos de prueba y error, donde algunos de ellos partían de una base inspirada en configuraciones extraídas de otros documentos que describían otros algoritmos de *over-sampling* similares.

Los parámetros configurables del sistema son muy diversos, y en líneas generales, podríamos separarlos en dos grandes grupos, parámetros básicos y parámetros avanzados. Los parámetros básicos son aquellos que modifican las principales características del sistema dando la posibilidad de adecuarse a las diversas características de la mayoría de los conjuntos de datos. Son los más sencillos de configurar y se ha facilitado su acceso agrupándolos en una función de configuración dentro del sistema. Por otro lado, los parámetros avanzados están relacionados con aspectos de funcionamiento más específicos y modifican el comportamiento del algoritmo a niveles más bajos que los parámetros básicos. Para modificar este tipo de parámetros será necesario entender la codificación del sistema y configurar dichos parámetros en las partes donde se considere oportuno.

En la Tabla 51, se describen los parámetros básicos y su posibilidad de configuración.

Nombre	Descripción	Intervalo de valores	Valor por defecto
test_size	Tamaño porcentual del conjunto de *Test.*	$(0, 1)$	0.5
train_size	Tamaño porcentual del conjunto de *Train.*	$(0, 1)$	0.3
val_size	Tamaño porcentual del conjunto de *Validation.*	$[0, 1)$	1 – (test_size + train_size)
runs	Número de veces que se ejecutará el algoritmo.	$[2, \infty)$	100
beta	Especifica el nivel de balanceo que será cubierto después de la generación de sintéticos.	$(0, 1]$	1
k	Número de vecinos a tener en cuenta para detectar e borde. Si es None el sistema será capaz de calcular de forma automática cuantos vecinos es necesario tener en cuenta según la distribución de cada base de datos.	$[1, \infty)$, None	None
n	Factor de ponderación entre $\hat{f}_{j1}$ y $\hat{f}_{j2}$.	$(0, 1)$	0.5
val_auc	Si es True, la validación se realiza en función del área bajo la curva ROC.	[True, False]	False
pintar_graficas	Si es True, sacará una gráfica en cada ejecución.	[True, False]	False
remove_noise	Si es True, se activa la función de eliminación de ruido.	[True, False]	False
imbalanced_degree	Variable que se utilizará para especificar la clase minoritaria en tiempo de ejecución. Toda clase que obstente una presencia menor que *imbalanced_degree* será considerada minoritaria y se unirán en una sola clase. Las clases restantes formaran la clase mayoritaria. Si no se especifica se considerara como clase minoritaria solo la más minoritaria de todas.	$(0, 1]$	None

Tabla 51: Parámetros de configuración básicos

A continuación comentaremos cuales son los parámetros avanzados y como se pueden modificar. Se ha optado por mencionar las funciones que han sido implementadas con cierta flexibilidad, y en las que se da la posibilidad de configurar ciertos aspectos de su funcionamiento. Como se ha dicho anteriormente, para configurar estos parámetros será necesario localizar las llamadas a estas funciones en el código de *Python* y modificar los parámetros oportunos para alcanzar el efecto deseado. Así pues, en las siguientes tablas se incluirá una breve descripción de cada función, y cuáles son los parámetros de las mismas que permiten cierta configuración especial.

Función		ratio_borde
Calcula el ratio de cercanía al borde llamado *rlist* (vecinos clase mayoritaria / k vecinos) y lo normaliza con respecto a su suma (rlist normalizado).		
k	**Intervalo de valores**	$[0, \infty)$
	Valor por defecto	None
	Objetivo	Establecer el número de vecinos que se analizaran para calcular el ratio de cercanía al borde de cada uno de los patrones de la clase minoritaria. Si no se pasa como parámetro, se calculará automáticamente adaptándose al conjunto que se esté ejecutando en ese momento.
return_rlist	**Intervalo de valores**	True - False
	Valor por defecto	False
	Objetivo	Si es True, la función *ratio_borde* devolverá el vector *rlist* además de *rlist* normalizado

Tabla 52: Configuración función ratio_borde

Función		neighbors_calculator
Calcula la etiqueta de clase de los *k* vecinos más cercanos		
k	**Intervalo de valores**	$[0, \infty)$
	Valor por defecto	None
	Objetivo	Establecer el número de vecinos que se analizaran para calcular el ratio de cercanía al borde de cada uno de los patrones de la clase minoritaria. Si no se pasa como parámetro, se calculará automáticamente adaptándose al conjunto que se esté ejecutando en ese momento.
self_neighbour	**Intervalo de valores**	True - False
	Valor por defecto	False
	Objetivo	Si es True, se considerará que un patrón es vecino de sí mismo

Tabla 53: Configuración función neighbors_calculator

Función		generate_synthetics
Esta es la función encargada de generar los patrones sintéticos, además de evolucionar el conjunto utilizando la metaheurística voraz iterativa. Devolverá el mejor conjunto conseguido una vez se cumpla la condición de parada de la metaheurística.		
return_clf	**Intervalo de valores**	True - False
	Valor por defecto	True
	Objetivo	Si es True, la función también devolverá el árbol entrenado con el que se consiguió el mejor conjunto.

Tabla 54: Configuración función generate_synthetics

Función		neighbors_index
Calcula los índices, y si es necesario las distancias, de los k vecinos más cercanos entre dos conjuntos.		
return_distances	**Intervalo de valores**	True - False
	Valor por defecto	False
	Objetivo	Si es True, se devolverá las distancias entre los patrones además de sus índices.
self_neighbour	**Intervalo de valores**	True - False
	Valor por defecto	False
	Objetivo	Si es True, se considerará que un patrón es vecino de sí mismo

Tabla 55: Configuración función neighbors_index

Función		validation
Calcula el G_mean sobre el conjunto de validación para guiar a la metaheurística. SI se desea, puede utilizarse el AUC en lugar de G_mean.		
clf	**Intervalo de valores**	Clf - None
	Valor por defecto	None
	Objetivo	Da la posibilidad de pasar un árbol ya entrenado para realizar la validación. Si no se proporciona, se creará y entrenará un árbol durante la ejecución de la función
return_clf	**Intervalo de valores**	True - False
	Valor por defecto	True
	Objetivo	Si es True, la función también devolverá el árbol entrenado que se ha conseguido durante la validación.

Tabla 56: Configuración función validation

Función		tester
Almacena a lo largo de una estructura de datos las medidas de precisión calculadas en la función *evaluator (*Tabla 58*),* durante las diferentes ejecuciones del algoritmo.		
clf	**Intervalo de valores**	Clf - None
	Valor por defecto	None
	Objetivo	Da la posibilidad de facilitar un árbol ya entrenado para que sea pasado a la función *evaluator*, encargada de realizar el test.

Tabla 57: Configuración función tester

Función		evaluator
Calcula una serie de medidas de precisión para evaluar la clasificación.		
clf	**Intervalo de valores**	Clf - None
	Valor por defecto	None
	Objetivo	Da la posibilidad de facilitar un árbol ya entrenado que será utilizado para realizar el test de la clasificación. Si no se proporciona, se creará y entrenará un árbol durante la ejecución de la función.

Tabla 58: Configuración función evaluator

Función		imprimir_resultado
Imprime en pantalla o sobre un fichero la media de las medidas de precisión alcanzadas en cada ejecución.		
aux_dates1	**Intervalo de valores**	ndarray - None
	Valor por defecto	None
	Objetivo	Da la posibilidad de pasar un segundo conjunto de medidas de precisión, para poder realizar una comparación más fácilmente.
aux_dates2	**Intervalo de valores**	ndarray - None
	Valor por defecto	None
	Objetivo	Da la posibilidad de pasar un tercer conjunto de medidas de precisión, para poder realizar una comparación más fácilmente.
aux_name1	**Intervalo de valores**	String - None
	Valor por defecto	'Second_column'
	Objetivo	El nombre que se le asignará a la columna de *aux_dates1*, si es que se utiliza.
aux_name2	**Intervalo de valores**	String - None
	Valor por defecto	'Aux_column'
	Objetivo	El nombre que se le asignará a la columna de *aux_date2*, si es que se utiliza.
to_screen	**Intervalo de valores**	True - False
	Valor por defecto	True
	Objetivo	Imprime los resultados por pantalla
to_csv	**Intervalo de valores**	True - False
	Valor por defecto	True
	Objetivo	Vuelca los resultados en un archivo con formato CSV
file_name	**Intervalo de valores**	String - None
	Valor por defecto	Results
	Objetivo	Nombre del archivo CSV donde se guardaran los resultados

Tabla 59: Configuración función imprimir resultado

13.2. Resultados de las pruebas

Como ya se dijo en el Capítulo 4: Objetivos, se pretenden comparar varios algoritmos:

- **Adaptive Synthetic Sampling** (ADASYN)
- **Adaptive neighbors Synthetic Sampling** (ANESYN)
- **Adaptive neighbors Iterated Greedy Synthetic Sampling** (ANEIGSYN)

Para la ejecución del algoritmo principal (ANEIGSYN) se ha dividido cada base de datos original en 3 subconjuntos estratificados, *Train*, *Test* y *Validation*. Esto es necesario debido a que el algoritmo principal utiliza una metaheurística como es *Iterated Greedy* para evolucionar un conjunto de sintéticos hacía otro que aporte una información más útil al clasificador. Para ello, es necesario utilizar el conjunto *Validation,* y comprobar si dicha evolución se realiza en la dirección correcta (sin provocar sobreaprendizaje), y asegurar de algún modo la convergencia hacía un mejor conjunto de sintéticos.

En los métodos que no utilizan un proceso de optimización mediante una metaheurística, solo es necesario dividir el conjunto original en *Train, Test.* El conjunto de *Test* debe ser el mismo en la comparación de diferentes métodos siempre que se utilice la misma base de datos en dicha comparación, para que de esta forma se pueda llegar a una conclusión más fiable sobre los resultados. Por lo que el conjunto de *Train* en este tipo de métodos sería equivalente a la unión de los conjuntos de *Train* y *Validation*, disponiendo de este modo de un conjunto de entrenamiento significativamente mayor. Dependiendo de la distribución original de la base de datos, estos métodos pueden beneficiarse en gran medida de este incremento del conjunto de *Train*, y conseguir mejores resultados en comparación con los valores iniciales de un método que cuente con una optimización mediante metaheurística. Desde este punto de vista, el algoritmo con metaheurística debería aprovechar el conjunto de *Validation* y optimizar los resultados aun cuando se utiliza un conjunto de *Train* más reducido.

En primer lugar analizaremos los valores del algoritmo principal o ANEIGSYN de forma individual, evaluando si las pruebas realizadas se consideran satisfactorias antes de pasar a la comparación con los demás algoritmos.

13.2.1. Evaluación del algoritmo ANEIGSYN

Comenzaremos por realizar unas pruebas que intentan evaluar si el método diseñado es capaz de generar un conjunto de sintéticos de calidad, si estos son unos buenos representantes de la clase minoritaria y si poseen una información útil aprovechable por el clasificador.

Evaluación de calidad del conjunto de sintéticos

En estas pruebas se pretende evaluar la calidad de la información atribuible al conjunto de patrones sintéticos. Dicho conjunto está generado a partir de la clase minoritaria, y por tanto cada sintético debería representar potencialmente un patrón de la clase minoritaria.

Para ver si los patrones que se generan tienen semejanzas con los originales y si son creados en una zona "perteneciente" a la clase minoritaria, se ha optado por generar un conjunto de sintéticos con el mismo tamaño que la clase minoritaria original y sustituir la clase minoritaria por dicho conjunto de sintéticos, es decir, crear un nuevo conjunto de datos formado por la clase mayoritaria y el conjunto de sintéticos como representante de la clase minoritaria.

De este modo, se obtienen dos conjuntos diferentes pero con el mismo tamaño y el mismo grado de desbalanceo. Uno de ellos sería el conjunto original, y el otro se corresponde con un conjunto muy similar al primero con la única diferencia de que la clase minoritaria ha sido eliminada y ahora está representada por patrones sintéticos en lugar de los originales. Si evaluamos ambos conjuntos, entrenamos un árbol de decisión y analizamos su resultado, podremos ver si el clasificador es capaz de distinguir entre los sintéticos que representan a la clase minoritaria y la clase mayoritaria, y si se alcanzan resultados similares a los que se obtienen con el conjunto original.

Cabe destacar que los conjuntos evaluados en estas pruebas aún siguen desbalanceados, ya que en este caso, los sintéticos no son utilizados para balancear el conjunto, sino que se está evaluando si los patrones sintéticos que se generan en el algoritmo realmente ostentan una buena representación de la clase minoritaria. La Tabla 60 recoge el valor de G-mean alcanzado en estas pruebas para todas las bases de datos utilizadas.

Nombre BBDD	**Abalone**		**Ionosphere**	
Tipo de conjunto	**Con sintéticos**	**Original**	**Con sintéticos**	**Original**
OA	0,865154	0,920239	0,849429	0,858
Precision_Class1	0,964345	0,956584	0,853597	0,878799
Precision_Class0	0,215103	0,307333	0,849462	0,823743
Recall_Class1	0,889855	0,958949	0,927111	0,905667
Recall_Class0	0,464118	0,291765	0,7096	0,7722
F1_measure_Class1	0,925229	0,957669	0,887906	0,891248
F1_measure_Class0	0,288212	0,289975	0,768915	0,794345
G-mean	0,633868	0,510315	0,808876	0,834903
AUC	0,676986	0,625357	0,818356	0,838933

Tabla 60: Sustitución de originales por sintéticos 1

Nombre	Page-blocks		Pima diabetes	
Tipo de conjunto	Con sintéticos	Original	Con sintéticos	Original
OA	0,939849	0,962458	0,686104	0.689351
Precision_Class1	0,982465	0,977899	0,73204	0.765394
Precision_Class0	0,661305	0,824549	0,563353	0.553508
Recall_Class1	0,949964	0,980351	0,820697	0.757065
Recall_Class0	0,851116	0,805491	0,433271	0.562150
F1_measure_Class1	0,965921	0,979115	0,773233	0.760474
F1_measure_Class0	0,743656	0,814374	0,486951	0.555944
G-mean	0,898975	0,88842	0,59353	0.650720
AUC	0,90054	0,892921	0,626984	0.659607

Tabla 61: Sustitución de originales por sintéticos 2

Nombre	Vehicle		Vowel	
Tipo de conjunto	Con sintéticos	Original	Con sintéticos	Original
OA	0,902242	0,910147	0,9719191	0,9724444
Precision_Class1	0,920884	0,939978	0,9910275	0,9847099
Precision_Class0	0,835028	0,816847	0,8114036	0,8549965
Recall_Class1	0,954595	0,94305	0,978000	0,9850444
Recall_Class0	0,73275	0,803625	0,9111111	0,8464444
F1_measure_Class1	0,937242	0,941277	0,9844354	0,9848430
F1_measure_Class0	0,778133	0,808065	0,8560946	0,8478621
G-mean	0,835092	0,869722	0,9435309	0,9121187
AUC	0,843672	0,873338	0,9445555	0,9157444

Tabla 62: Sustitución de originales por sintéticos 3

Como se aprecia, los resultados obtenidos con ambos conjuntos son muy similares, sin observar diferencias significativas. Consideramos, por tanto, que el clasificador es capaz de distinguir de una forma aceptablemente buena entre los patrones sintéticos y la clase mayoritaria, comparable o incluso mejor en algunos casos, a cómo lo hace entre las clases originales.

Los resultados de estas pruebas no solo nos han servido para apreciar que el algoritmo es capaz de generar un conjunto de sintéticos portadores de información útil que un clasificador es capaz de aprender, sino que también será utilizada como referencia de los valores que se alcanzan con el conjunto original sin balancear para su posterior comparación con los valores que se alcancen tras la ejecución del algoritmo.

Evolución de la metaheurística

En estas pruebas se desea comprobar si la metaheurística consigue una evolución satisfactoria del conjunto de sintéticos. Como ya se ha comentado, el algoritmo se ejecuta varias veces para poder obtener un conjunto de resultados a los que poder hacer la media, y de esta forma desligar el resultado final de la influencia que las cuestiones de aleatoriedad puedan causar en una ejecución individual.

Tanto el Gráfico 1 como el Gráfico 2, hacen referencia a unas ejecuciones extraídas de un total de 100 utilizando la base de datos Abalone y la semilla "65231478". En ellos se intenta mostrar cómo puede influir la evolución del conjunto de validación, sobre los resultados conseguidos en el conjunto de test.

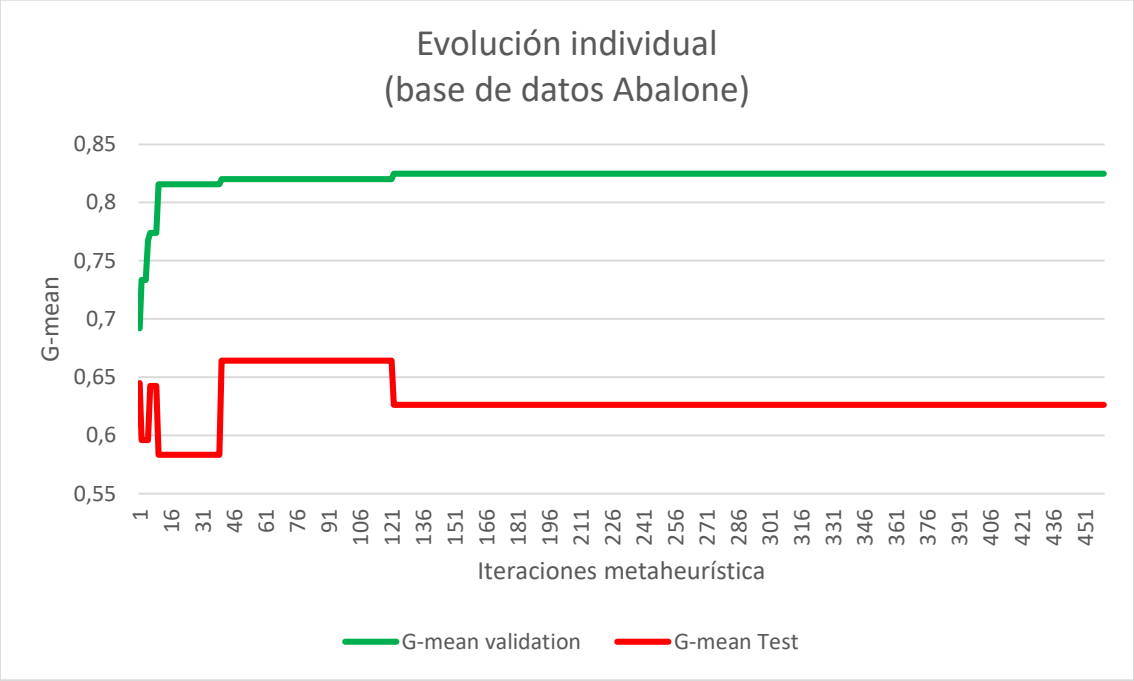

Gráfico 1: Ejecución individual 1

Como observamos, el valor de G-mean para el conjunto *Validation* siempre aumenta a medida que la ejecución se desarrolla dado que es el conjunto que se utiliza como referencia para guiar la evolución de la metaheurística. En cambio, el valor de G-mean en *Test* sufre variaciones pero no siempre se corresponden con un aumento de su G-mean.

Este fenómeno es comúnmente apreciable en este tipo de algoritmos, y se debe a que un cambio durante la optimización de la metaheurística que provoque una mejora en los resultados del conjunto de *Validation* no tiene por qué implicar una mejora sistemática en el conjunto de Test, debido a que estos conjuntos son diferentes y cada uno presenta unas características únicas. Aun así, la tendencia del conjunto de *Test* debe ser mejorar sus resultados y acompañar al conjunto de Validation en su evolución en las sucesivas ejecuciones del algoritmo.

A modo de ejemplo mostraremos una ejecución individual más para la misma base de datos utilizada en el Gráfico 1.

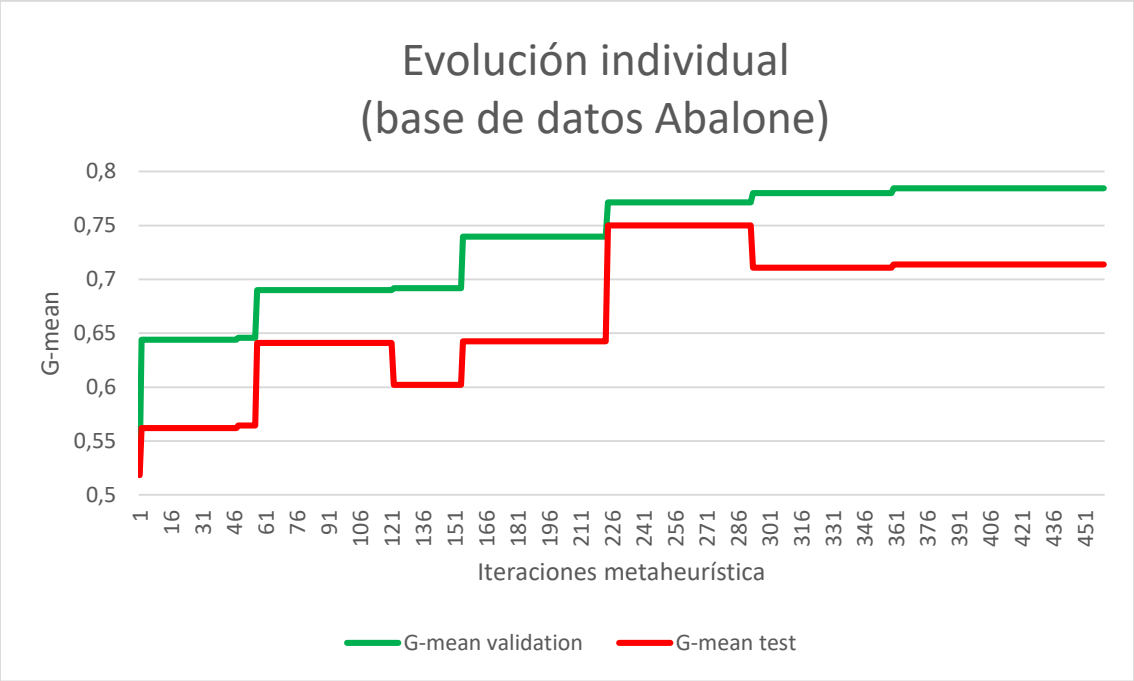

Gráfico 2: Ejecución individual 2

En esta ocasión, se puede apreciar claramente cómo, aun sufriendo algunas variaciones, los valores de *Test* acompañan a la evolución de *Validation* y consiguen un valor mucho mejor que el inicial.

Estás gráficas donde se muestran una única ejecución individual son muy importantes para conseguir una primera valoración sobre el funcionamiento del algoritmo, y para analizar si se están dando algunos eventos, como podría ser sobrentrenamiento. Sin embargo, si queremos analizar la tendencia general del algoritmo necesitamos otro tipo de prueba. Es por esto por lo que se ha diseñado la siguiente comprobación.

1º. En primer lugar, recabaremos los datos de 100 ejecuciones diferentes para cada base de datos.

2º. Dado que en cada ejecución el número de iteraciones de la metaheurística puede llegar a ser distinto, estos datos necesitan un tratamiento. Debemos replicar el último valor alcanzado tanto de validación como de test en aquellas ejecuciones más cortas hasta alcanzar a la ejecución más larga dentro de las 100 ejecuciones de su misma base de datos.

3º. Una vez conseguidos los valores de las 100 ejecuciones y que todas ellas posean el mismo número de valores, realizaremos una media aritmética sobre los datos.

4º. Finalmente analizaremos la media obtenida en el paso anterior utilizando, por ejemplo, un gráfico.

Seguidamente se mostraran los gráficos de la media de 100 ejecuciones para cada base de datos.

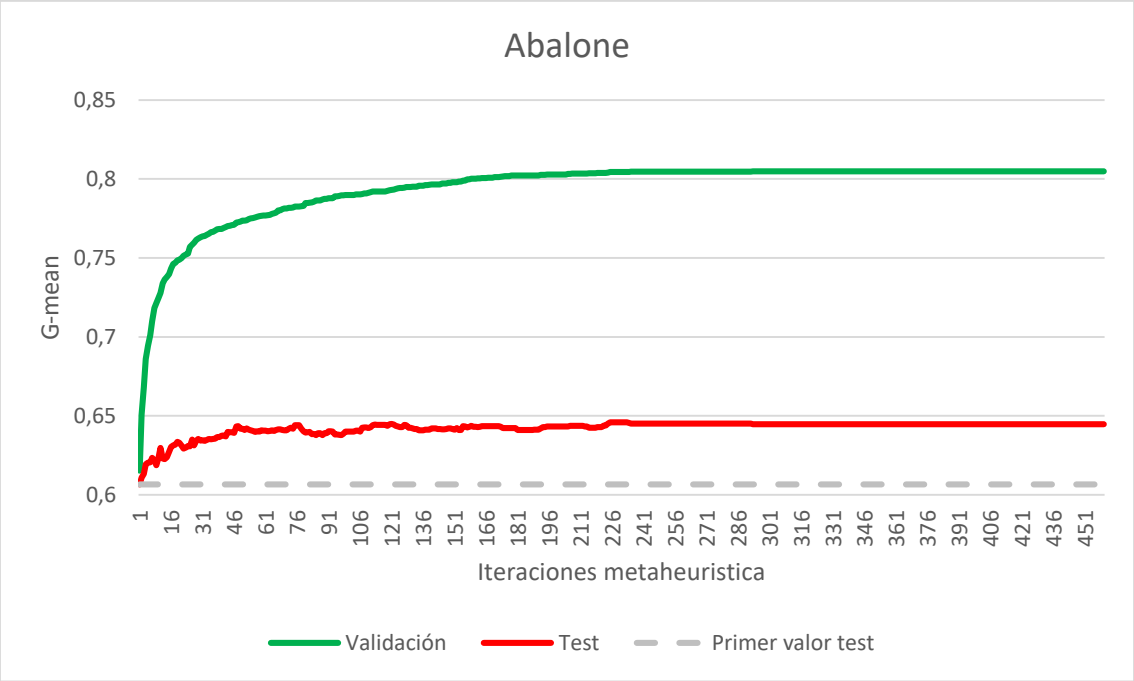

Gráfico 3: Media de resultados para el dataset abalone

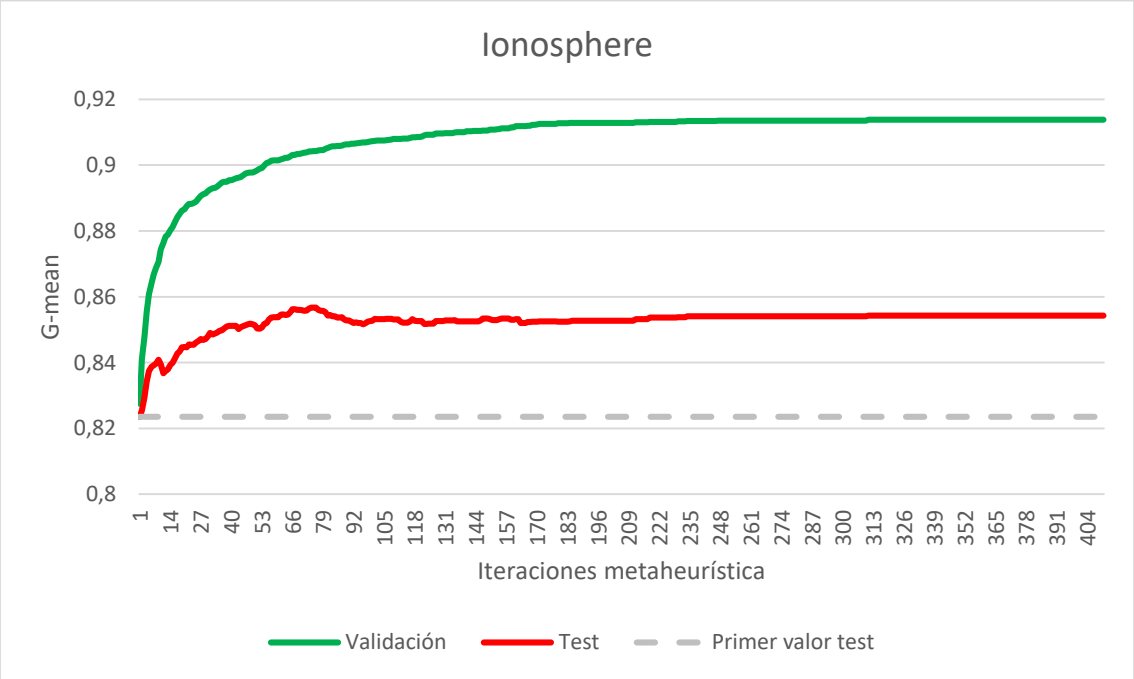

Gráfico 4: Media de resultados para el dataset ionosphere

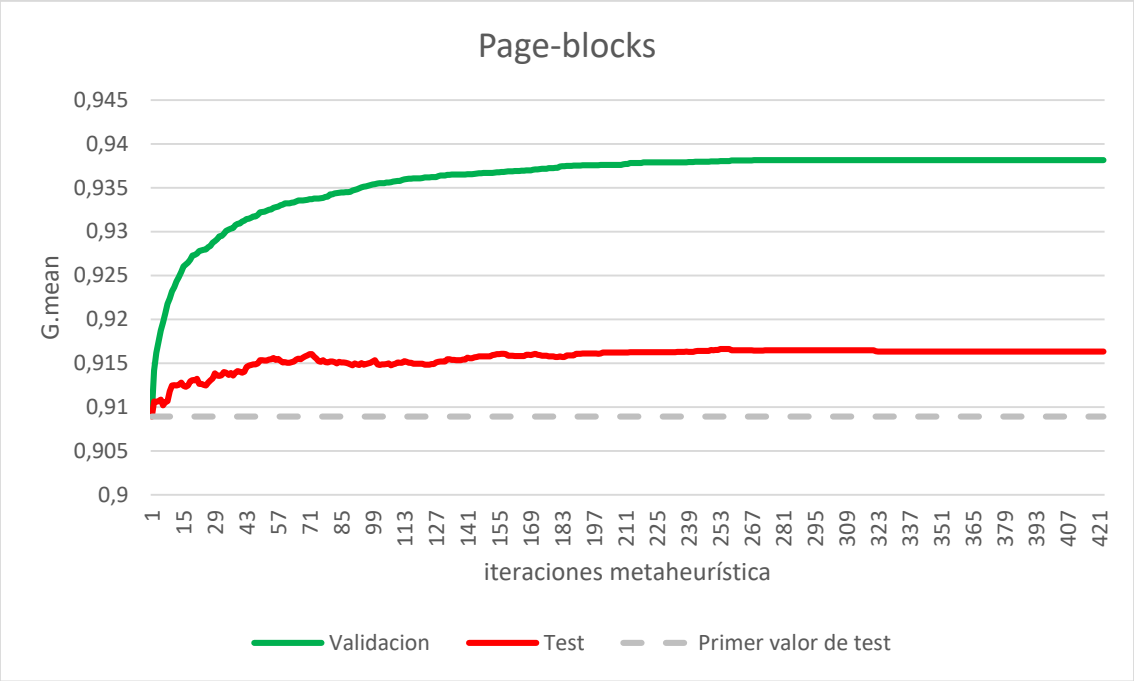

Gráfico 5: Media de resultados para el dataset page-blocks

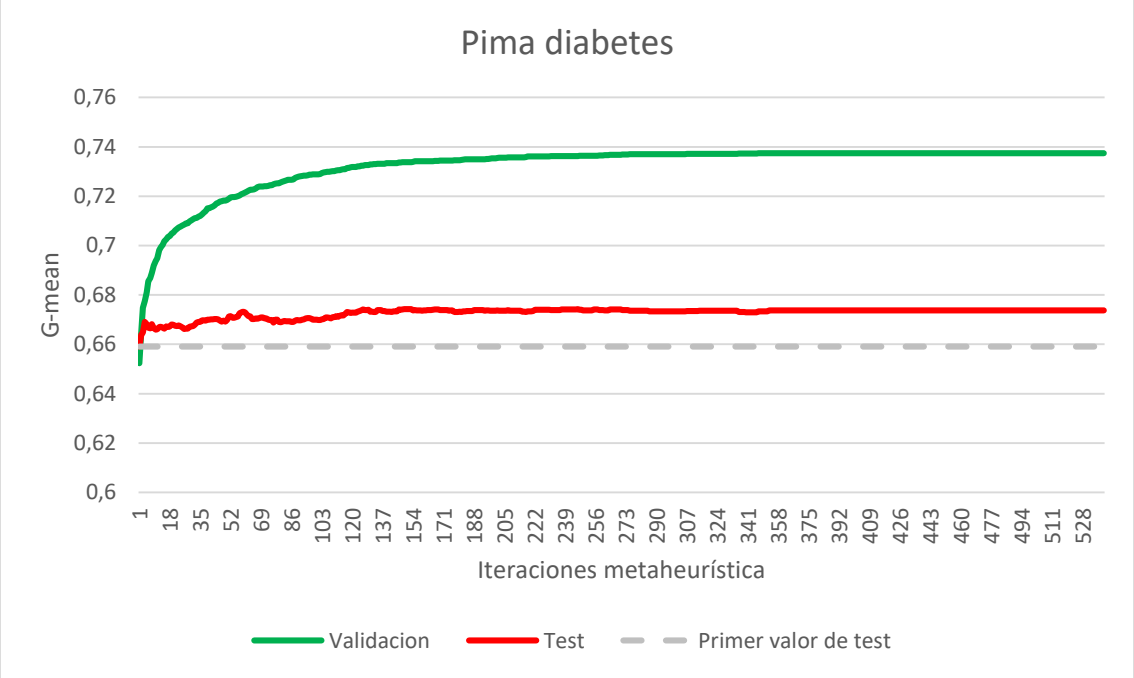

Gráfico 6: Media de resultados para el dataset pima diabetes

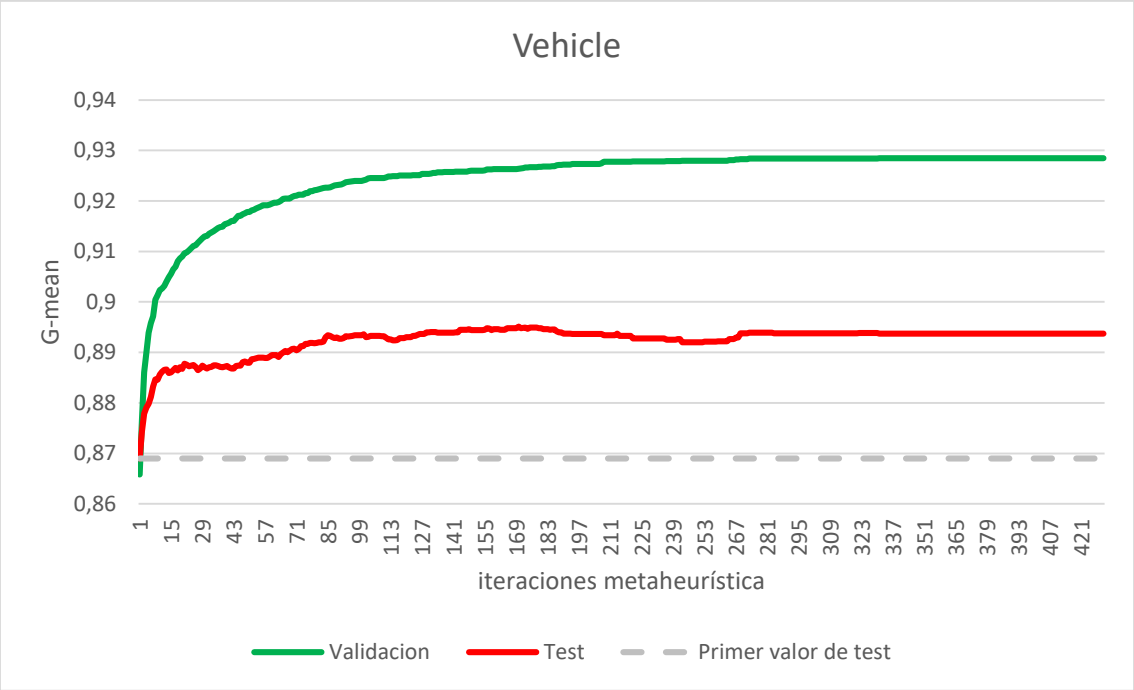

Gráfico 7: Media de resultados para el dataset vehicle

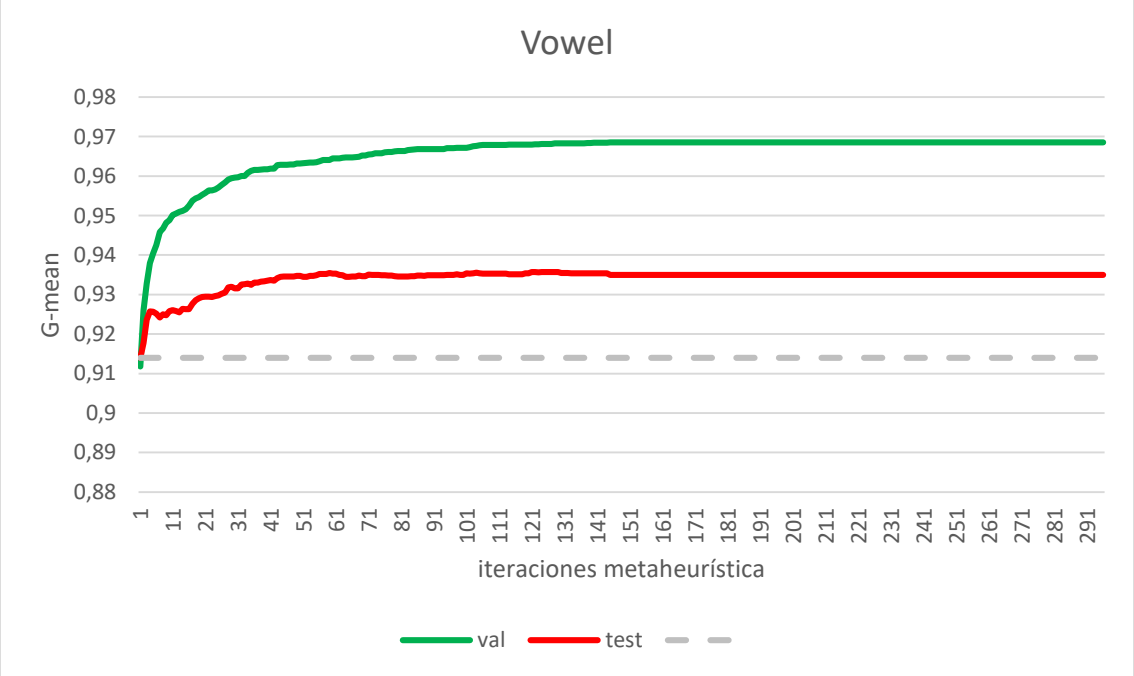

Gráfico 8: Media de resultados para el dataset vowel

En estos gráficos vemos claramente como la tendencia de los valores de test es alcanzar mejores valores de G-mean, y finaliza el proceso con un valor significativamente superior al valor de G-mean alcanzado por el primer conjunto de sintéticos sin evolucionar representado por la línea discontinua. Estás pruebas demuestran que durante la ejecución del algoritmo existe una evolución real llevada a cabo por parte de la metaheurística voraz iterativa aplicada al método de generación de patrones sintéticos.

Influencia del conjunto de sintéticos en la evolución de test conseguida

Como se mencionó en 10.2. Aplicación de Iterated Greedy, el árbol entrenado durante el proceso de validación y a través del cual se consigue el mejor valor para validación, se almacena y utiliza en la evaluación del sistema con el conjunto de *Test*. Este hecho podría llevar a pensar que la evolución demostrada en las pruebas anteriores podría no deberse a la obtención de mejores conjuntos de sintéticos, sino que podría estar relacionada con la búsqueda y utilización de un árbol mejor entrenado.

Para intentar demostrar la influencia positiva del conjunto en la clasificación se ha llevado a cabo la siguiente prueba.

1º. Se han almacenado el conjunto de sintéticos al comenzar la evolución de la metaheurística y el conjunto de sintéticos alcanzado al finalizar la misma para su comparación.

2º. Se entrenan y evalúan 150 árboles diferentes de forma independiente para cada uno de los dos conjuntos de sintéticos. Para entrenar cada árbol, se utiliza una semilla distinta.

3º. Se almacenan los resultados para ser evaluados.

A continuación, se muestran unos gráficos de caja o boxplot utilizando el G-mean alcanzado en la ejecucion de los 150 árboles para evaluar los resultados. A la izquierda aparecerá el boxplot referente al primer conjunto, y a la derecha al conjunto final conseguido mediante la evolucion de la metaheuristica.

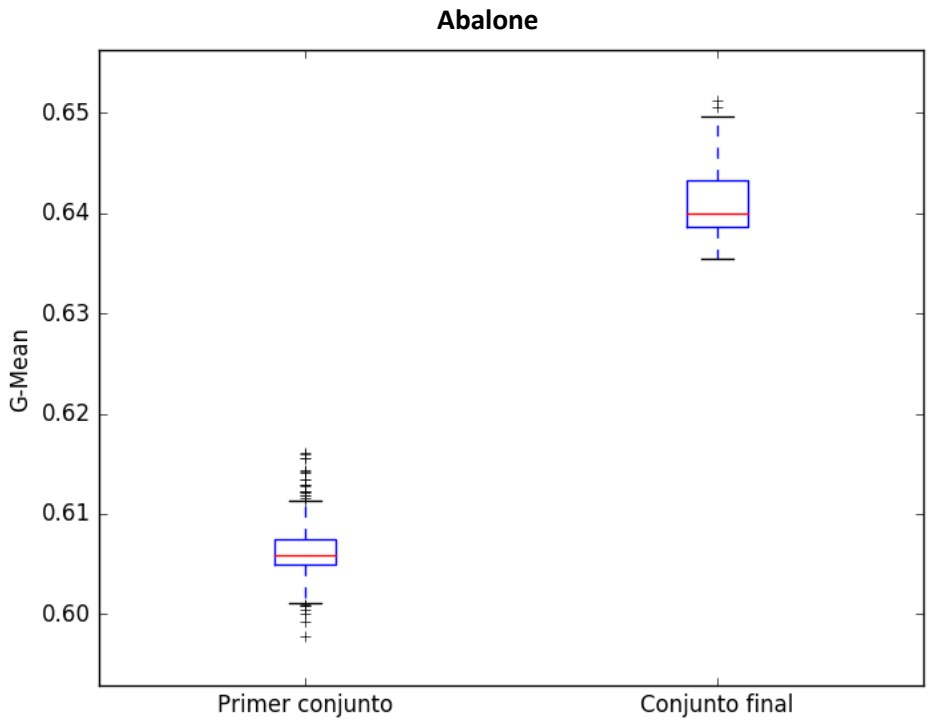

Gráfico 9: Comparación primer y último árbol para el dataset abalone

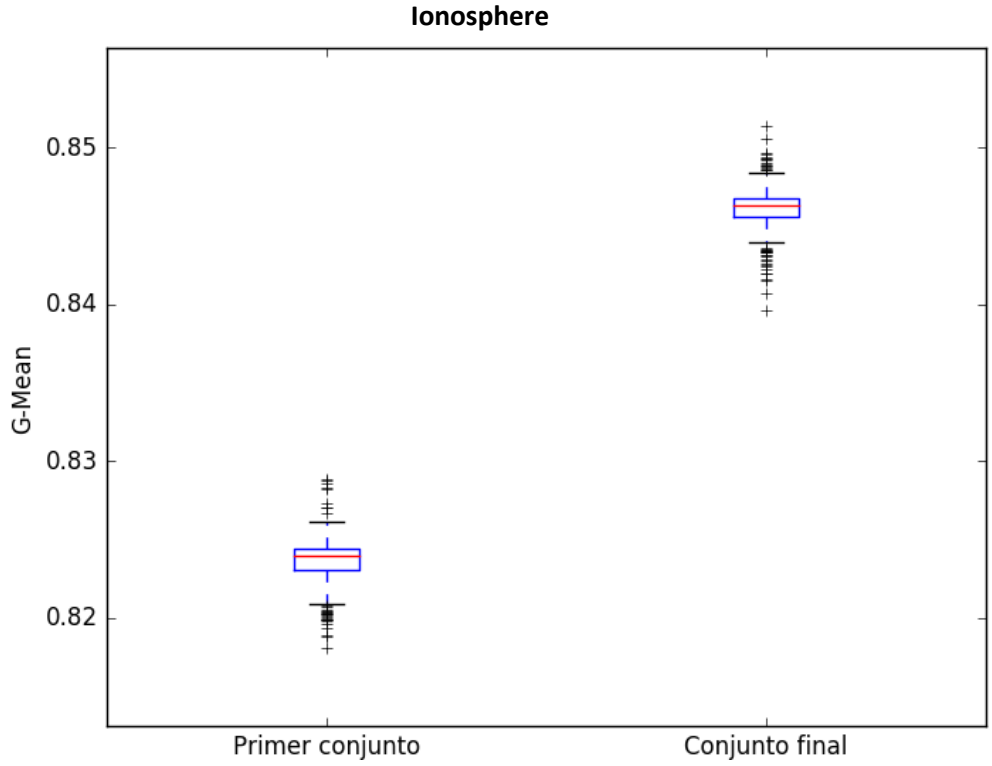

Gráfico 10: Comparación primer y último árbol para el dataset ionosphere

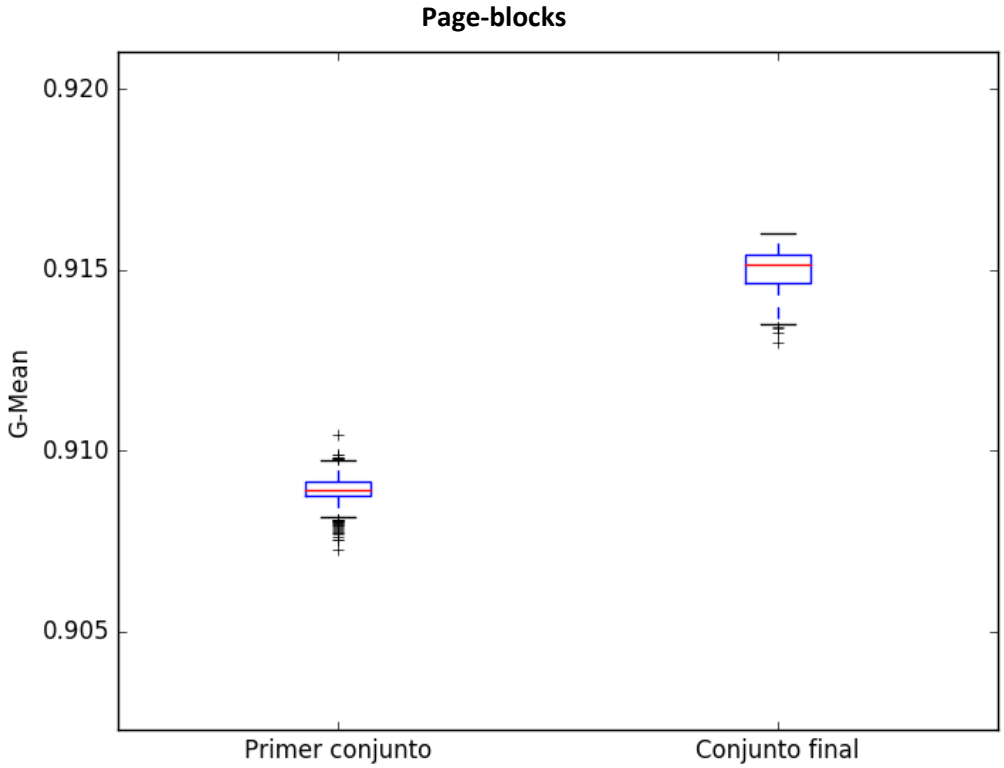

Gráfico 11: Comparación primer y último árbol para el dataset page-blocks

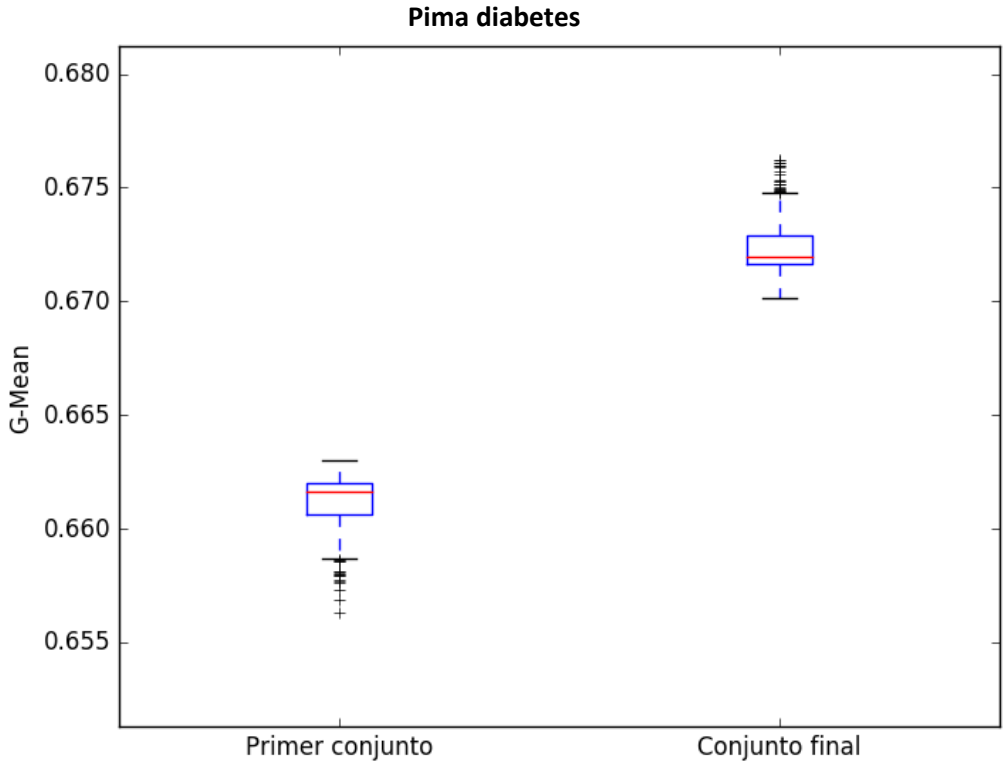

Gráfico 12: Comparación primer y último árbol para el dataset vehicle

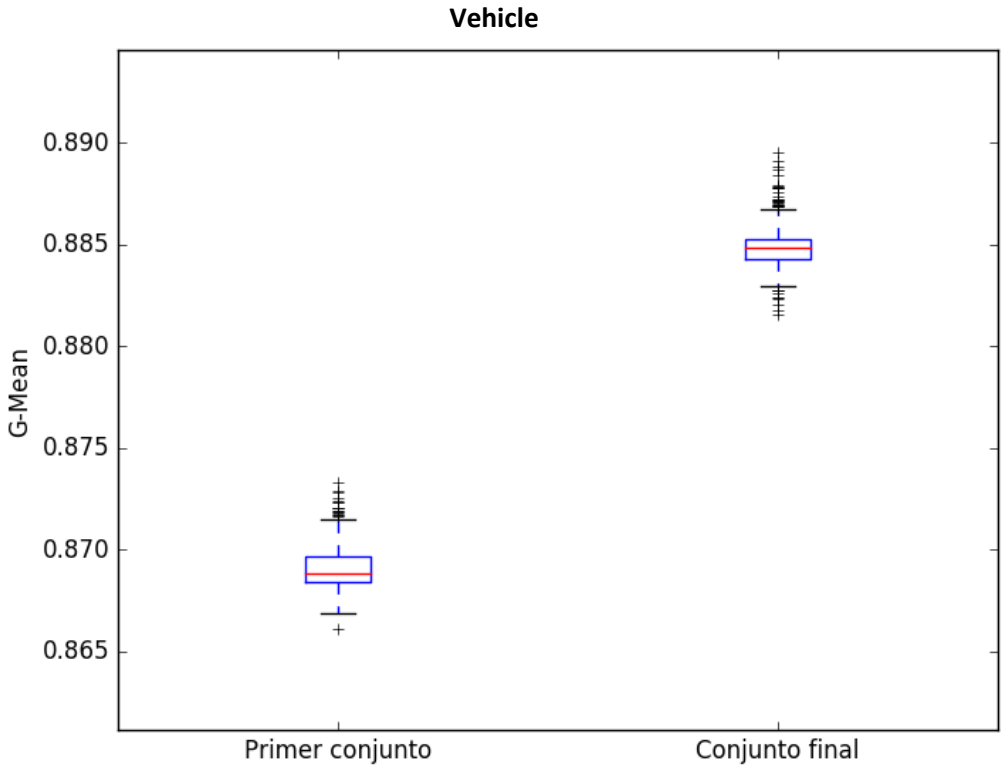

Gráfico 13: Comparación primer y último árbol para el dataset vehicle

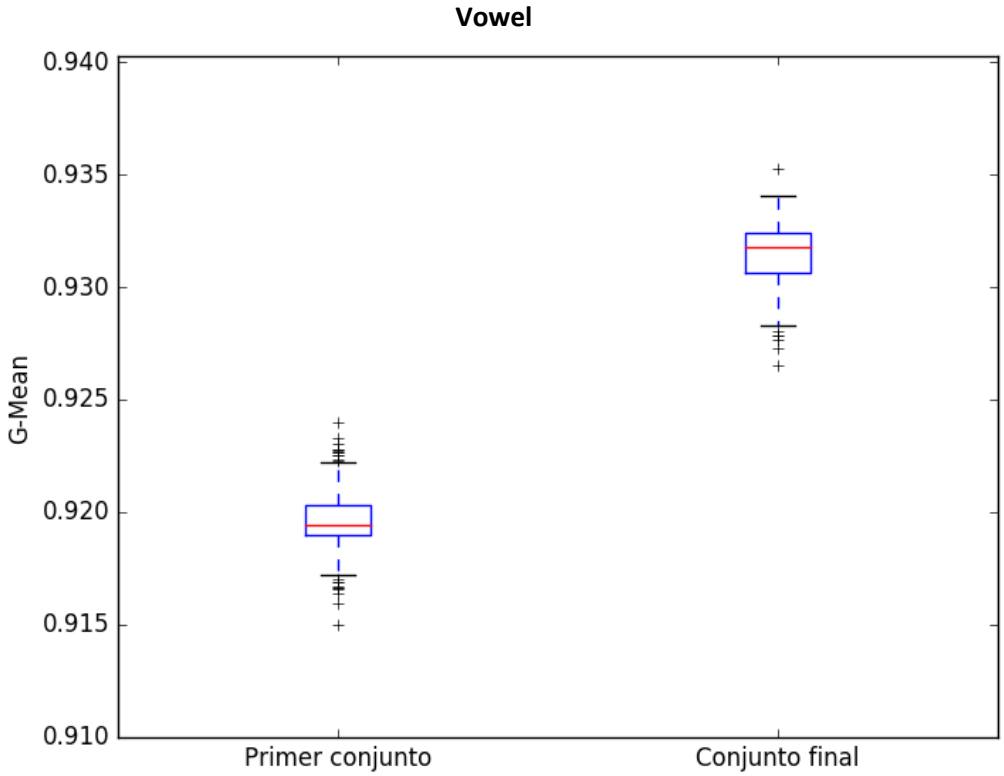

Gráfico 14: Comparación primer y último árbol para el dataset vowel

Como vemos, los valores alcanzados con los patrones evolucionados con la metaheurística son claramente superior a los valores alcanzados por los patrones antes de ser evolucionados, tanto que incluso no existe una superposición de los gráficos. Estos resultados nos llevan a concluir que el conjunto evolucionado alcanza mejores resultados independientemente del árbol utilizado y que, por tanto, la evolución demostrada en las pruebas anteriores se debe sobre todo a la obtención de mejores conjuntos de sintéticos.

Tras estas pruebas podemos afirmar que el algoritmo funciona satisfactoriamente y que es capaz de conseguir un conjunto de sintéticos capaz de influir positivamente en un clasificador para conseguir mejores resultados en test.

13.2.2. Comparación entre algoritmos

En esta sección se realizará una comparación entre varios algoritmos.

- **Adaptive Synthetic Sampling** (ADASYN)
- **Adaptive neighbors Synthetic Sampling** (ANESYN)
- **Adaptive neighbors Iterated Greedy Synthetic Sampling** (ANEIGSYN)

En todos los algoritmos se ha optado por utilizar el valor de *k* por defecto (número de vecinos a tener en cuenta). En el caso de ADASYN *k* = 5, mientras que ANESYN y ANEIGSYN utilizan un método adaptativo para calcular cuántos vecinos hacen falta tener en cuenta para cada base de datos de forma automática. También se ha optado por probar diferentes valores del parámetro *n*, que recordamos que es un factor de ponderación entre los dos factores de

importancia $\hat{f}_{j1}$ y $\hat{f}_{j2}$, utilizados para seleccionar un vecino cercano al borde que separa ambas clases, y al mismo tiempo, cercano al patrón original.

Las particiones de los datos para las pruebas ha sido la siguiente:

- **Entrenamiento** (*Train)*: 30%
- **Validación** (*Validation):* 20%
- **Test** (*test):* 50%

Para los algoritmos de ADASYN y ANESYN tendremos dos ejecuciones diferentes. Una de ellas utilizando un conjunto de *Train=30%,* y otra con *Train*=50% (*Train + Validation*).

Resultados comparativos

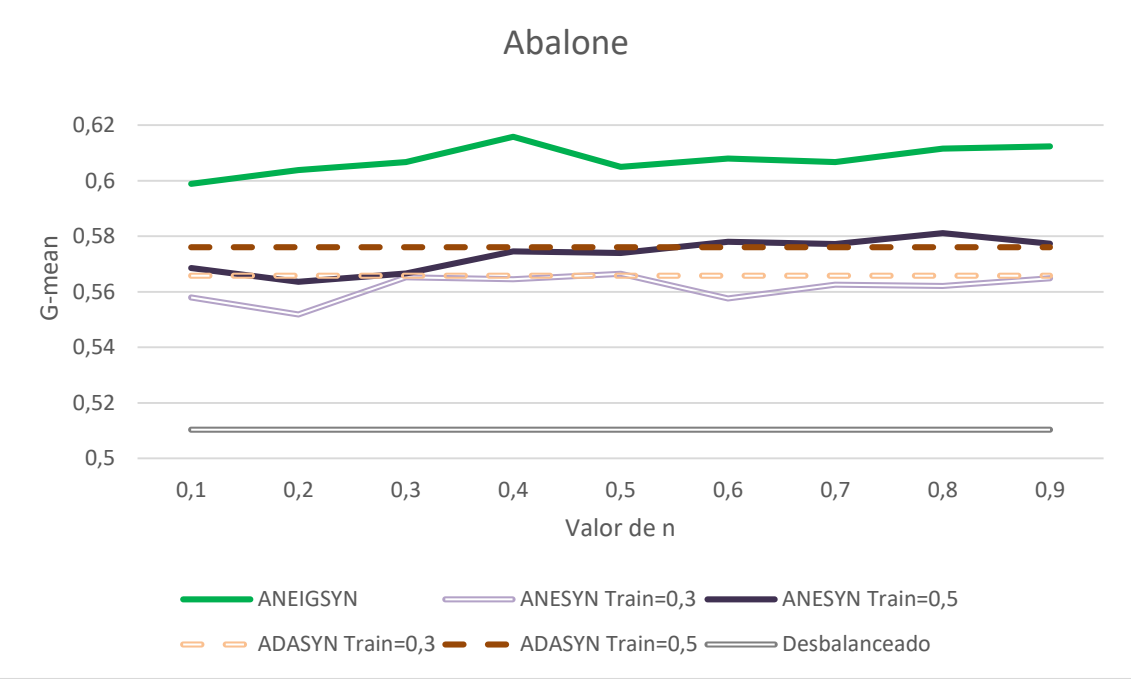

Gráfico 15: Comparación de algoritmos en Abalone

Abalone						
Algoritmo	**ANEIGSYN**	**ANESYN Train = 30%**	**ANESYN Train = 50%**	**ADASYN Train = 30%**	**ADASYN Train = 50%**	**Conjunto original**
n	0.8	0.8	0.8	--	--	--
OA	0.8800546	0.8951912	0.9004098	0.8920218	0.8966120	0,920239
Precisión Class1	0.9627648	0.9592728	0.9607135	0.9593766	0.9602072	0,956584
Precisión Class0	0.2286726	0.2382079	0.2548207	0.2329218	0.2435072	0,307333
Recall Class1	0.9079710	0.9283188	0.9325507	0.9246666	0.9288695	0,958949
Recall Class0	0.4214285	0.3509523	0.3723809	0.3557142	0.3666666	0,291765
F1_measure Class1	0.9342396	0.9433477	0.9463254	0.9415148	0.9441584	0,957669
F1_measure Class0	0.2899325	0.2765429	0.2986823	0.2752019	0.2888017	0,289975
G_mean	0.6115408	0.5621192	0.5811236	0.5657931	0.5760641	0,510315
AUC	0.6646997	0.6396356	0.6524658	0.6401904	0.6477681	0,625357
K	5	5	5	5	5	--

Tabla 63: Resultados Abalone

Nombre	Instancias totales	Clase minoritaria	Clase mayoritaria	Atributos
Abalone	731	42	689	7

Tabla 64: Distribución Abalone

A simple vista, podemos ver como el algoritmo de ANEIGSYN es claramente superior al resto, superando con más de un 10% al valor de G-mean obtenido con el conjunto original desbalanceado.

En este caso, la base de datos posee muy pocas instancias para la clase minoritaria, lo que supone que cuando se realiza la partición en subconjuntos (*Train, Test*), quede una cantidad mínima de instancias de la clase minoritaria para que el clasificador pueda aprender, o con las que generar un conjunto heterogéneo de sintéticos. El problema se acentúa aún más cuando el conjunto se divide en tres subconjuntos (*Train*, *Validation, Test*). Este hecho perjudica gravemente a la clasificación e impide que se puedan conseguir valores de precisión altos para ambas clases. Sin embargo, se considera que se ha conseguido un incremento considerablemente satisfactorio en la clasificación usando el algoritmo de ANEIGSYN.

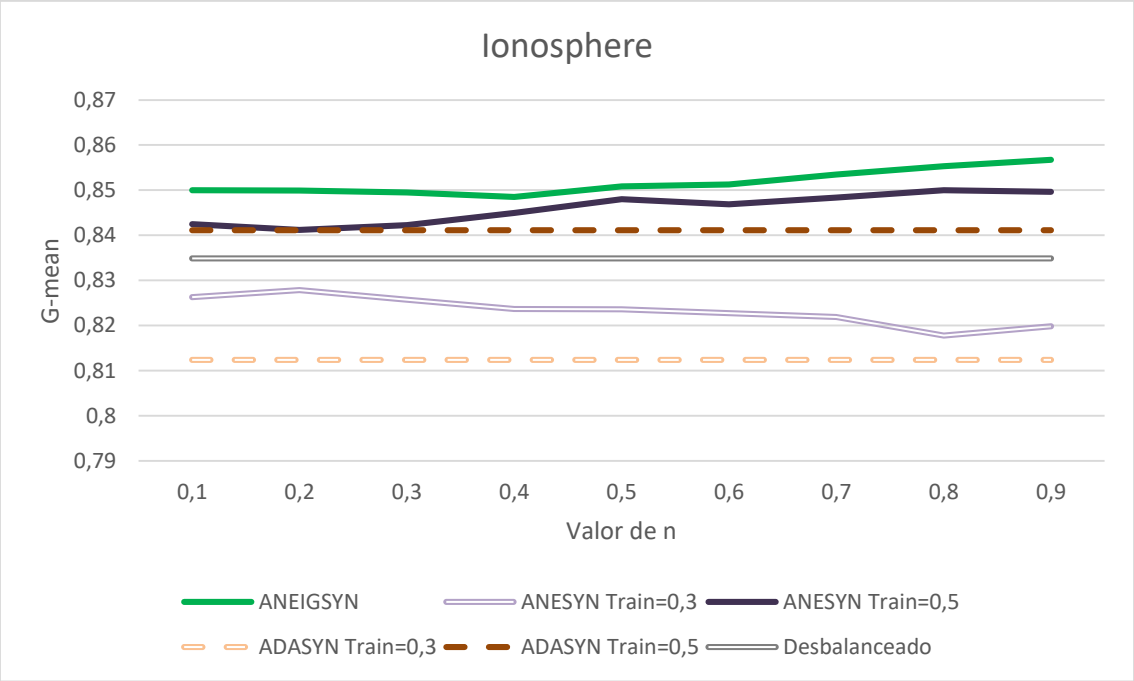

Gráfico 16: Comparación de algoritmos en Ionosphere

Ionosphere						
Algoritmo	**ANEIGSYN**	**ANESYN Train = 30%**	**ANESYN Train = 50%**	**ADASYN Train = 30%**	**ADASYN Train = 50%**	**Conjunto original**
n	0.9	0.9	0.9	--	--	--
OA	0.8711428	0.8417714	0.8652	0.8362857	0.8593142	0,858
Precision Class1	0.8992309	0.8723323	0.8945198	0.8659703	0.8865148	0,878799
Precision Class0	0.8253861	0.7933837	0.8182341	0.7888964	0.8141580	0,823743
Recall Class1	0.9019800	0.8859252	0.8978887	0.8844050	0.8976343	0,905667
Recall Class0	0.8155069	0.7622068	0.8062007	0.7494777	0.7901817	0,7722
F1_measure Class1	0.8999189	0.8778241	0.8953823	0.8739064	0.8913380	0,891248
F1_measure Class0	0.8183891	0.7737228	0.8095903	0.7649036	0.7996954	0,794345
G_mean	0.8567561	0.8198851	0.8496579	0.8123789	0.8411437	0,834903
AUC	0.8587435	0.8240660	0.8520447	0.8169413	0.8439080	0,838933
K	8.0	8.0	8.0	5	5	--

Tabla 65: Resultados Ionosphere

Nombre	Instancias totales	Clase minoritaria	Clase mayoritaria	Atributos
Ionosphere	351	126	225	34

Tabla 66: Distribución Ionosphere

Para Ionosphere, se puede apreciar en ANEIGSYN y ANESYN cierta tendencia a mejorar los resultados de G-mean para los valores altos del parámetro *n.* Un valor alto de *n* prioriza el hecho de elegir un vecino cercano al borde.

Es destacable el hecho que en esta base de datos el conjunto original obtiene unos resultados considerablemente altos, superando incluso a los métodos donde el conjunto de *Train* equivale al 30% de los datos, quedando estos claramente por debajo del resto.

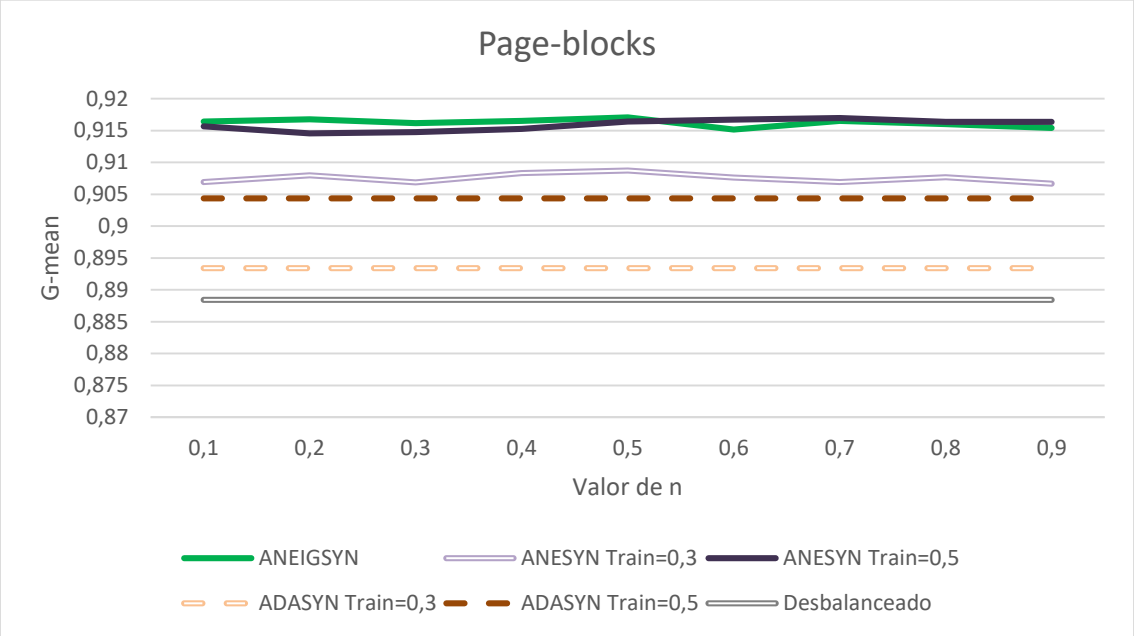

Gráfico 17: Comparación de algoritmos en Page-blocks

Page-blocks						
Algoritmo	**ANEIGSYN**	**ANESYN Train = 30%**	**ANESYN Train = 50%**	**ADASYN Train = 30%**	**ADASYN Train = 50%**	**Conjunto original**
n	0.2	0.2	0.2	--	--	--
OA	0.9451571	0.9467397	0.9540570	0.9538413	0.9589108	0,962458
Precision Class1	0.9862171	0.9839272	0.9846764	0.9798137	0.9818095	0,977899
Precision Class0	0.6791396	0.6932866	0.7329825	0.7503458	0.776321853	0,824549
Recall Class1	0.9522431	0.9563260	0.9638446	0.9685623	0.972266637	0,980351
Recall Class0	0.8828438	0.8624411	0.8679916	0.8243923	0.841463133	0,805491
F1_measure Class1	0.9689138	0.9699121	0.9741377	0.9741423	0.977004756	0,979115
F1_measure Class0	0.7671092	0.7679866	0.7943120	0.7849686	0.807059673	0,814374
G_mean	0.9167595	0.9079946	0.9145523	0.8933831	0.904361171	0,88842
AUC	0.9175434	0.9093835	0.9159181	0.8964773	0.906864885	0,892921
K	50.0	50.0	50.0	5	5	--

Tabla 67: Resultados Page-blocks

Nombre	Instancias totales	Clase minoritaria	Clase mayoritaria	Atributos
Page-blocks	5472	559	4913	10

Tabla 68: distribución Page-blocks

Page-blocks posee una alta cantidad de instancias, por lo que es la base de datos que menos se ve afectada por la partición de los subconjuntos. Al mismo tiempo, los algoritmos entrenados con el 30% de los datos funcionan mejor que en las otras base de datos utilizadas debido a dicho incremento del número de instancias, y al mismo tiempo, de la diversidad de los datos.

Curiosamente, podemos observar en page-blocks como se consiguen mejores resultados para ANESYN entrenado con *Train* = 30% que para ADASYN entrenado con *Train* = 50%. Esto puede ser debido a que ADASYN utiliza un método puramente aleatorio para seleccionar el vecino con el que se genera los sintéticos. Dependiendo de la distribución original del conjunto esto puede llevar a introducir patrones sintéticos en zonas conflictivas poco deseadas perjudicando de ese modo a la clasificación. En cambio, ANESYN utiliza un método que evita notablemente este fenómeno.

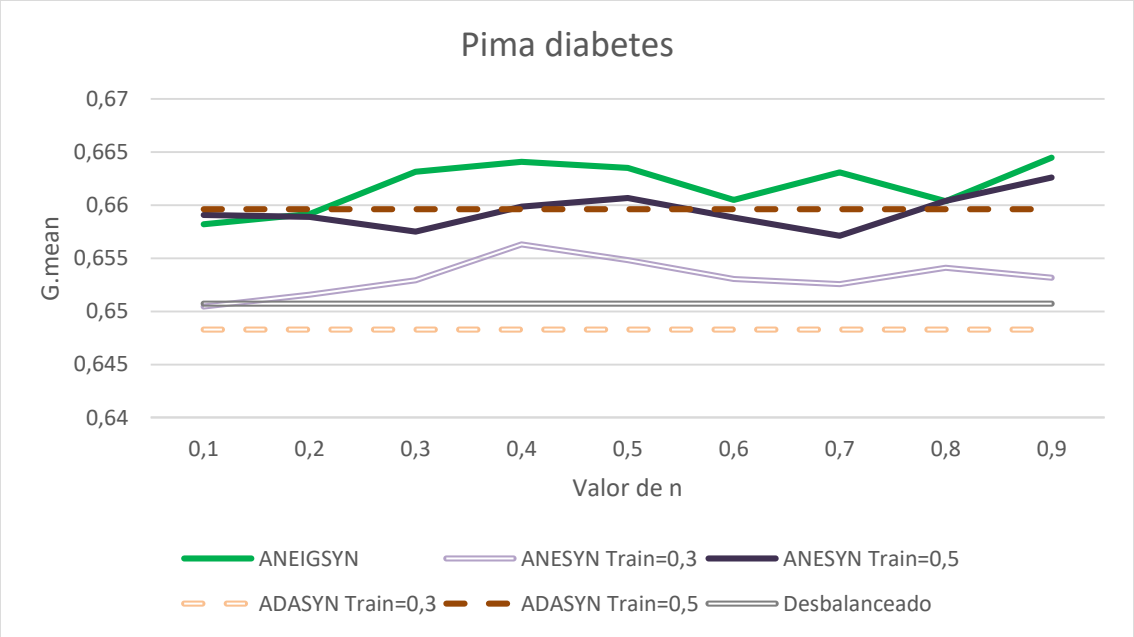

Gráfico 18: Comparación de algoritmos en Pima diabetes

Pima diabetes						
Algoritmo	**ANEIGSYN**	**ANESYN Train = 30%**	**ANESYN Train = 50%**	**ADASYN Train = 30%**	**ADASYN Train = 50%**	**Conjunto original**
n	0.9	0.9	0.9	--	--	--
OA	0.688697	0.679479	0.691692	0.679010	0.688828	0.689351
Precision Class1	0.775742	0.767704	0.772740	0.763081	0.770704	0.765394
Precision Class0	0.549983	0.538598	0.555774	0.538675	0.551601	0.553508
Recall Class1	0.73492	0.7288	0.74644	0.73596	0.74372	0.757065
Recall Class0	0.602462	0.587462	0.589552	0.572761	0.586417	0.562150
F1_measure Class1	0.754303	0.747138	0.758964	0.748786	0.756613	0.760474
F1_measure Class0	0.574073	0.560715	0.571276	0.554154	0.567696	0.555944
G_mean	0.664481	0.653163	0.662598	0.648283	0.659639	0.650720
AUC	0.668691	0.658131	0.667996	0.6543605	0.665068	0.659607
K	11.0	11.0	11.0	5	5	--

Tabla 69: Resultados Pima diabetes

Nombre	**Instancias totales**	**Clase minoritaria**	**Clase mayoritaria**	**Atributos**
Pima diabetes	768	268	500	8

Tabla 70: distribución Pima diabetes

En esta base de datos, no se aprecia un gran incremento en los resultados obtenidos con respecto al conjunto original desbalanceado, salvo por ANEIGSYN que consigue un incremento cercano al 2%. Por otro lado, ANESYN oscila ligeramente sobre el valor de ADASYN sin apreciarse cambios considerables, por lo que deducimos que la influencia de *n* en este casi no es significativa.

Se observa también, que ADASYN con TRAIN = 30% está por debajo del valor resultante al aplicar un árbol de decisión directamente al conjunto original desbalanceado. Esto puede deberse, a un tamaño excesivamente reducido den conjunto de *Train*, que pueda generar prácticamente una duplicidad en los datos con los problemas que eso conlleva, o a que el método de selección de vecinos de ADASYN no sigue ningún control para evitar generar patrones sintéticos en zonas de superposición de clases.

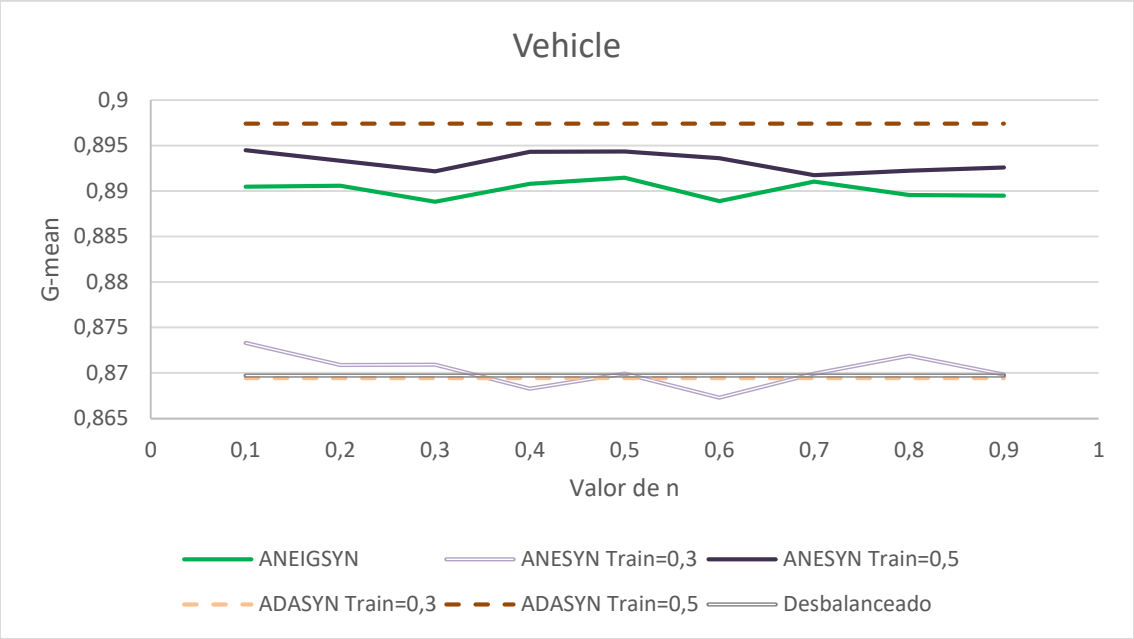

Gráfico 19: Comparación de algoritmos en Vehicle

Vehicle						
Algoritmo	**ANEIGSYN**	**ANESYN Train = 30%**	**ANESYN Train = 50%**	**ADASYN Train = 30%**	**ADASYN Train = 50%**	**Conjunto original**
n	0.1	0.1	0.1	--	--	--
OA	0.9167612	0.9064539	0.9220330	0.9061938	0.9245862	0,910147
Precision Class1	0.9522695	0.9439755	0.9530820	0.9411721	0.9542240	0,939978
Precision Class0	0.8103160	0.7929958	0.8276085	0.7972073	0.8337395	0,816847
Recall Class1	0.9385366	0.9335059	0.9448722	0.9361882	0.9471259	0,94305
Recall Class0	0.8458383	0.8183505	0.8476515	0.8085111	0.8511797	0,803625
F1_measure Class1	0.9452105	0.9385219	0.9488077	0.9385346	0.9505256	0,941277
F1_measure Class0	0.8264923	0.8039067	0.8361437	0.8016120	0.8411271	0,808065
G_mean	0.8904611	0.8733164	0.8944640	0.8694421	0.8974104	0,869722
AUC	0.8921875	0.8759282	0.89626187	0.8723496	0.8991528	0,873338
K	16	16	16	5	5	--

Tabla 71: Resultados Vehicle

Nombre	Instancias totales	Clase minoritaria	Clase mayoritaria	Atributos
Vehicle	846	199	647	18

Tabla 72: Distribución Vehicle

Los mejores resultados en esta ocasión se consiguen a través de ADASYN con *Train* = 50%, seguidamente de ANESYN con *Train* = 50%, y en tercer lugar se encuentra ANEIGSYN. Si observamos la precisión de ambas clases con el conjunto original, podemos ver que no existe una gran diferencia de clasificación entre las dos. Teniendo en cuenta que originalmente esta base de datos poseía 4 clases, y que fueron transformadas en 2 para poder aplicar estos algoritmos, es posible que como resultado del mestizaje y re-etiquetado de los datos, existan algunas instancias descolocadas de difícil aprendizaje, y que la mayor dificultad que presente el clasificador en este caso no sea al grado de desbalanceo, sino más bien la existencia de dichas instancias raras.

Como se ha visto en la base de datos de "Pima diabetes", el algoritmo de ADASYN que utiliza un conjunto de Train = 30% consigue resultados por debajo del conjunto desbalanceado original.

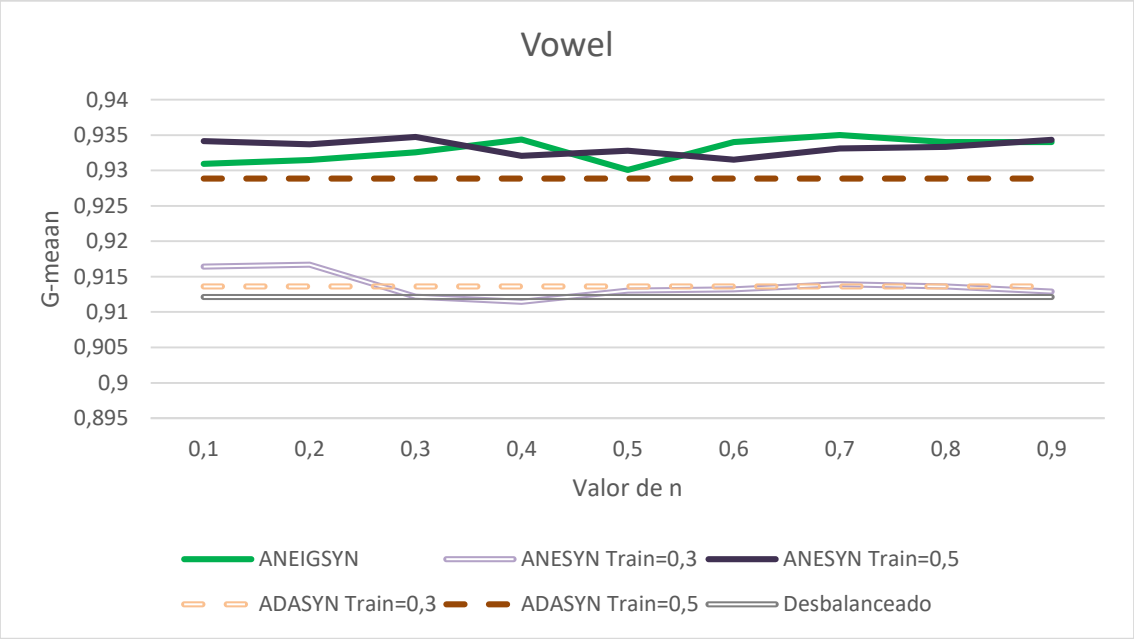

Gráfico 20: Comparación de algoritmos en Vowel

Vowel						
Algoritmo	**ANEIGSYN**	**ANESYN Train = 30%**	**ANESYN Train = 50%**	**ADASYN Train = 30%**	**ADASYN Train = 50%**	**Conjunto original**
n	0.7	0.7	0.7	--	--	--
OA	0.9680606	0.9635959	0.9706868	0.9647676	0.9725858	0,97244444
Precision Class1	0.9897076	0.9859046	0.9890560	0.9857818	0.9880107	0,98470992
Precision Class0	0.7870893	0.7764597	0.8133666	0.7846902	0.8343450	0,85499656
Recall Class1	0.9750666	0.9739777	0.9786444	0.9754222	0.9818222	0,98504444
Recall Class0	0.898	0.8597777	0.8911111	0.8582222	0.8802222	0,84644444
F1_measure Class1	0.9822951	0.9798356	0.9837774	0.9805096	0.9848686	0,98484309
F1_measure Class0	0.8363963	0.8114727	0.8473718	0.8153896	0.8537479	0,84786217
G_mean	0.9350029	0.9139465	0.9331265	0.9136147	0.9288539	0,91211872
AUC	0.9365333	0.9168777	0.9348777	0.9168222	0.9310222	0,91574444
K	8.0	8.0	8.0	5	5	--

Tabla 73: Resultados Vowel

Nombre	Instancias totales	Clase minoritaria	Clase mayoritaria	Atributos
Vowel	990	90	900	10

Tabla 74: Distribución Vowel

Vowel también fue una de las bases datos que fue necesario transformar para conseguir solo dos clases, por lo que no sería un ejemplo natural en el que aplicar este tipo de algoritmos. En el Gráfico 20, podemos ver como ANEIGSYN y ANESYN oscilan entre ellos sin destacar ninguno de los dos, pero ambos se mantienen por encima de ADASYN.

Una vez más podemos ver como los algoritmos que utilizan un conjunto de Train=30% están por debajo en cuanto a los resultados obtenidos.

13.2.3. Conclusiones generales de los resultados

Tras examinar los resultados, podemos percibir que el algoritmo expuesto en el presente documento (ANEIGSYN), suele destacar de manera general sobre los demás, seguido en la mayoría de las ocasiones de su versión sin *Iterated Greedy* (ANESYN). En cuanto a la comparación de ANESYN con ADASYN, distinguimos como normalmente se consiguen mejores resultados con ANESYN, llegando a superarlo aun cuando se utiliza un conjunto de entrenamiento porcentualmente más pequeño, como se ha visto en "Page-blocks"

También podemos apreciar que no siempre hay una clara tendencia en cómo influye *n* como variable de ponderación entre los dos factores de importancia a la hora de elegir con que

vecino se genera un sintético, y que para conseguir un mejor resultado puede ser necesario probar varias configuraciones diferentes.

Es destacable el hecho de que los algoritmos que utilizan un conjunto de *Train* formado por el 50% de los datos, aprovechan el incremento de instancias para realizar un entrenamiento más ventajoso, que se refleja en un mejor resultado. No obstante, el algoritmo de ANEIGSYN es capaz de conseguir muy buenos resultados aun cuando solo utiliza el 30% de los datos para el conjunto de *Train* gracias a la optimización de la metaheurística mediante el conjunto de validación.

Parte V
Conclusiones

Capítulo 14:
Conclusiones formales

Es este capítulo se indicará cuáles han sido los objetivos que se han alcanzado con la realización de este Proyecto. Una vez se ha finalizado el desarrollo del sistema y tras haber superado exitosamente las pruebas realizadas al mismo, es posible concretar que se han cumplido los requisitos especificados es este documento (Capítulo 8: Especificación de requisitos.)

14.1. Objetivos de formación alcanzados

En este apartado hará referencia a los conocimientos adquiridos y desarrollados durante la realización del Trabajo.

- Se ha desarrollado una mayor destreza en el uso del lenguaje de programación *Python*.

- Se ha generado una mayor experiencia en el manejo y tratamiento de bases de datos desbalanceadas.

- Se ha estudiado en profundidad la utilización de las librerías científicas y matemáticas existentes en *Python* tales como *Scikit-Learn*, *SciPy* o *NumPy*.

- Se ha conseguido una mayor habilidad en el manejo de estructuras de datos.

- Se ha obtenido un mayor conocimiento sobre todo el proceso que conlleva elaborar un sistema y las herramientas de apoyo necesarias, desde la implementación del código hasta la elaboración del manual técnico o el manual de usuario.

- Se ha llevado a cabo un estudio teórico sobre temáticas relacionadas con la detección del borde de las clases, la selección de patrones, técnicas de sobremuestreo o over-sampling y metaheurísticas.

- Se han ampliado conocimientos relacionados con el tratamiento de archivos .CSV y hojas de cálculo, debido al tratamiento de archivos necesario en este trabajo.

- Se han desarrollado conocimientos de configuración de algoritmos para adaptarlos a diferentes necesidades.

- Se ha conseguido un mayor conocimiento en el análisis y comparación de los resultados del sistema.

14.2. Objetivos operacionales alcanzados

Los requisitos operacionales que se han podido alcanzar durante el desarrollo del sistema son los siguientes.

- Se ha desarrollado con éxito un sistema funcional que equilibra la distribución de conjuntos de datos desbalanceados mediante la generación de patrones sintéticos.

- Se ha conseguido aplicar eficazmente una metaheurística que optimiza la generación de patrones sintéticos.

- Se ha conseguido que la aplicación sea capaz de realizar el tratamiento necesario al conjunto de datos original, sin la necesidad de un procesamiento externo al sistema, para que el algoritmo pueda ser aplicado a dicho conjunto sin problema.

- Se ha dotado de flexibilidad y de la capacidad de configuración por parte del usuario tanto al algoritmo de over-sampling como a sus módulos e incluso a la metaheurística que lo optimiza.

- Se ha dotado al sistema de la posibilidad de mostrar al usuario por pantalla el resultado alcanzado por el sistema, así como de poder ser almacenado en un archivo.

- Se ha evaluado la eficacia del algoritmo mediante un conjunto de problemas de prueba, y se han refinado hasta la obtención de unos resultados óptimos.

Debido a la consecución de todos los objetivos anteriores, consideramos que se han alcanzado todas las metas propuestas de forma muy satisfactoria.

14.3. Conclusiones y comentarios sobre el sistema

Tras el análisis del sistema y de sus objetivos finalizaremos con este capítulo esbozando una serie de conclusiones sobre el sistema en sí mismo.

Las técnicas de over-sampling pueden ser optimizadas mediante una metaheurística como demuestran los resultados obtenidos en este TFG. La optimización de la generación de patrones sintéticos a través de la aplicación de *Iterated Greedy*, ha sido implementada de una forma funcional y altamente satisfactoria y ha sido demostrada con éxito. Teniendo en cuenta que la partición en 3 subconjuntos de un dataset desbalanceado puede ser perjudicial para un clasificador, ya que se obtendría un conjunto de *Train* con una cantidad reducida de instancias de la clase minoritaria, la metaheurística consigue optimizar el aprendizaje del clasificador con este conjunto y obtener buenos resultados.

Aun cuando la principal dificultad que pueda presentar un dataset para un clasificador sea la existencia de instancias raras difíciles de aprender, la aplicación de técnicas de over-sampling supone una ayuda positiva en la clasificación de conjuntos desbalanceados como se explica en el apartado 7.1. Aprendizaje sobre conjunto de datos desbalanceados.

Independientemente de la aplicación de *Iterated Greedy*, ANESYN presenta dos interesantes aportaciones al funcionamiento de ADASYN que han demostrados ser funcionales en las pruebas realizadas:

1. **Calculo adaptativo de la vecindad:** ANEIGSYN y ANESYN pueden calcular de forma automática cuantos vecinos han de tenerse en cuenta a la hora de calcular la cercanía de un patrón al borde de separación de las clases.

2. **Selección del vecino adecuado:** ANEIGSYN y ANESYN utilizan un método de selección del vecino más adecuado para generar un sintético, esto no solo potencia el borde que separa ambas clases, sino que además evita la creación de sintéticos en zonas conflictivas como podría pasar en ADASYN.

La mayoría de los algoritmos de aprendizaje desequilibrados presuponen que la información útil sobre el problema a tratar está disponible antes del proceso de aprendizaje. Sin embargo, en muchos problemas del mundo real como las redes de sensores móviles, la minería Web, la vigilancia, la seguridad nacional y las redes de comunicación, los datos pueden aparecer de forma incremental en pequeños trozos repartidos durante un periodo de tiempo específico. Ante esta situación, un algoritmo debe ser capaz de acumular experiencia previa y adaptarse a la información adicional. Gracias a la capacidad adaptativa de ANEIGSYN y ANESYN para calcular cuanta vecindad que debe ser analizada en cualquier momento, y gracias a la redistribución de g_i (número de patrones a generar por cada patrón original) implementada para el correcto funcionamiento de *Iterated Greedy*, sería muy sencillo adaptar ANEIGSYN y ANESYN para este tipo de problemas, a diferencia de otros algoritmos.

Además, la aplicación cuenta con una configuración paramétrica bastante amplia sin la necesidad de grandes conocimientos del sistema, pero podría conseguirse una mayor exactitud y flexibilidad si se recurre a la configuración avanzada como se explica en el apartado 13.1.2 (Parámetros de configuración).

Una vez se han analizado todos los aspectos destacables relacionados con el sistema, y habiéndose cumplido todos los requisitos para los que se diseñó, consideraremos que la aplicación es válida.

Capítulo 15:
Futuras mejoras

En el presente capítulo intentaremos definir aquellas consideraciones a tener en cuenta, que aplicadas a nuestro sistema, podrían mejorar de alguna forma el desempeño del propio del sistema y aumentar su utilidad.

15.1. Mejoras del algoritmo

- Implementar un criterio de aceptación en la metaheurística más seguro que evite algunos problemas como el sobreentrenamiento y explore diferentes regiones del espectro de soluciones. Por ejemplo, enfriamiento simulado.

- Implementar una lista tabú a *Iterated Greedy* para evitar volver a soluciones exploradas anteriormente.

- Combinar con métodos de ensemble como Bagging y Boosting.

- Combinar alguna técnica de over-sampling como ANEIGSYN, con técnicas de under-sampling implementadas de forma poco intrusiva para no perder gran cantidad de datos. Un ejemplo podría ser un método de eliminación de ruido que en lugar de eliminar los patrones de la clase minoritaria, los eliminara de la clase mayoritaria.

- Combinar una solución a nivel de datos como es ANEIGSYN, con otra a nivel de algoritmo como por ejemplo Adaboost, que incrementa el peso de los patrones mal clasificados y decrementa a los que se han clasificado de forma correcta, para entrenar de forma más eficaz al clasificador.

- Actualmente, el sistema puede aplicarse a problemas con varias clases transformándolos en un problema bi-clase, pero podría adaptarse para realizar un sobremuestreo de cada clase de forma individual y conseguir un funcionamiento mejor adaptado a la naturaleza de los problemas multiclases.

- Algunos clasificadores como G-mean experimentan una variación discreta a "saltos" dependiendo del número de instancias bien clasificadas, es decir, que una pequeña variación en la clasificación puede provocar un salto en G-mean. Sería interesante, sobre todo para poder llevar a cabo una evolución mediante la metaheurística de forma más continua e incremental, utilizar otras medidas de *fitness* como el error cuadrático medio, y otros clasificadores como SVM o redes neuronales.

15.2. Otras posibles mejoras

Una vez comentadas algunas posibles futuras mejoras a tener en cuenta, en este apartado se enumerarán algunas aportaciones de otra índole:

- Incorporación de una interfaz gráfica que facilite la utilización y configuración del algoritmo.
- Incorporación del algoritmo a alguna librería pública para facilitar su acceso a la mayor cantidad de gente posible.
- Facilitar un método para lanzar una batería de ejecuciones por medio de scripts.
- Incorporación de gráficos adicionales para poder realizar otros tipos de análisis.
- Traducir este documento a varios idiomas para mejorar su internacionalización
- Crear un video tutorial explicativo con ejemplos de uso

Capítulo 16: Bibliografía

[1] H. He y E. A. Garcia, «Learning from Imbalanced Data», *IEEE Trans. Knowl. Data Eng.*, vol. 21, n.º 9, pp. 1263-1284, sep. 2009.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, y W. P. Kegelmeyer, «SMOTE: Synthetic Minority Oversampling TEchnique», *J. Artif. Intell. Res.*, vol. 16, pp. 321-357, 2002.

[3] H. He, Y. Bai, E. A. Garcia, y S. Li, «ADASYN: Adaptive synthetic sampling approach for imbalanced learning», en *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322-1328.

[4] S. Barua, M. M. Islam, X. Yao, y K. Murase, «MWMOTE–Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning», *IEEE Trans. Knowl. Data Eng.*, vol. 26, n.º 2, pp. 405-425, feb. 2014.

[5] *Pycharm*. JetBrains.

[6] G. van Rossum, B. Warsaw, y N. Coghlan, «PEP 8–style guide for python code», *Available Ht Tpwww Python Orgdevpepspep-0008*, 2001.

[7] The Python Software Foundation., «Python 2.7.12 documentation», oct-2016. [En línea]. Disponible en: https://docs.python.org/2.7/. [Accedido: 01-sep-2016].

[8] F. Pedregosa *et al.*, «Scikit-learn: Machine Learning in Python», *J. Mach. Learn. Res.*, vol. 12, n.º Oct, pp. 2825-2830, 2011.

[9] S. van der Walt, S. C. Colbert, y G. Varoquaux, «The NumPy Array: A Structure for Efficient Numerical Computation», *Comput. Sci. Eng.*, vol. 13, n.º 2, pp. 22-30, mar. 2011.

[10] W. McKinney, «pandas: a Python data analysis library», *See Httppandas Pydata Org*, 2015.

[11] J. D. Hunter y others, «Matplotlib: A 2D graphics environment», *Comput. Sci. Eng.*, vol. 9, n.º 3, pp. 90–95, 2007.

[12] G. E. A. P. A. Batista, R. C. Prati, y M. C. Monard, «A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data», *SIGKDD Explor Newsl*, vol. 6, n.º 1, pp. 20–29, jun. 2004.

[13] D. J. Dittman, T. M. Khoshgoftaar, R. Wald, y A. Napolitano, «Comparison of Data Sampling Approaches for Imbalanced Bioinformatics Data.», en *FLAIRS Conference*, 2014.

[14] I. Tomek, «Two modifications of CNN», *IEEE Trans Syst. Man Cybern.*, vol. 6, pp. 769–772, 1976.

[15] P. Hart, «The condensed nearest neighbor rule (Corresp.)», *IEEE Trans. Inf. Theory*, vol. 14, n.º 3, pp. 515-516, may 1968.

[16] N. V. Chawla, «Data Mining for Imbalanced Datasets: An Overview», en *Data Mining and Knowledge Discovery Handbook*, O. Maimon y L. Rokach, Eds. Springer US, 2005, pp. 853-867.

[17] H. Han, W.-Y. Wang, y B.-H. Mao, «Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning», en *Advances in Intelligent Computing*, D.-S. Huang, X.-P. Zhang, y G.-B. Huang, Eds. Springer Berlin Heidelberg, 2005, pp. 878-887.

[18] C. Bunkhumpornpat, K. Sinapiromsaran, y C. Lursinsap, «Safe-Level-SMOTE: Safe-Level-Synthetic Minority Over-Sampling TEchnique for Handling the Class Imbalanced Problem», en *Advances in Knowledge Discovery and Data Mining*, T. Theeramunkong, B. Kijsirikul, N. Cercone, y T.-B. Ho, Eds. Springer Berlin Heidelberg, 2009, pp. 475-482.

[19] F. Glover, «Future paths for integer programming and links to artificial intelligence», *Comput. Oper. Res.*, vol. 13, n.º 5, pp. 533-549, ene. 1986.

[20] S. H. Zanakis y J. R. Evans, «Heuristic "Optimization": Why, When, and How to Use It», *Interfaces*, vol. 11, n.º 5, pp. 84-91, oct. 1981.

[21] I. H. Osman y J. P. Kelly, *Meta-Heuristics: Theory and Applications*. Springer Science & Business Media, 2012.

[22] R. Ruiz y T. Stützle, «A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem», *Eur. J. Oper. Res.*, vol. 177, n.º 3, pp. 2033-2049, mar. 2007.

[23] M. Glinz, «On Non-Functional Requirements», en *15th IEEE International Requirements Engineering Conference (RE 2007)*, 2007, pp. 21-26.

[24] F. Provost y G. M. Weiss, «The effect of class distribution on classifier learning: an empirical study», *Dept Comput Sci Rutgers Univ Newark*, 2001.

[25] J. Laurikkala, «Improving Identification of Difficult Small Classes by Balancing Class Distribution», en *Artificial Intelligence in Medicine*, S. Quaglini, P. Barahona, y S. Andreassen, Eds. Springer Berlin Heidelberg, 2001, pp. 63-66.
[26] T. E. Oliphant, «Python for Scientific Computing», *Comput. Sci. Eng.*, vol. 9, n.º 3, pp. 10-20, may 2007.
[27] G. Van Rossum y others, «Computer programming for everybody», *Propos. Corp. Natl. Res. Initiat.*, 1999.
[28] B. Venners, «The Making of Python Conversation with Guido van Rossum», *Part Httpwww Artima ComintvpythonP Html Artima Artic. Interviews Httpwww Artima Comarticlesindex Jsp*, 2003.
[29] T. Peters, «The Zen of Python», en *Pro Python*, C. Andres, S. Anglin, M. Beckner, E. Buckingham, G. Cornell, J. Gennick, J. Hassell, M. Lowman, M. Moodie, D. Parkes, J. Pepper, F. Pohlmann, D. Pundick, B. Renow-Clarke, D. Shakeshaft, M. Wade, T. Welsh, M. Tobin, N. Sixsmith, y A. Alchin, Eds. Apress, 2010, pp. 301-302.
[30] L. Rosen, *Open source licensing*, vol. 692. Prentice Hall, 2005.
[31] C. J. Van Rijsbergen, *Information retrieval*, 2nd ed. London ; Boston: Butterworths, 1979.
[32] M. Kubat y S. Matwin, «Addressing the Curse of Imbalanced Training Sets: One-Sided Selection», en *In Proceedings of the Fourteenth International Conference on Machine Learning*, 1997, pp. 179–186.
[33] J. A. Swets y others, «Measuring the accuracy of diagnostic systems», *Science*, vol. 240, n.º 4857, pp. 1285–1293, 1988.
[34] A. P. Bradley, «The use of the area under the ROC curve in the evaluation of machine learning algorithms», *Pattern Recognit.*, vol. 30, n.º 7, pp. 1145–1159, 1997.
[35] R. S. Pressman y J. M. Troya, *Ingeniería del software*. McGraw Hill, 1988.

Parte VI
Apéndices

Manual de usuario

Este documento constituye el manual de usuario de la aplicación implementada en *Python* para el Trabajo Fin de Grado "*Creación de patrones sintéticos para conjuntos de datos desbalanceados mediante la metaheurística voraz iterativa*"

A.1. Descripción del producto

El producto explicado en este manual, es una aplicación que integra un algoritmo para sobremuestreo de conjuntos desbalanceados optimizado mediante una metaheurística. Todo el sistema se engloba bajo un mismo archivo o *script* para facilitar su ejecución. El sistema podrá ser configurado mediante unos parámetros para ajustar su correcto funcionamiento a cualquier base de datos asociada a un problema de clasificación. Adicionalmente, contará con una configuración inicial genérica y algunos métodos de autoconfiguración para abstraer al usuario final de esta tarea lo máximo posible.

El sistema debe ser capaz de generar un conjunto de sintéticos que ayude a un clasificador en el proceso de entrenamiento para que pueda conseguir una mejor clasificación en conjuntos de datos desbalanceados. Los resultados que se consigan pueden ser mostrados por pantalla y almacenados en un archivo CSV

Entre las características del sistema encontramos las siguientes:

- Desarrollado mediante la utilización del lenguaje de programación *Python*.
- Incorporación de métodos de tratamiento sobre base de datos de entrada para adaptarlos al sistema.
- Opciones de configuración.
- Incorporación de métodos de autoconfiguración.
- Generación de archivos con los resultados.

A.2. Requisitos del sistema

La aplicación puede ser utilizada bajo cualquier sistema operativo que incorpore *Python* 2.7, entre los que se incluye GNU/Linux, OS X, Windows debido a la capacidad multiplataforma de *Python*. Para facilitar la ejecución se recomienda la instalación de *Python* Anaconda. Esta distribución está especialmente preparada para ser usada como un entorno virtual de *Python* para uso científico, donde se han incluido una extensa variedad de librerías totalmente

actualizadas. Esto permite el uso de nuestra aplicación sin comprometer en ningún momento al sistema, la versión de *Python* o las librerías instaladas, es decir, este entorno está aislado de nuestra aplicación gracias a Anaconda.

Por tanto, los requisitos mínimos de nuestra aplicación, van a depender de los requisitos mínimos de anaconda en cada Sistema operativo.

A.2.1. Requisitos mínimos

- **Procesador:** Intel Pentium 3 con 500Mhz o superior
- **RAM:** 1024MB
- **Almacenamiento:** 3GB para poder almacenar las bases de datos de entrada y los archivos de salida

A.2.2. Requisitos software

- **Sistema operativo:** OS X, GNU/Linux o Windows

A.3. Instalación del software

Como se ha mencionado anteriormente será necesario instalar el entorno de *Python*, Anaconda. Se recomienda la instalación en usuarios locales, ya que de este modo se evita la concesión de permisos de administrador, y facilita la tarea de aislamiento que se desea conseguir.

Existe una versión más liviana que no incorpora ninguna de las librerías añadidas al entrono que se va a instalar, esta versión es llamada *Miniconda*[7] y solo incorpora *Python* y *Conda*.

A.3.1. Anaconda para Windows

Para instalar Anaconda en Windows es necesario seguir los siguientes pasos:

1º. Descarga el instalador de Anaconda correspondiente a Windows desde aquí[8].
2º. Haz doble clic en el archivo .exe descargado para abrir la ventana gráfica.
3º. Sigue los pasos del instalador hasta el final.
4º. Selecciona la localización que desees para instalar *Anaconda*.
5º. Listo, Anaconda debería estar instalado correctamente

IMPORTANTE:

1. No instalar como administrador a menos que sea necesario

[7] Miniconda: http://conda.pydata.org/miniconda.html
[8] Anaconda para Windows: https://www.continuum.io/downloads

2. En caso de encontrar algún error durante la instalación es posible que sea necesario desactivar el antivirus e intentar instalarlo de nuevo. Una vez instalación ha finalizado correctamente, puede volver a habilitar el antivirus sin ningún problema.

Si la instalación ha finalizado con éxito, debe ver una ventana similar a la de la siguiente imagen:

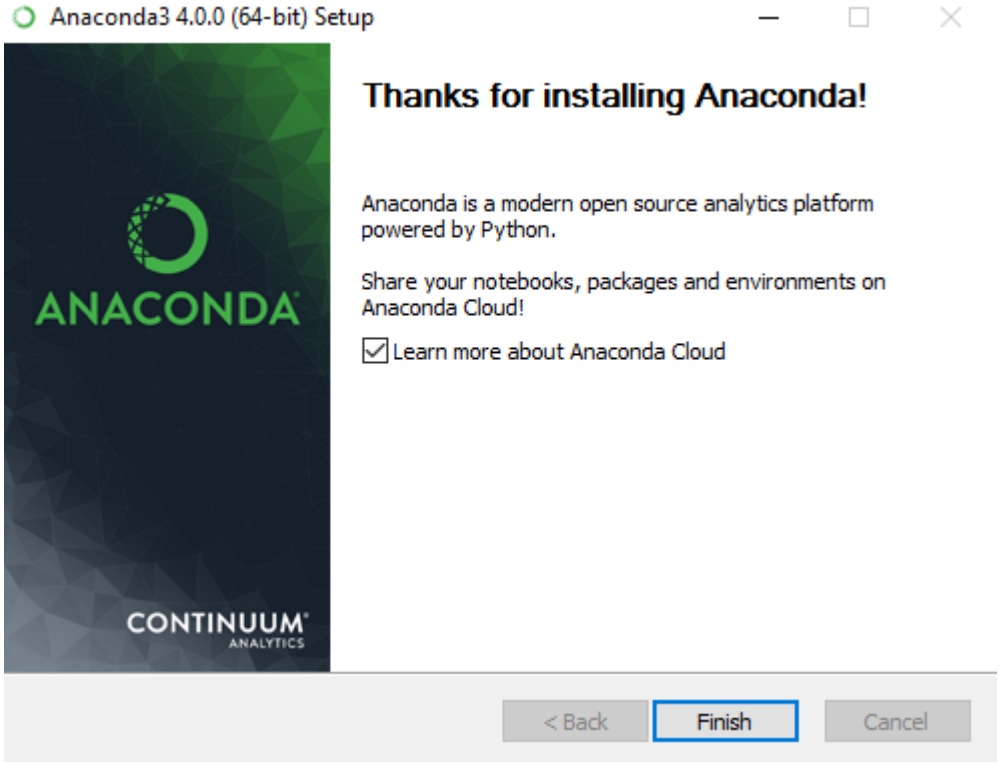

Figura A 1: Instalación exitosa Anaconda para Windows

A.3.2. Anaconda para OS X

Para instalar Anaconda en OS X es necesario seguir los siguientes pasos:

3. Descarga el instalador de Anaconda correspondiente a OS X desde aquí[9].
4. Haz doble clic en el archivo .pkg descargado para abrir la ventana grafica
5. Sigue los pasos del instalador y selecciona, "instalar solo para mí" cuando sea necesario
6. Selecciona la localización que desees para instalar *Anaconda* (por defecto se instalará en la carpeta de usuario).
7. Los usuarios avanzados podrán seleccionar no añadir anaconda *al bash PATH*.
8. Listo, Anaconda debería estar instalado correctamente

[9] Anaconda para OSX: https://www.continuum.io/downloads

IMPORTANTE: Si durante la instalación se obtiene el mensaje de error "No puedes instalar Anaconda en esta localización" vuelve a seleccionar "Instalar solo para mí"

Si la instalación se ha completado con éxito deberá ver una ventana como la siguiente:

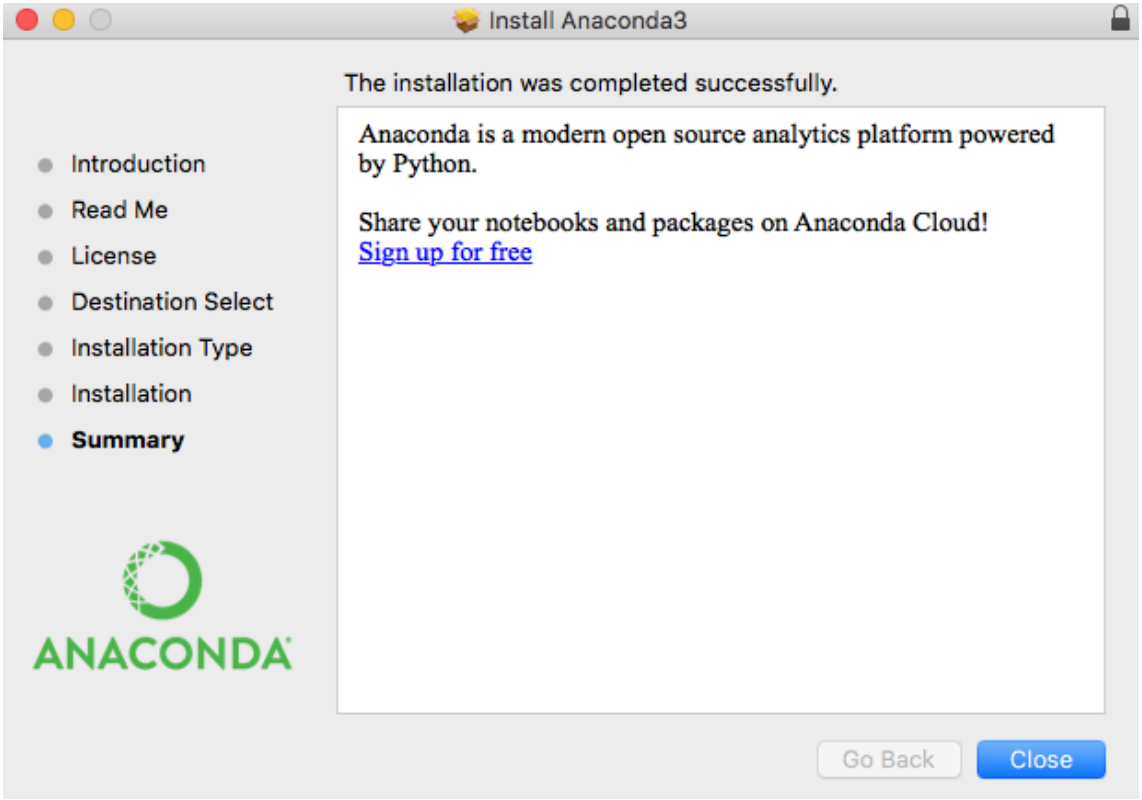

Figura A 2: Instalación exitosa Anaconda para OSX

A.3.3. Anaconda para GNU/Linux

En Linux solo es posible instalar Anaconda usando línea de comandos.

Para realizar la instalación siga los siguientes pasos:

1º. Descarga el instalador de Anaconda correspondiente a GNU/Linux desde aquí[10].
2º. En Consola escribe el siguiente comando: ***/Downloads/Anaconda3-4.0.0-Linux-x86 64.sh***
Remplace *Downloads* por la carpeta donde se encuentre el archivo descargado y remplace además *Anaconda3-4.0.0-Linux-x86 64.sh* por el nombré del archivo que haya descargado.

Es posible cambiar algunas configuraciones de la instalación, como por ejemplo la ruta de instalación, pero si no está seguro sobre algunas de estas opciones de configuración se recomienda el uso de las opciones por defecto. Estas podrán ser cambiadas posteriormente.

[10] Anaconda pata Linux: https://www.continuum.io/downloads

Si la instalación se ha completado con éxito deberá aparecer un mensaje en la terminal anunciando que todo ha ido bien, normalmente este tipo de mensajes suele ser algo parecido a "instalación finalizada". Para asegurarse de que los cambios están disponibles para su uso se recomienda cerrar y reabrir el terminal.

A.3.4. Actualizar Anaconda

Para actualizar Anaconda basta con abrir una terminal y escribir los siguientes comandos:

conda update conda
conda update anaconda

A.3.5. Desinstalar anaconda

- **Windows:** Accede al panel de control y selecciona "Desinstalar un programa", busca "*Python* 3.5 (Anaconda)" y haz clic en desinstalar.
- **GNU/Linux y OS X:** Abre un terminal y borra completamente el directorio de Anaconda, que generalmente se llama *anaconda2* o anaconda3. Para ello, haz uso del siguiente comando:

rm -rf ~/anaconda2
rm -rf ~/anaconda3

A.4. Instalación de la aplicación

La aplicación será facilitada en un CD-ROM junto a este manual. Al tratarse de un script de *Python*, la aplicación no requiere de ningún tipo de instalación, pudiendo ser incluso ejecutada desde el mismo CD-ROM en el que es proporcionada, pero aun así, se recomienda crear un directorio especifico en el sistema y transferir los archivos a este directorio para que la lectura de los datos sea más rápida y para poder almacenar los archivos generados de forma correcta. No obstante, aunque la aplicación en sí misma no requiera de instalación, es necesaria la instalación de Anaconda, como ya se ha explicado.

A.5. Desinstalación de la aplicación

Como se ha comentado, la aplicación no requiere de ninguna instalación, por tanto, para llevar a cabo la eliminación de la aplicación será suficiente con eliminar del sistema el script original y los archivos creados en la ejecución, en caso de haber sido copiado al sistema.

Si se ha seguido la recomendación de crear un directorio específico, tan solo será necesario eliminar dicho directorio para la eliminación completa de la aplicación.

A.6. Ejecución de la aplicación

A.6.1. Algoritmo ANEIGSYN

Para ejecutar la aplicación abriremos una terminal y nos desplazaremos al directorio donde hemos almacenado el script. Una vez allí ejecutaremos el siguiente comando:

./ANEIGSYN.py RutaBaseDeDatos Semilla

Donde "Ruta" es la ruta completo del directorio donde se encuentran la base de datos que quieres cargar como por ejemplo: **basesDatos/csv/pimadiabetes.csv**

Y donde semilla es un número entero cualquiera que servirá como semilla aleatoria para controlar la aleatoriedad de las ejecuciones del algoritmo. Se recomienda utilizar una permutación de los números del 1 al 8, por ejemplo: **25631487**

Ejemplo de comando de ejecución completo:

./ANEIGSYN.py basesDatos/csv/vehicle.csv 65231478

Se ha incluido una versión de este algoritmo que genera más archivos para poder llevar a cabo un estudio más exhaustivo. Esta versión se llama *ANEIGSYN _Graphics,* y se ejecuta con el siguiente comando:

./ANEIGSYN_Graphics.py basesDatos/csv/vehicle.csv 65231478

A.6.2. Algoritmo ANESYN

Este algoritmo es una versión de ANEIGSYN que no incorpora la metaheurística *Iterated Greedy*. Su ejecución, por tanto, será más liviana y menos costosa a nivel computacional, implicando en la mayoría de los casos que los resultados que se alcancen sean inferiores.

Su ejecución sigue exactamente la misma estructura, solo que en este caso debemos usar el nombre apropiado de este algoritmo.

Ejemplo de comando de ejecución completo:

./ANESYN.py basesDatos/csv/vehicle.csv 65231478

A.6.3. Representación de gráficas

Tras la ejecución del algoritmo *ANEIGSYN _Graphics*, se habrán creado una serie de archivos con datos sobre la evolución llevada a cabo por la metaheurística que pueden ser interesante analizar posteriormente.

Se ha añadido un *script* de *Python* que leerá estos archivos y mostrará por pantalla una gráfica con colores para facilitar su análisis.

Se ejecutará muy fácilmente a través del siguiente comando.

```
./imprimir.py
```

A.7. Configuración de los parámetros

En este apartado explicaremos cómo configurar los distintos parámetros que condicionarán el funcionamiento del algoritmo. Para ello será necesario abrir el *script* con algún editor de texto, dirigirnos a la primera función de configuración creada para este fin, y cambiar el valor por defecto de los algunos parámetros que se consideren necesarios.

Los parámetros que pueden ser modificados son:

Nombre	Descripción	Intervalo de valores	Valor por defecto
test_size	Tamaño porcentual del conjunto de *Test.*	$(0, 1)$	0.5
train_size	Tamaño porcentual del conjunto de *Train*	$(0, 1)$	0.3
val_size	Tamaño porcentual del conjunto de *Validation.*	$[0, 1)$	1 – (test_size + train_size)
runs	Número de veces que se ejecutará el algoritmo	$[2, \infty)$	100
beta	Especifica el nivel de balanceo que será cubierto después de la generación de sintéticos.	$(0, 1]$	1
k	Número de vecinos a tener en cuenta pata detectar e borde. Si es None el sistema será capaz de calcular de forma automática cuantos vecinos es necesario tener en cuenta según la distribución de cada base de datos.	$[1, \infty)$, None	None
n	Factor de ponderación entre $\hat{f}_{j1}$ y $\hat{f}_{j2}$. Se utilizará para seleccionar a los vecinos con los que generar patrones sintéticos.	$(0, 1)$	0.7
val_auc	Si es True, la validación se realiza en función del área bajo la curva ROC.	[True, False]	False
pintar_graficas	Si es True, sacará una gráfica en cada ejecución.	[True, False]	False
remove_noise	Si es True, se activa la función de eliminación de ruido.	[True, False]	False
imbalanced_degree	Variable que se utilizará para especificar la clase minoritaria en tiempo de ejecución. Toda clase que obstente una presencia menor que *imbalanced_degree* será considerada minoritaria y se unirán en una sola clase. Las clases restantes formaran la clase mayoritaria. Si no se especifica se considerara como clase minoritaria solo la más minoritaria de todas.	$(0, 1]$	None

Figura A 3: Parámetros de configuración

A.8. Archivos generados

La ejecución de cualquiera de los algoritmos incluidos creará un archivo llamado *Results.csv*, con el siguiente formato:

	Nombre algoritmo
OA	0.907021857923
Precision Class1	0.960000253108
Precision Class0	0.270204834504
Recall Class1	0.940608695652
Recall Class0	0.355238095238
F1_measure Class1	0.950119991125
F1_measure Class0	0.303062985329
G_mean	0.568115812275
AUC	0.647923395445

Figura A 4: Ejemplo ilustrativo del archivo Results

Donde podremos ver los valores de varias medidas de precisión calculadas al finalizar el algoritmo, es decir, en el caso de ANEIGSYN y ANESYN serían los valores sobre el conjunto balanceado después de la generación de patrones sintéticos.

Si se ejecuta el algoritmo *ANEIGSYN _Graphics,* se generaran varios archivos adicionales.

- **validaciones.csv**: Archivo con todos los valores de G-mean o AUC (según el parámetro *val_auc*) referentes al conjunto de validación, y con los que se puede estudiar cómo ha evolucionado esta medida a lo largo de la ejecución del sistema. Este archivo tendrá tantas columnas como veces se halla ejecutado el sistema (configurable mediante el parámetro *runs*), y un número de filas variable dependiendo de las iteraciones que se hayan necesitado durante la evolución de la metaheurística.

- **testeos.csv**: Archivo con todos los valores de G-mean o AUC (según el parámetro *val_auc*) referentes al conjunto de test, y con los que se puede estudiar cómo ha evolucionado esta medida a lo largo de la ejecución del sistema. Este archivo tendrá tantas columnas como veces se halla ejecutado el sistema (configurable mediante el parámetro *runs*), y un número de filas variable dependiendo de las iteraciones que se hayan necesitado durante la evolución de la metaheurística.

- **betterSet_Trees.csv:** Durante cada ejecución del sistema, se generan 150 árboles diferentes entrenados por el primer conjunto que genera ANEIGSYN antes de ser evolucionado, y se evalúan usando el conjunto de test (la evaluación se hace con G-mean o AUC según el parámetro *val_auc*). Este archivo está formado por 150 columnas (recogiendo las evaluaciones de cada árbol) y por un número de filas igual al número de ejecuciones del sistema (configurable mediante el parámetro *runs*).

- **firstSet_Trees.csv:** Durante cada ejecución del sistema, se generan 150 árboles diferentes entrenados por el último conjunto que genera ANEIGSYN una vez que se ha completado la evolución, y se evalúan usando el conjunto de test (la evaluación se hace con G-mean o AUC según el parámetro *val_auc*). Este archivo está formado por 150 columnas (recogiendo las evaluaciones de cada árbol) y por un número de filas igual al número de ejecuciones del sistema (configurable mediante el parámetro *runs*).

A.8.1. Gráficas

SI después de ejecutar *ANEIGSYN_Graphics,* se ejecuta el archivo *imprimir.py*, se mostraran por pantalla unas gráficas similares a las que se muestran en Figura A 5 y Figura A 6.

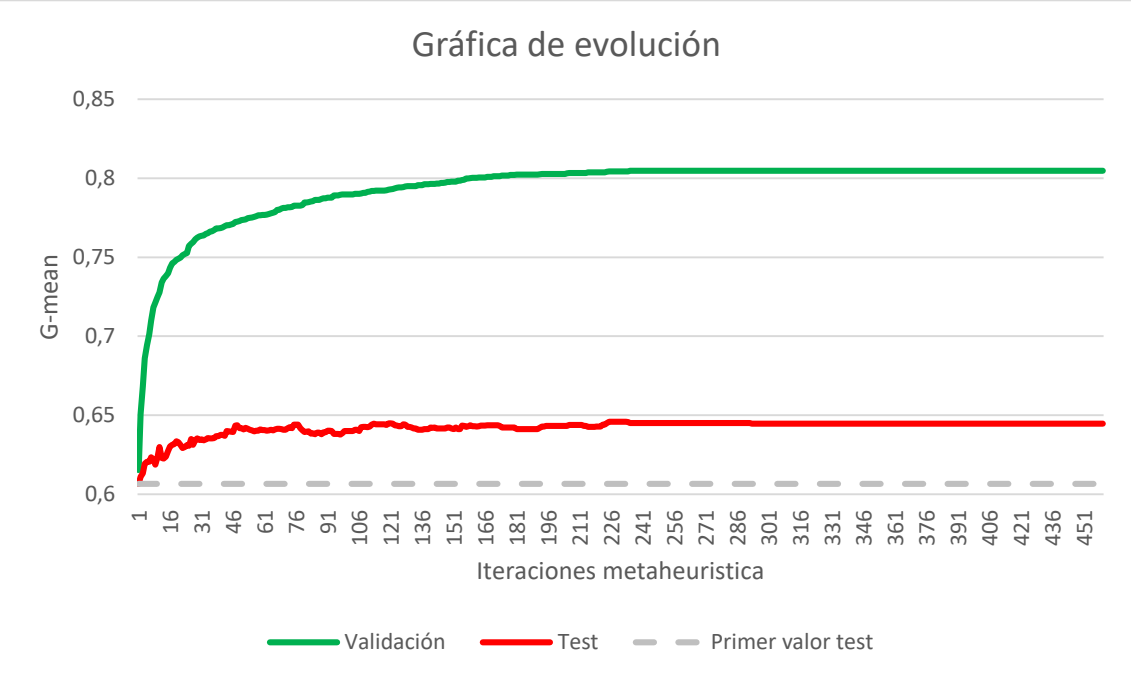

Figura A 5: Gráfica de evolución

Esta gráfica es generada a partir de los archivos *validaciones.csv* y *testeos.csv*. En ella se muestra una media de los valores alcanzados en cada ejecución del sistema, por lo que es muy buena referencia para ver si el sistema consigue una evolución del conjunto de sintéticos generado de forma satisfactoria. Si los valores de test acompañan a los de validación y se consiguen mejores resultados en comparación con el valor inicial de test, podremos considerar que se ha conseguido evolucionar exitosamente el conjunto de sintéticos generado.

Boxplot referente a árboles

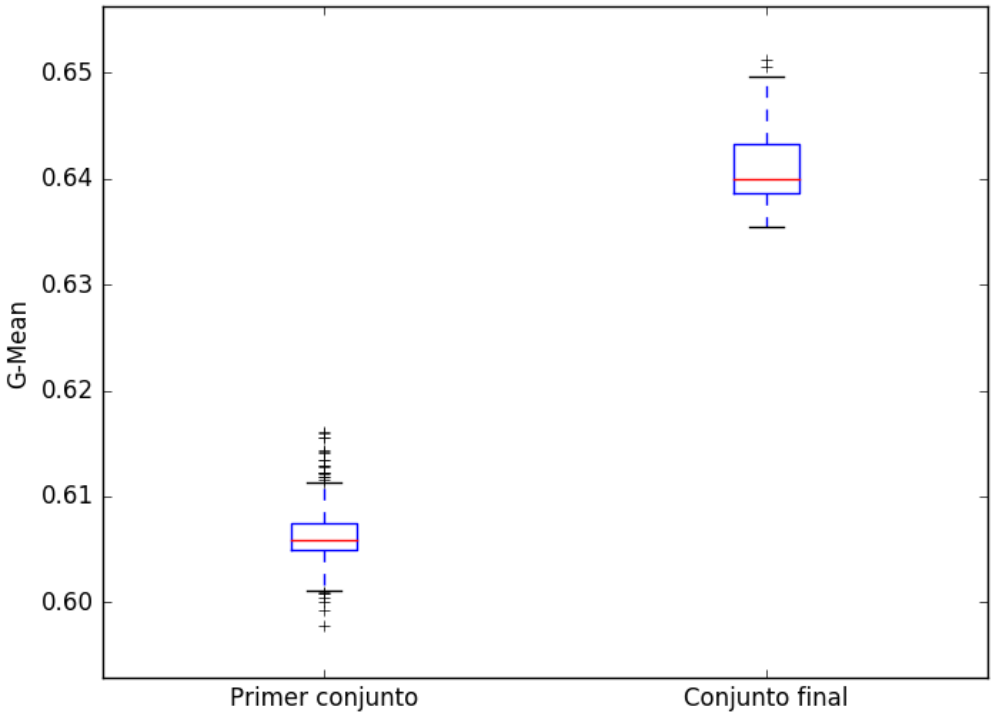

Figura A 6: Boxplot referentes a los árboles

Este gráfico se genera a partir de los archivos *betterSet_Trees.csv* y *firstSet_Trees.csv*, y en él se intenta mostrar como el conjunto de sintéticos evolucionado mediante la metaheurística *Iterated Greedy*, aporta una mayor información útil a un clasificador, siendo capaz de generar árboles que clasifiquen de forma más acertada los patrones pertenecientes al conjunto de test, y por extensión, los patrones que aparezcan posteriormente. Si el boxplot que representa al conjunto final obtiene una horquilla de valores superior a la que se consigue con el primer conjunto consideraremos este gráfico como una prueba satisfactoria del correcto funcionamiento del sistema.