Maestría en Computación Aplicada

Generación de árboles de decisión usando un algoritmo inspirado en la Física

Autor:
Camilo Flores Rodríguez

Director:
Dr. Efrén Mezura Montes
(Centro de Investigación en Inteligencia Artificial- UV)

Co-Director:
Dr. Rafael Rivera López
(Instituto Tecnológico de Veracruz)

Marzo 2021

Agradecimientos

En primer lugar quiero expresar mi agradecimiento a Dios por permitirme culminar este objetivo que me propuse en mi vida.

Agradezco especialmente a mis directores de tesis, el Dr. Efrén Mezura Montes, así como al Dr. Rafael Rivera López, por su tiempo, paciencia, apoyo y libertad que me brindaron para que este trabajo se llevara a cabo.

A mis padres, que han dejado este mundo, donde quiera que ellos se encuentren.

Así mismo, agradezco a mi esposa Italia, a mis hijos Italia Naomi e Ian Daniel por la paciencia, comprensión y solidaridad que me tuvieron durante esta travesía, por el tiempo robado a nuestra historia como familia para cumplir este sueño. A mi hermana Estela, a su esposo Roberto e hijo Alan; a mi hermana Graciela y sus hijos, que estuvieron ahí apoyando cuando más lo necesitaba.

A mis profesores, que tuvieron la gentileza de compartir sus conocimientos conmigo.

Por último, agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el apoyo económico otorgado para cursar la Maestría en Computación Aplicada (MCA) bajo el número de becario 924763.

A todos, muchas gracias.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Resumen

La inducción de árboles de decisión es un tema importante dentro del campo del aprendizaje automático, porque es muy empleada para tareas, tanto en tareas de clasificación, como de predicción. En el presente proyecto de tesis se implementa una estrategia de búsqueda global basada en el Algoritmo de Centros Evolutivo (ECA)[44] para inducir árboles de decisión paralelos a los ejes. Los nodos internos de un árbol de decisión serán codificados linealmente en un vector de valores reales, de tal manera que representen soluciones candidatas y ECA pueda evolucionarlos, usando la precisión o desempeño del árbol como medida de su valor de aptitud y obtenga árboles cercanos al óptimo. ECA creará una población aleatoria de vectores con valores reales, les aplicará sus operadores y seleccionará el mejor de ellos, para ello aplicará un proceso para mapear árboles de decisión a partir de las soluciones candidatas, usando la regla SPV (Small Position Value) [74] y el conjunto de entrenamiento. El mejor árbol de decisión en la población final será refinado, reemplazando algunos nodos hoja con subárboles para mejorar el desempeño del árbol, y posteriormente podado con el objetivo de reducir su tamaño.

Los resultados obtenidos en esta investigación han resultado ser competitivos. La mayoría de los conjuntos de datos que se han utilizado para inducirlos, mejoran tanto en el tamaño, como en el porcentaje de clasificación, pues se han comparado con el método tradicional que se tienen como referente en la literatura para la inducción j48 [79]. Así mismo, ha mejorado algunos resultados obtenidos con el método ED-ADT$_{SPV}$ publicados en [62].

1.2. Introducción

El presente documento describe elaboración del proyecto de investigación para implementar un algoritmo basado en la física llamado Algoritmo de

Centros Evolutivos (ECA, por sus siglas en ingles) para generar árboles de decisión.

Este trabajo es importante ya que utilizando una novedosa técnica de optimización genera modelos de clasificación con mejor poder predictivo y con un nivel interpretativo mayor que los generados con otras técnicas encontradas en la literatura especializada, ya que sirve en tareas de decisión donde es muy importante entender cómo se analiza la información para tomar una decisión (en medicina, por ejemplo).

ECA es un algoritmo de optimización, pertenece al área del cómputo bio-inspirado y se basa en principios físicos [44]. Utiliza el concepto de centro de masa como eje principal de su funcionamiento para determinar nuevas direcciones hacia zonas promisorias del espacio de búsqueda, tomando en cuenta los valores de la función objetivo y de esta manera mejorar la calidad de las soluciones obtenidas.

Pese a que existen métodos clásicos que han demostrado ser muy eficientes para resolver problemas de clasificación, los algoritmos bio-inspirados han probado ser una opción viable cuando se desean resolver problemas complejos. Lo anterior se debe a que poseen un diseño simple y su número de parámetros es relativamente bajo, de tal manera que facilita el procesamiento de ajuste fino cuando se resuelve un problema particular.

Los algoritmos bio-inspirados se han aplicado para resolver problemas complejos en muchas áreas del conocimiento, destacando su uso en el aprendizaje automático, donde ayudan a generar modelos predictivos como las redes neuronales y los árboles de decisión. En particular, los árboles de decisión son modelos de clasificación caracterizados por sus altos niveles de robustez e interpretabilidad. Los árboles de decisión ordinariamente se construyen con métodos recursivos, que buscan un mínimo local de acuerdo a medidas como la entropía con base en un conjunto de datos de entrenamiento. Otras soluciones, como los algoritmos evolutivos o los inspirados en la Física, utilizan estrategias de búsqueda global, también pueden generar árboles de decisión más compactos en cuanto a su tamaño y más precisos en su poder predictivo.

En la literatura se puede encontrar una amplia variedad de tipos de árboles de decisión, sin embargo, el presente trabajo se enfocará en generar árboles de decisión con particiones paralelas respecto a los ejes del espacio de los artibutos. Los árboles paralelos a los ejes (también conocidos como univariables) evalúan un solo atributo en cada condición de prueba. Por otro lado están los árboles oblicuos, estos utilizan una combinación lineal de atributos para separar el espacio de los datos de entrenamiento. Estos últimos muestran mejoras de rendimiento, además de que son más compactos [24]. Sin embargo, tienen la desventaja de que tienen un mayor costo computacional y un menor nivel interpretativo.

En la figura 1.1 se puede apreciar un árbol de decisión univariable, in-

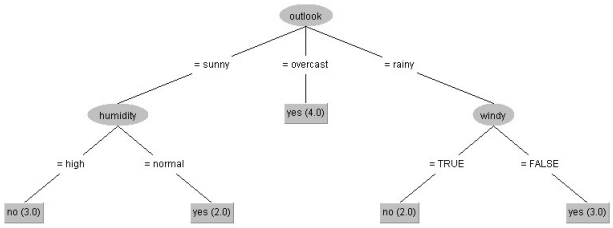

Figura 1.1: Ejemplo de un árbol de decisión univariable

ducido utilizando WEKA [3] [1] para el problema de decidir si una persona juega o no juega tenis, a partir de las condiciones del clima. En ella se puede apreciar que el atributo outlook (panorama) tiene tres valores (soleado, nublado y lluvioso). Si el panorama es nublado, entonces se puede asegurar que la persona sí jugará al tenis; pero cuando el panorama es soleado, o bien, lluvioso, se deben valorar otros atributos para tomar una decisión. Considerando que el atributo panorama tiene el valor soleado, entonces se debe seguir la rama de la izquierda, por lo que se debe tomar en cuenta el atributo humedad para poder decidir, si el valor de esta es alta, entonces se puede decir con seguridad que la persona no jugará su partido de tenis. Del mismo modo se pueden considerar las otras ramas del árbol en el análisis del mismo.

De la la figura 1.1 también se puede apreciar que se ha obtenido un árbo pequeño, por lo que el análisis del mismo ha sido fácil. Sin embargo, para problemas más complejos se podrían inducir árboles de mayor tamaño, lo cual dificulta esta característica.

El objetivo del presente trabajo es generar árboles de decisión más pequeños, y así favorecer su interpretabilidad.

1.3. Antecedentes y motivación

Con la creciente necesidad del análisis de datos en las distintas disciplinas tales como el aprendizaje automático, la ciencia de datos, el big data, entre muchas otras, y debido a la facilidad para recolectarlos, es necesario diseñar mejores métodos que generan modelos que representen con precisión las relaciones existentes entre los datos. Los problemas complejos del mundo real demandan modelos confiables y precisos que den soluciones fiables. Es por ello que se deben buscar nuevas soluciones o técnicas más efectivas que mejoren a las que ya existen y nos den mejores resultados.

[1] WEKA es una colección de algoritmos de aprendizaje automático para tareas de minería de datos. Contiene herramientas para tareas de clasificación, regresión, agrupamiento (clustering), minería de reglas de asociación y visualización [3]

Por esta razón, tener modelos de clasificación fáciles de interpretar se convierte en elemento clave para el éxito de su implementación en la solución de problemas complejos de aprendizaje automático.

Por tal motivo, se opta por combinar técnicas de Inteligencia Artificial, en este caso de técnicas de aprendizaje automático con algoritmos de cómputo bio-inspirado, que permitan generar modelos más sencillos de interpretar y más precisos.

Los árboles de decisión se utilizan para construir modelos predictivos. Su representación gráfica es una de sus principales ventajas para explicar un fenómeno, además de que pueden manejar el ruido en los datos, es decir, datos con valores atípicos, o irrelevantes, y aun así, hacer predicciones correctas.

Inducir árboles de decisión no es una tarea trivial, por lo que se han propuesto distintas técnicas que van desde aquellas basadas en estadística, hasta las que utilizan algoritmos genéticos para su construcción. El algoritmo ID3 [58] es uno de los primeros algoritmos que aparecieron para generarlos. Este algoritmo se basa en el cálculo de la información que aportan los atributos que permitan elegir el mejor atributo para crear las respectivas ramas y nodos. La idea básica de ID3 es construir árboles de decisión al emplear una técnica recursiva descendente y una búsqueda voraz sobre el conjunto de datos para probar cada atributo en cada nodo del árbol, y poder seleccionar el atributo con la mayor ganancia de información.

El algoritmo C4.5 [66] desarrollado por Quinlan en 1993 como una extensión del ID3 para datos de tipo categóricos o valores numéricos. C4.5 se basa en el criterio de proporción de la ganancia (Gain ratio) para evitar que las variables con mayor número de valores salgan beneficiadas en la selección. Al finalizar la creación de árbol, C4.5 poda el árbol generado para reducir el sobreajuste propiciado al construir el árbol con los datos de entrenamiento. Además, este algoritmo puede manejar de manera adecuada los valores faltantes, esto debido a que estos valores no se utilizan para calcular la ganancia.

El algoritmo CART (Classification and Regresion Trees) [66] genera árboles de decisión binarios, tanto para regresión como para clasificación. La idea básica del método es elegir una división, de entre varias posibles, en cada nodo de tal manera que los nodos hijo resultantes sean lo "más puros" posibles. Usa el índice de Gini como una medida de división en la selección del atributo a dividir. Este algoritmo soporta valores tanto categóricos como continuos.

Los métodos mencionados anteriormente consideran la inducción de árboles univariables, sin embargo, la técnica propuesta en el método OC1 (Oblique Classifier 1) [10], permite generar árboles de decisión multivariables o también llamados oblicuos. El método busca en cada nodo el mejor hiperplano para particionar el espacio del conjunto de entrenamiento dado en regiones homogéneas, usando para ello tanto un método determinista y uno

aleatorio para tratar de evitar que el algoritmo quede atrapado en un mínimo local. Sin embargo, todos estos algoritmos buscan soluciones localmente. Por otro lado, también se han ocupado algoritmos de búsqueda global, tales como los algoritmos evolutivos, para inducir árboles de decisión, y que han dado muy buenos resultados. Como ejemplo se puede mencionar el algoritmo evolución diferencial para generar árboles de decisión (ED-ADT$_{SPV}$) [61]. Como resultado de aplicar este algoritmo, se obtienen árboles de decisión de la misma calidad para que los que se obtienen con los métodos tradicionales, pero más pequeños, lo que los hace mucho más fáciles de interpretar.

Así como existen algoritmos inspirados en metáforas evolutivas, también existen aquellos los que se inspiran en fenómenos físicos, de los cuales no se han estudiado sus capacidades para generar árboles de decisión. El presente trabajo de tesis tiene como eje el Algoritmo de Centros Evolutivos (ECA) [44], el cual es inspirado en un concepto físico llamado centro de masa, que es utilizado para crear nuevas direcciones y mover las soluciones con menor calidad en la población hacia mejores regiones basados en los valores de la función objetivo y de esta manera mejorar la calidad de las soluciones.

Con base en un estudio previo [62] donde se aplica el algoritmo de evolución diferencial para inducir árboles de decisión, el presente proyecto de tesis busca explorar las capacidades de ECA para implementar una búsqueda global en el espacio de clasificadores y construir árboles de decisión univariables y evaluar su desempeño. En otras palabras, se implementa ECA para inducir árboles de decisión, y valorar si se pueden generar árboles compactos, igual o mejores que los producidos por el algoritmo de evolución diferencial. La motivación para ocupar ECA se debe a que ha mostrado un desempeño superior en problemas de optimización numérica, y se busca trasladar ese buen desempeño ahora a la inducción de árboles de decisión.

Se ha seleccionado el estudio de los árboles paralelos a los ejes, debido a que las técnicas tradicionales para inducir estos clasificadores se basa en criterios de búsqueda local, y existe trabajo desarrollado utilizando generalmente algoritmos genéticos, por lo que es interesante estudiar la generación de este tipo de clasificadores con nuevas técnicas de cómputo evolutivo que han demostrado proveer de soluciones competitivas a problemas complejos [44].

Al generar un sistema híbrido inteligente se busca precisamente mejorar la calidad y confiabilidad de modelos que permitan clasificar eficientemente a partir de un conjunto de datos.

1.4. Objetivos de la investigación

El objetivo general del presente trabajo es implementar ECA para inducir árboles de decisión paralelos a los ejes.

Para lograr este objetivo se plantean los siguientes objetivos específicos.

- Determinar las ventajas de utilizar un enfoque evolutivo sobre las técnicas tradicionales para inducir árboles de decisión.
- Evaluar el comportamiento de búsqueda en el espacio de los árboles de decisión de ECA.
- Comparar el desempeño del enfoque propuesto contra los resultados obtenidos con algunas técnicas tradicionales y contra el algoritmo basado en evolución diferencial

1.5. Hipótesis

Al aplicar el Algoritmo de Centros Evolutivos (ECA) para inducir árboles de decisión, se generarán modelos con similar precisión de clasificación, pero con menor complejidad (más pequeños) que los generados con técnicas tradicionales y con el algoritmo evolución diferencial.

1.6. Contribuciones

La principal contribución del presente trabajo de tesis es en el campo de minería de datos y el aprendizaje automático, al introducir un enfoque basado en una metáfora de la Física, de manera más específica basado en ECA.

En esta tesis se implementa ECA para generar árboles de decisión, pero sólo considerando árboles de decisión paralelos a los ejes. Del mismo modo, se realizan experimentos y pruebas con conjuntos de datos del sitio de la Universidad de California en Irvine (UCI)[16].

El proyecto no contempla datos reales, por lo que el pre-procesamiento de datos no está en el ámbito de este trabajo.

1.7. Organización

Este documento está organizado como sigue: el presente capítulo presenta una introducción y se describen tanto la hipótesis, así como los objetivos que se persiguen en la investigación. El capítulo 2 presenta las bases claves para desarrollar el presente trabajo de investigación, se ofrece un panorama general sobre el estado del arte de los árboles de decisión y los métodos que se han utilizado para inducirlos, se revisa los conceptos claves de los algoritmos evolutivos, y se describe detalladamente el algoritmo ECA sobre el que gira el presente trabajo de investigación. El capítulo 3 se describe la implementación de ECA para inducir árboles de decisión. El capítulo 4 presenta el conjunto de pruebas realizadas, y muestra los resultados de las comparaciones con otros algoritmos clásicos. Finalmente, las conclusiones y trabajos futuros son descritas en el capítulo 5.

Capítulo 2

Marco Teórico

2.1. Técnicas de clasificación en aprendizaje automático

2.1.1. Aprendizaje automático

Cada segundo que pasa, la cantidad de datos en el mundo crece de manera exponencial. Gracias a las computadoras, al internet y a los dispositivos móviles, los procesos para su obtención y almacenamiento se han convertido en algo muy sencillo, y una tarea común para todas las compañías tanto públicas como privadas, así como para el área científica.

Los datos contienen información oculta potencialmente útil que rara vez se hace explícita o se aprovecha [78]. Con el uso de la minería de datos y técnicas de aprendizaje automático se busca encontrar patrones en los datos, es decir, se intenta buscar algún patrón en la información que se encuentra almacenada en las bases de datos de las corporaciones, de tal manera que las personas puedan solucionar problemas al tratar analíticamente los datos y tengan herramientas que les permitan tomar mejores decisiones.

El aprendizaje automático es una disciplina de la Inteligencia Artificial cuyo objetivo principal es que un agente artificial mejore su rendimiento (aprenda) a partir de resultados previamente obtenidos. Desde el punto de vista del aprendizaje automático, obtener conocimiento implica extraer información potencialmente útil y desconocida a partir de un conjunto de datos. Para lograrlo, es necesario generar métodos, que generalmente son no-triviales, que nos permitan analizar los datos, y a partir de dicho análisis producir un modelo mediante el cual se pueda representar un fenómeno. El aprendizaje automático nos proporciona procedimientos para construir esos modelos, ya que se basa en el aprendizaje inductivo [61].

2.1.2. Técnicas de clasificación

La tarea de extraer información que sea útil para el ser humano se le conoce como reconocimiento de patrones [78] y engloba un amplio rango de problemas de gran importancia que puede ir desde el reconocimiento del lenguaje natural, el reconocimiento de imágenes o hasta el diagnóstico de una enfermedad, entre otros. El aprendizaje automático es una disciplina de la inteligencia artificial que proporciona las técnicas adecuadas para lograr este objetivo.

Los métodos para el reconocimiento de patrones han ganado importancia en los últimos años gracias a la gran facilidad que existe para la obtención de los datos por medio de dispositivos electrónicos, al internet, entre otros factores, y al crecimiento de la demanda del análisis de datos en las distintas disciplinas tanto científicas como gubernamentales o privadas.

En aprendizaje automático se pueden diferenciar dos tipos de métodos que se utilizan para la construcción de los modelos previamente mencionados: aprendizaje supervisado y no supervisado. En aplicaciones en las cuales el conjunto de datos del fenómeno estudiado comprende datos etiquetados se dice que es un problema de aprendizaje supervisado [10]. En aprendizaje no supervisado, los conjuntos de datos no poseen datos etiquetados y el problema principal es descubrir grupos (clúster) con elementos que compartan similitudes.

Existen diversas técnicas que permiten obtener modelos que representen el patrón obtenido a partir de los conjuntos de datos. Los árboles de decisión, las redes neuronales y las máquinas de soporte vectorial han sido ampliamente utilizados para para construir modelos predictivos. Por otro lado, los métodos de clústering han sido aplicados para construir métodos descriptivos [61].

El presente trabajo de tesis se ubica en el aprendizaje supervisado. En aprendizaje supervisado existen dos técnicas principales: clasificación y regresión. La mayoría de aplicaciones de propósito general involucran clasificación [58]. Una tarea de clasificación consiste en tomar un vector de n datos de entrada $x = (x_1, x_2, x_3 \ldots, x_n)$ y asignarlo a una de las K clases discretas (etiquetas de clase) que representan la membresía de cada instancia a cada clase $C = (c_1, c_2, c_3, \ldots, c_k)$. En el escenario más común las clases se consideran disjuntas, de tal manera que cada entrada es asignada a una y solo una clase. El espacio de la entrada es de esta manera dividido en regiones de decisión cuyos límites son llamados límites de decisión o superficies de decisión [10]. Cada ejemplo de entrenamiento (instancia o caso de estudio) está compuesto de una colección de valores de atributos y una etiqueta de clase.

Esta tarea se puede dividir en dos etapas, primero, se usa parte del conjunto de datos, para construir el modelo de entrenamiento o aprendizaje; y en una segunda etapa, el modelo construido se usa en la otra parte del

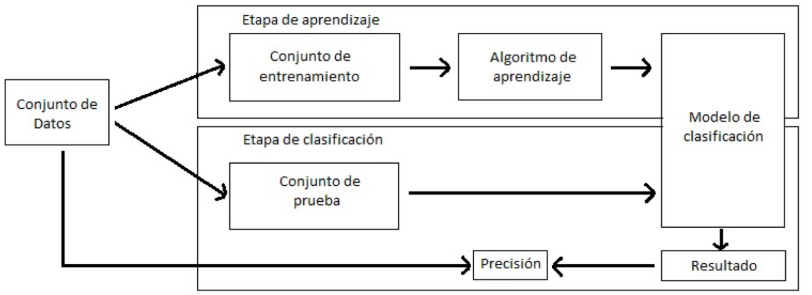

Figura 2.1: Proceso de clasificación [61]

conjunto de datos para predecir la membresía de clase de nuevos ejemplos no clasificados (clasificación) [38], como se puede apreciar en la figura 2.1. En la literatura podemos encontrar diferentes técnicas de clasificación, sin embargo, las más ampliamente usadas son:

Redes Neuronales Artificiales (ANNs, por sus siglas en inglés): Son sistemas de procesamiento adaptativos artificiales inspirados en el modelo biológico neuronal del ser humano, capaces de modificar su estructura interna en relación con una función objetivo [60]. Estas redes neuronales artificiales son un conjunto de unidades de entrada-salida en las cuales cada conexión tiene asociado un peso. Durante la fase de entrenamiento, la red aprende al ajustar los pesos para poder predecir la etiqueta de la clase correcta de los datos de entrada [60]. Las redes neuronales son métodos aplicados para resolver problemas complejos que por su funcionamiento se les conoce como modelos de cajas negras [52], debido principalmente a su escasa interpretabilidad porque proporcionan poca información sobre el significado simbólico detrás de los pesos aprendidos de las capas ocultas de la red [75], esto las hace candidatas poco idóneas para utilizarlas en minería de datos, considerando que la interpretabilidad del modelo es sumamente importante [31].

Entre sus ventajas se incluye su alta tolerancia a datos faltantes o con ruido, así como su habilidad para clasificar patrones en los cuales ellas no han sido entrenadas. Se pueden usar cuando se tenga poco conocimiento sobre las relaciones que existen entre los atributos y las clases. Son muy adecuadas para tratar con entradas que tengan valores continuos, a diferencia de otras técnicas de clasificación. Han tenido éxito sobre una amplia gama de datos del mundo real, incluyendo reconocimiento de caracteres escritos, patología, entre otros. Los algoritmos de redes neuronales son inherentemente paralelos, ya que se pueden usar técnicas de cómputo paralelo para acelerar el proceso de cálculo, además de que se han desarrollado técnicas para la extracción de reglas de las redes neuronales entrenadas [60].

Máquinas de Soporte Vectorial (SVM): Es uno de los algoritmos de aprendizaje automático supervisado que se ha empleado especialmente

en problemas de reconocimiento de patrones, tales como reconocimiento de imágenes, reconocimiento de lenguaje natural, detección de rostros, entre otros. Son una mezcla de modelado lineal y aprendizaje basado en instancias [78] que se utiliza para tareas de clasificación de datos lineales y no lineales [60].

Los patrones pueden ser lineales, estos se pueden clasificar fácilmente en dimensiones bajas, es decir, se puede usar una línea recta que permita separar los puntos que representan los datos en una o dos dimensiones. Sin embargo, también pueden ser no lineales, este tipo de patrones es más difícil separarlos con una línea recta, incluso las dimensiones necesarias para su representación pueden ser muy grandes.

Para este último tipo de patrones, predecir la etiqueta a la que pertenece cada uno no es una tarea trivial. Para resolver este tipo de problemas, las máquinas de soporte vectorial utilizan técnicas basadas en vectores de soporte y márgenes [52], y en maximizar el margen que existe entre estos vectores y un hiperplano utilizado para representar la frontera entre una clase y otra.

La idea central detrás de las máquinas de soporte vectorial es la construcción de un hiperplano cercano al óptimo, que puede ser usado para clasificar o separar las distintas instancias o patrones. El objetivo es obtener aquel hiperplano que maximice el margen, de tal manera que clasifique correctamente las instancias.

Cuando los patrones se encuentran altamente mezclados, es claro que la clasificación se convierte en una tarea difícil. Para resolver este problema este algoritmo utiliza un mapeo no lineal para transformar el dato de entrenamiento en una alta dimensión para que en esta nueva dimensión busque un hiperplano óptimo que permita separar los datos, que calcula márgenes de separación entre los datos por medio de vectores de soporte, para ello utiliza un método basado en kernel [55].

Clasificadores bayesianos: Los clasificadores bayesianos utilizan métodos estadísticos para realizar tareas de clasificación, más específicamente, se basan en el teorema de Bayes para realizarlas. Estos pueden predecir la probabilidad que tiene una instancia de pertenecer a una determinada clase en particular. El clasificador Naïve Bayes es el más simple dentro de este tipo de clasificadores, ya que para hacer los cálculos de las probabilidades más sencillos, el clasificador asume que los valores de los atributos son independientes entre ellos (a esta suposición se le conoce como independencia condicional de clase), de ahí el término naïve o ingenuo. Si se tiene un conjunto de ejemplos de entrenamiento, el objetivo del clasificador es determinar la probabilidad $(C_i|X)$ de que el ejemplo x pertenezca a la clase c_i, es decir, se predice que x pertenece a la clase c_i, si y solo si.

$$P(C_i \mid X) > P(C_j \mid X) \; para \; 1 \leq j \leq k, \; j \neq i \tag{2.1}$$

De esta manera se define la clase que maximiza $P(C_i \mid X)$.

Por otro lado, también hay modelos gráficos probabilísticos conocidos como Redes Bayesianas que permiten la representación de dependencias entre los subconjuntos de atributos [61]. Este tipo de modelo permite representar distribuciones de probabilidad de manera gráfica como una red de nodos, uno por cada atributo, conectado por ejes direccionados que especifican la probabilidad condicional de cada atributo dado por sus padres. Una característica importante, es que son grafos dirigidos acíclicos [25]. Estos modelos se utilizan para representar sistemas multivariados orientados a la clasificación, el diagnóstico y la toma de decisiones [57].

Árboles de decisión: Los árboles de decisión son una estructura jerárquica compuesta por un conjunto de nodos internos y nodos hoja [61] que clasifican un conjunto de datos, al clasificarlos desde el nodo raíz hasta algún nodo hoja. Cada nodo interno en el árbol especifica una condición de prueba que evalúa uno o varios atributos, y cada rama descendiente del árbol representa una secuencia de decisiones hechas por el modelo para determinar la membresía de la clase de un nuevo ejemplo no clasificado [45].

A diferencia de otros modelos de clasificación considerados como cajas negras, los árboles de decisión son modelos de caja blanca que permiten ver de manera clara por qué el modelo clasifica de una forma u otra, o argumentar dicha clasificación. Este tipo de clasificadores es el tema central de la presente tesis y se describen de manera detallada en la siguiente sección.

2.2. Árboles de decisión

2.2.1. Conceptos y tipos de árboles de decisión

Formalmente, un árbol de decisión consiste de nodos que forman un árbol dirigido con un nodo raíz que no tienen un nodo predecesor. Todos los demás nodos tienen exactamente un nodo predecesor. Un nodo con arcos de salida se conocen como nodo interno o nodo prueba, mientras que los nodos que no poseen nodos sucesores son nodos hoja o nodos de decisión [63]. Entonces, matemáticamente, un árbol de decisión es un grafo acíclico dirigido con un nodo raíz $A = (G(V, E), v_1)$, donde V es un conjunto de nodos, E es el conjunto de enlaces que unen pares de nodos de V, y v_1 es el nodo raíz [18].

El *aprendizaje o la inducción de árboles* de decisión se realiza a partir de un conjunto de datos mediante un algoritmo que permite obtener un modelo que intenta descubrir la relación que existe entre los atributos de entrada y la clase objetivo. Al aplicarlo a nuevos ejemplos éste es capaz de predecir el valor de la clase objetivo y de esta manera clasificarlos correctamente. A este conjunto de datos que se utiliza para entrenar u obtener el modelo se le llama *conjunto de entrenamiento*.

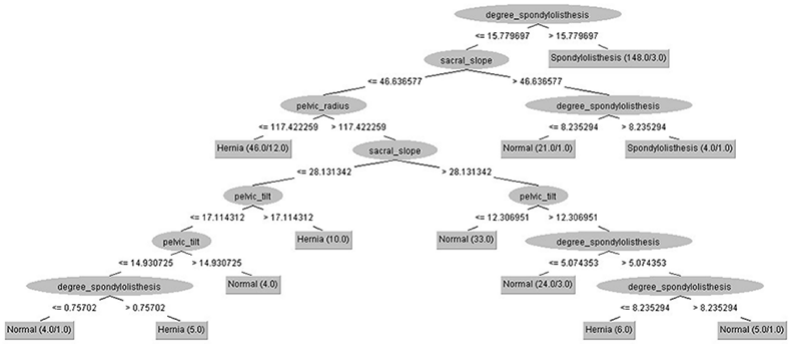

Figura 2.2: Árbol de decisión univariable para clasificar pacientes como normales o sanos

Los atributos pueden ser de dos tipos: *nominal o numérico*. Cuando un atributo es nominal es útil denotarlo como $N = \{v_{i,1},\ v_{i,2}, v_{i,3},\ ...,\ v_{i\ |dom(a_i)|}\}$ éste es el dominio de todos los valores posibles que pueda tener. Mientras que $dom(c) = \{c_{i,1},\ \ c_{i,2},\ c_{i,3},\ ...,\ c_{i\ |dom(c_i)|}\}$ representa el dominio de las clases objetivo.

Al conjunto de todos los ejemplos posibles es llamado el *espacio de ejemplos* y es definido como el producto cartesiano de todos los atributos de entrada $X = \{dom(a_1)$ x $dom(a_1)$ x ... x $dom(a_n)\}$ [63].

El *conjunto de entrenamiento* consiste de un conjunto de m tuplas, cada una es descrita por un vector de valores de atributos. Formalmente, el conjunto de entrenamiento es representado por $S(R) = (< x_1, y_1 >, \ldots, < x_m, y_m >)$ donde $x_i \in X$ y $y_i \in dom(c)$.

La figura 2.2 muestra un ejemplo de un árbol de decisión univariable inducido con el algoritmo de clasificación j48 en la herramienta WEKA [3], a partir de un conjunto de datos biomédicos obtenidos del sitio de la Universidad de California en Irvine (UCI) [16], con seis características biomédicas de la columna vertebral, utilizadas para clasificar a los pacientes ortopédicos si son diagnosticados como pacientes normales (sanos) o con hernia discal.

A diferencia de otras técnicas de clasificación, los árboles de decisión ofrecen una alta interpretabilidad gracias a su "interfaz" gráfica. En comparación con otras técnicas de caja negra, los árboles de decisión son muy transparentes, por simple inspección se puede argumentar el por qué el modelo clasifica en una clase u otra. Estos modelos se pueden utilizar para resolver problemas complejos [58].

En función del número de atributos que son evaluados en cada condición de prueba que se realiza en un nodo se pueden inducir dos tipos de árboles de decisión: árboles univariables y árboles multivariables [61]. En un árbol de decisión univariable en cada nodo de decisión considera solo el valor de un atributo o variable para que a partir de éste se divida el conjunto de

entrenamiento, mientras que en un árbol de decisión multivariable en cada nodo de decisión se divide el espacio de entrada en dos con un hiperplano arbitrario que genera a divisiones oblicuas [80].

Los árboles de decisión univariables son los más conocidos y los métodos para inducirlos son más sencillos. En cada nodo, solo un atributo es usado, si éste es continuo entonces la decisión es de la forma:

$$a_i > t_j \tag{2.2}$$

haciendo una división binaria donde a_i es el i-ésimo atributo de entrada y t_j es una constante o umbral elegido adecuadamente y representa el punto de corte del a_i atributo. Si a_i posee un valor discreto dentro de un conjunto de valores nominales con m valores, entonces en el nodo se realiza una división de m-aria.

Por otro lado, en los árboles de decisión multivariables, en cada nodo la decisión se debe realizar tomando en cuenta más de un atributo. Para este tipo de árboles, los algoritmos que se utilizan para inducirlos no seleccionan el mejor atributo como en el caso de los univariables, sino que utilizan la combinación lineal de los atributos. En este caso las decisiones que se realizan en cada nodo toman la forma:

$$\sum_{i=1}^{n} x_i a_i > t_0 \tag{2.3}$$

donde w_i son los pesos, y t_0 es un umbral. Este tipo de árboles se utiliza cuando existen en el conjunto de entrenamiento una distribución de datos compleja [80].

Los árboles univariables también se conocen como *árboles paralelos al eje*, esto es debido a que en cada nodo interno las condiciones de prueba representan hiperplanos paralelos a los ejes, tal como se puede apreciar en la figura 2.3a), donde cada hiperplano es paralelo ya sea al eje Y o al eje X, y de esta manera cada uno divide o separa el espacio de los ejemplos en dos regiones. Por otro lado, los árboles multivariables también se les llama oblicuos, ya que las pruebas que realizan en cada nodo son equivalentes a hiperplanos en una orientación oblicua a los ejes del espacio de atributos, figura 2.3b) [35].

2.2.2. Técnicas clásicas para inducción de árboles de decisión

Se han desarrollado diferentes técnicas que se pueden considerar clásicas y estándares para inducir árboles de decisión en la comunidad de aprendizaje automático.

Algoritmo ID3. Uno de los trabajos pioneros fue el de Quinlan con el algoritmo ID3 [58]. Este algoritmo induce árboles de decisión basados en la ganancia de información obtenida de los ejemplos de entrenamiento

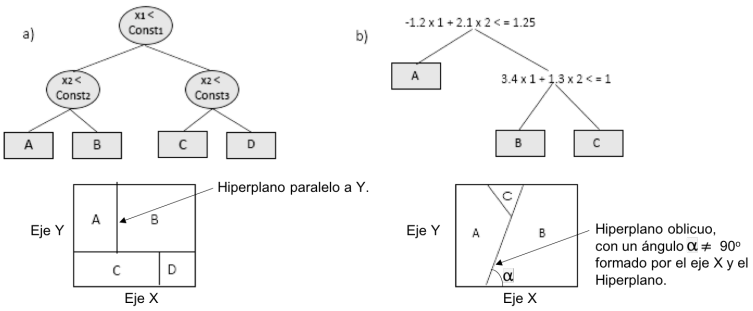

Figura 2.3: Árboles a) paralelos al eje y b) oblicuos [5]

y luego lo usa para clasificar el conjunto de prueba. El conjunto de datos generalmente posee atributos nominales para realizar la tarea de clasificación con valores no faltantes.

De manera general, si se proporciona una distribución de probabilidad $P = (p_1,\ p_2,\ p_3, ..., p_n)$ y un ejemplo S, luego la información que lleva esta distribución, se le conoce como *entropía* de P, y es calculada por:

$$EntropiaP = -\sum_{i=1}^{n} p_i \cdot log(p_i) \tag{2.4}$$

Si se tienen funciones que permiten medir el grado de mezcla de las clases para todos los ejemplos y por lo tanto cualquier posición del árbol en construcción. Se debe tener una nueva función que permita seleccionar el atributo que debe etiquetar el nodo actual. Esto define la ganancia para una prueba T y una posición p:

$$Ganancia(p, T) = EntropiaP - \sum_{j=1}^{n} (p_j \cdot Entropia(p_j)) \tag{2.5}$$

Donde los valores (p_j) es el conjunto de todos los valores posibles para el atributo T. De esta manera se puede usar esta medida para poder determinar que atributo es mejor y construir el árbol de decisión donde en cada nodo se encuentra el atributo con la más alta ganancia de información de todos los atributos que aún no se han considerado en la ruta desde el nodo raíz [33].

En el **Algoritmo** 1 se muestra el pseudocódigo del algoritmo ID3, para un conjunto de datos X, con un número n de atributos $(x_1,\ x_2,\ x_3,\ ...,\ x_n)$, y un atributo categórico c.

Algoritmo C4.5. El algoritmo C4.5 es una extensión del algoritmo ID3 que el mismo Quinlan introdujo en 1993 [66] para mejorar algunas deficiencias que ID3 tiene. Ya que no estaba pensado para atributos numéricos y no usa poda para reducir el sobre-entrenamiento. Para solucionar este tipo de inconvenientes, en algoritmo C4.5 se utiliza un nuevo cálculo que permite

Algoritmo 1 Pseudocódigo del algoritmo ID3 [33]

procedimiento ID3(R, C, S) ▷ Entrada: R un conjunto de atributos no objetivo; C, el atributo objetivo; S, dato de entrenamiento. Salida: árbol de decisión
  si $S = \emptyset$ **entonces**
    regresar $\emptyset$ ▷ Regresar conjunto vacío
  fin si
  si S tiene valores con el mismo valor objetivo **entonces**
    regresar un único nodo con este valor
  fin si
  $D \leftarrow Calcular\ la\ Ganancia(D, S) con\ 2.5$
  $\{d_j j = 1, 2, 3..., m\} \leftarrow valores\ de\ atributo\ de\ D$
  $\{S_j\ con\ j = 1, 2, 3..., m\} \leftarrow$ *los subconjuntos de E construído de* d_j *registrar el valor del atributo D*
  regresar Un árbol cuya raíz es D y los arcos son etiquetados por d_1, d_2, $\cdots, d_m$. E ir a los sub-árboles ID3$(R - \{D\}, C, S_1), ID3(R - \{D\}, C, S_2), \cdots, ID3(R - \{D\}, C, S_m)$
fin procedimiento

medir una razón de ganancia. Gracias a la introducción de este nuevo cálculo se permite medir una relación de ganancia:

$$RelGan(p, T) = \frac{Gan(p, T)}{infDiv(p, T)} \tag{2.6}$$

Donde:

$$infDiv(p, test) = -\sum_{j=1}^{k} p(\frac{j}{p}) \cdot log(p'(\frac{j}{p})) \tag{2.7}$$

$p'(\frac{j}{p})$ es la proporción de elementos presentes en la posición p, tomando el valor de la j-ésima prueba. A diferencia de la entropía que es utilizada en ID3, la relación de ganancia es independiente de la distribución que tengan los objetos en las diferentes clases.

C4.5 maneja de manera más efectiva los atributos con valores desconocidos gracias al evaluar la ganancia o la relación de ganancia para este tipo de atributos considerando únicamente los registros para los cuales este registro está definido. Para hacerlo, el algoritmo estima las probabilidades de salidas diferentes, entonces el nuevo criterio de ganancia toma la forma:

$$Gan(p) = F(Info(T) - Info(p, T)) \tag{2.8}$$

Donde F es el número de ejemplos en el conjunto de datos con valores conocidos para un número total de ejemplos dados en un conjunto de datos del atributo.

$$Info(T) = \sum_{i=1}^{n}((p_j)Entropia(p_j)) \tag{2.9}$$

Así mismo, también trata los atributos con valores continuos. Sea R_i un atributo con valor continuo en un intervalo de valores continuos. Se examinan los valores de esos atributos en el conjunto de entrenamiento. Se ordenan esos valores de forma ascendente a $v_1, v_2, v_3, \ldots, v_n$, posteriormente para cada uno de esos valores la partición entre registros que tienen valores de C, menor o igual que los valores de v_j. La ganancia para cada una de esas particiones es calculada y la partición que maximiza la ganancia es seleccionada.

Por último, C4.5 utiliza una técnica de poda de tal manera que se reduzca la tasa de error. Esta técnica reduce el tamaño del árbol al quitar secciones que pueden estar basadas en datos erróneos o faltantes, de esta manera se reduce la complejidad del árbol y se mejora su poder de clasificación.

CART (Classification and Regression Trees) [12]. A diferencia de las técnicas de inducción de árboles de decisión analizadas hasta ahora que son para generar árboles n-arios, el método CART genera árboles de decisión binarios. La idea básica del método es elegir una división, de entre varias posibles, en cada nodo de tal manera que los nodos hijo resultantes sean lo "más puros" posibles.

Sea $A = \{x_1, x_2, x_3, \cdots, x_n\}$ el conjunto de atributos o variables predictoras. Si A es categórica, de n número de categorías, entonces hay $2^{n-1} - 1$ posibles divisiones para este nodo predictor. Por otro lado, si A es de tipo continuo con n diferentes valores, entonces hay $n - 1$ diferentes divisiones.

Rutkowski *et al.* [65] describen el método CART con el siguiente procedimiento. Inicialmente se crea el nodo raíz al que llaman L_0. En el proceso para inducir los nodos del árbol L_q, incluido el nodo raíz, se procesa un subconjunto S_q del conjunto de entrenamiento X. Si todos los elementos del conjunto S_q son de la misma clase entonces el nodo es etiquetado como nodo hoja y la división no es llevada a cabo. En caso contrario, en función del criterio de medición de la división el mejor atributo es elegido de entre los atributos disponibles en el nodo que se esté considerando. Para cada atributo disponible x_i, el conjunto de los valores que los atributos pueden tomar A^i es dividido en dos subconjuntos disjuntos A^i_I y A^i_D, así, $A^i = A^i_I \cup A^i_D$. Una vez que se haya elegido A^i_I automáticamente se determina el subconjunto complementario A^i_D, y la partición está representada en relación únicamente por A^i_I. El conjunto de todas las divisiones posibles del conjunto A^i se denota por V_i. Como resultado, los subconjuntos A^i_I y A^i_D dividen el conjunto de entrenamiento S_q en dos subconjuntos disjuntos: izquierdo $I_q(A^i_I)$ y derecho $D_q(A^i_D)$.

Luego:

$$I_q(A^i_I) = \left\{x_j \in S_q \mid v^i_j \in A^i_I\right\} \tag{2.10}$$

$$D_q(A_I^i) = \left\{x_j \in S_q \mid v_j^i \in A_D^i\right\} \tag{2.11}$$

Los conjuntos $I_q(A_I^i)$ y $D_q(A_I^i)$ dependen de los atributos seleccionados y las particiones de sus valores. $p_{I,q}(A_I^i)(p_{D,\ q}(A_I^i))$ denota la fracción de los elementos del dato X_q el cual pertenece al subconjunto $I_q(A_D^i)(D_q(A_I^i))$. Las fracciones $p_{I,q}(A_I^i)$ y $p_{D,q}(A_I^i)$ son dependientes

$$p_{I,q}(A_I^i) \ = \ 1 \ - \ p_{I,q}(A_I^i) \tag{2.12}$$

Únicamente uno de esos parámetros es necesario para ser considerado, es decir, $p_{I,q}(A_I^i)$. La fracción de los elementos de $I_q(A_D^i)(D_q(A_I^i))$, de la clase k, es denotado por $p_{kI,q}(A_I^i)(p_{kD,\ q}(A_I^i))$. La fracción de todos los elementos S_q en el nodo considerado N_q, de la clase k, es denotado por $p_{k,q}$, donde $k = 1, 2, 3, \cdots, K$, no son dependientes en el atributo elegido a^i y la división A_I^i. Ahora bien, la medida de impureza usada por CART es el índice de Gini. Para cualquier subconjunto X_q del conjunto de entrenamiento es dado por:

$$Gini\ (X_q) \ = \ 1 \ - \ \sum_{k=1}^{K} (p_{k,q})^2 \tag{2.13}$$

El índice de Gini ponderado de los subconjuntos S_q resultan de la selección de la partición A_I^i es definido por:

$$wGini(S_q,\ A_I^i) \ = \ p_{I,\ q}(A_I^i)Gini(I_q(A_I^i)) \ + \ (1 - p_{I,\ q}(A_I^i)Gini(D_q(A_I^i)) \tag{2.14}$$

donde los índices de Gini de los conjuntos $I_q(A_I^i) y D(A_I^i)$ son dados análogamente por:

$$Gini\ (I_q(A_I^i)\ ) \ = \ 1 \ - \ \sum_{k=1}^{K} (p_{kI,q}(A_I^i))^2 \tag{2.15}$$

$$Gini\ (D_q(A_I^i)\ ) \ = \ 1 \ - \ \sum_{k=1}^{K} (p_{kD,q}(A_I^i))^2 \tag{2.16}$$

Una función de medida de división en el algoritmo CART es definido como la diferencia entre el índice Gini [2.13] y el índice Gini ponderado proporcionado por la ecuación 2.14. Por lo tanto, esta ganancia de Gini se define como:

$$g(S_q,\ A_I^i) \ = \ Gini(S_q) \ - \ wGini(S_q,\ A_I^i) \tag{2.17}$$

De todas las posibles particiones de A_I^i del conjunto A_i, el único que maximiza el valor de la ganancia de Gini es seleccionado por:

$$\dot{A}^i_{I,q} = arg\ max_{A^i_I \in V_i} \{g(S_q, A^i_I)\} \tag{2.18}$$

El Algoritmo 2 muestra en pseudocódigo el procedimiento del algoritmo CART.

Algoritmo 2 Pseudocódigo del algoritmo CART [67]

1: **procedimiento** CART
2: Iniciar con el nodo raíz ($n = 1$)
3: Buscar una división s^* en el conjunto de posibles candidatos s que proporciona la menor impureza
4: Dividir el nodo 1 ($n = 1$) en dos nodos ($n = 2, n = 3$) usando la división s^*
5: Repetir el proceso de búsqueda de divisiones con los nodos ($n = 2, n = 3$) como se indica del paso 1 al paso 3, hasta que el árbol crezca y se cumplan ciertas reglas
6: **fin procedimiento**

OC1. En 1993, Murthy et al. [47] introducen un método llamado Oblique Classifier 1 (OC1) que como su nombre indica permite inducir árboles de decisión oblicuos. A diferencia de las técnicas C4.5 y CART descritas anteriormente, el método OC1 evalúa múltiples variables y no impone restricciones a los hiperplanos considerados, utiliza por lo tanto hiperplanos oblicuos, y por esta razón es que al producto de inducción obtenido se le llama *árbol oblicuo*.

El método busca en cada nodo el mejor hiperplano para particionar el espacio del conjunto de entrenamiento dado en regiones homogéneas, usando para ello el método determinista Hill climbing, sin embargo, los autores justifican el uso de aleatoriedad, sobre todo al inicio del algoritmo para colocar un hiperplano aleatorio, que posteriormente será perturbado para colocarlo en nuevas ubicaciones y para resolver el problema de los mínimos locales. Así mismo, esta característica de aleatoriedad permite al algoritmo producir varios árboles diferentes para el mismo conjunto de datos de entrenamiento, por lo que de manera simultánea se están creando una familia de algoritmos clasificadores: los algoritmos de k-arboles de decisión, en los cuales un ejemplo es clasificado por el voto mayoritario de k árboles.

El método OC1 de inducción de árboles de decisión oblicuos es el siguiente:

- Dado un conjunto de entrenamiento de n ejemplos en d dimensiones. Se coloca un hiperplano aleatorio inicial en cada nodo del árbol.
- Se busca el mejor hiperplano dentro del espacio de posibles hiperplanos, por medio de un procedimiento de perturbación del hiperplano actual se divide el espacio de ejemplos P en un nodo del árbol. Por lo

tanto, P contiene n ejemplos, cada uno con n atributos, donde cada ejemplo pertenece a una categoría particular. La ecuación del hiperplano actual es por lo tanto:

$$H = \sum_{i=1}^{d} (a_i X_i) + a_{d+1} = 0 \tag{2.19}$$

Donde a_i son los coeficientes de la ecuación lineal que representa el hiperplano. Sea $P_j = (x_{j1}, x_{j2}, x_{j3}, ..., x_{jd})$ el j-ésimo ejemplo del espacio de ejemplos P. Sustituyendo P_j en la ecuación para H se tiene:

$$\sum_{i=1}^{d} (a_i \; x_{ji}) + a_{d+1} = V_j \tag{2.20}$$

Donde el signo de V_j determina si el punto P_j está arriba o debajo del hiperplano H. OC1 perturba los coeficientes de H uno a la vez. Si se considera el coeficiente a_m como una variable y todos los otros coeficientes como constantes, entonces se puede ver a V_j como una función de a_m.

Sea

$$U_j = \frac{a_m \; x_{jm} - V_j}{x_{jm}} \tag{2.21}$$

Luego el punto P_j está debajo de H si $a_m > U_j$ y debajo en otro caso. De esta manera, al fijar los valores de los coeficientes $a_1, a_2, ..., a_{d+1}$, excepto a_m, se obtiene n constantes sobre el valor de a_m, usando los n puntos en el conjunto P. El problema se reduce a encontrar el valor para a_m que satisfaga a tantas constantes como sea posible; a_m es solo una división paralela al eje en una dimensión. El valor a_{m1} obtenido al solucionar este problema unidimensional es un buen candidato a ser usado como el nuevo valor del coeficiente a_m. Sea H_1 el hiperplano obtenido al cambiar a_m por a_{m1} en H. Si H tiene menos impureza que H_1 luego H_1 es descartado, en caso contrario H_1 es ahora la nueva la nueva localización del hiperplano.

Para decidir cual de los $d + 1$ coeficientes elegir para la perturbación en [47] los autores describen tres órdenes diferentes de perturbación de coeficientes.

Cuando se realiza la búsqueda de mejor hiperplano se presenta el problema de que el algoritmo se quede atrapado en un mínimo local cuando alguna perturbación del hiperplano actual no disminuye la medida de impureza y el hiperplano no minimiza globalmente la medida de

impureza. Los autores implementaron dos estrategias: la perturbación del hiperplano en una dirección aleatoria y la ejecucion del algoritmo de perturbación para la obtención de hiperplanos iniciales adicionales, esto es, cuando el hiperplano:

$$H = \sum_{i=1}^{d} a_i \cdot x_i + a_{d+1} \tag{2.22}$$

no puede ser mejorado por medio de la perturbación determinista, entonces ser realiza lo siguiente:

1. Sea $R = r_1, r_2, r_3, ..., r_{d+1}$ un vector aleatorio, y sea α la cantidad sugerida para perturbar H, entonces:

$$H = \sum_{i=1}^{d} (a_i \cdot \alpha r_i) x_i + (a_{d+1} + \alpha r_{d+1}) \tag{2.23}$$

es el nuevo hiperplano perturbado. Como se puede apreciar la única variante en la ecuación de H es α, por lo tanto, cada uno de los n ejemplos en P dependen de su categoría, imponen una restricción sobre el valor de α. Por otro lado, si H_1 obtiene mejoras en las mediciones de impureza, entonces acepta la perturbación.

Debido a que se usa aleatoriedad al inicio del método para obtener hiperplanos aleatorios, esta misma aleatoriedad determina qué mínimo local encontrará primero. Cuando el método de perturbación aleatoria falla para que el algoritmo salga del mínimo local, los autores sugieren iniciar nuevamente con nuevos hiperplanos iniciales.

El Algoritmo 3 muestra el pseudocódigo del algoritmo OC1.

2.3. Algoritmos evolutivos

Los algoritmos evolutivos (AEs) son un área de investigación dentro de la Inteligencia Artificial y de manera más específica en el campo de la inteligencia computacional. Tal como su nombre lo indica, son un tipo especial de cómputo cuyos métodos se inspiran en el proceso de la evolución natural. Se refiere a una forma especial de resolver problemas, tomando como analogía para hacerlo la propia evolución natural, debido a su éxito en las diferentes especies que habitan el planeta, cada una compitiendo por su sobrevivencia y por una oportunidad para reproducirse. Esta metáfora de evolución natural relaciona al cómputo evolutivo con un estilo muy particular de resolución de problemas, donde existe una colección de posibles soluciones y con las técnicas correctas éstas pueden ser seleccionadas para competir para sobrevivir y reproducirse, tal como los seres vivos, usando la técnica de prueba

Algoritmo 3 Pseudocódigo del algoritmo OC1 [46]

procedimiento OC1
    Normalizar todos los valores de los atributos
    $L = 0$
    mientras TRUE **hacer**
        $L = L + 1$
        Permitir que la división actual s_l sea $v < c$ donde $v = \sum_{i=1}^{d} a_i x_i$
        para $i = 1, \ldots, d$ **hacer**
            para $\gamma = -0.25, 0, 0.25$ **hacer**
                Buscar para el δ que maximiza la división $v - \delta(a_i + \gamma) \leq c$
            fin para
            Dejar que δ^*, γ^* sea la mejor configuración en las 3 búsquedas.
            $a_i = a_i - \delta^*, c = c - \gamma^*$
        fin para
        Perturbar c para maximizar la mejora de s_L, conservando las constantes $a_1, \ldots, a_d$
        si $|mejora(s_L) - mejora(s_{L-1}) \leq \epsilon|$ **entonces**
            Salir del bucle While
        fin si
    fin mientras
    Convertir s_L una división sobre los atributos sin normalización
    regresar el mejor s_L y las mejor división paralela a los ejes como la división de T
fin procedimiento

y error. Mejía-de-Dios [43] define de manera formal un algoritmo evolutivo, éste consiste de una tupla de 8 elementos:

$$AE = (I, \phi, \Omega, \Psi, s, l, \mu, \lambda) \tag{2.24}$$

donde, $I = A_x$ x A_s denota el espacio de las soluciones, A_x y A_s son conjuntos arbitrarios. ϕ denota una función objetivo o función fitness que asigna valores reales a los individuos.

$$\Omega = \left\{ w_{\theta_1}, w_{\theta_2}, w_{\theta_3}, \ldots, w_{\theta_i} | w_{\theta_i} : I^\lambda \to I^\lambda \right\} \cup \left\{ w_{\theta_0} : I^\mu \to I^\lambda \right\} \tag{2.25}$$

es un conjunto de operadores genéticos probabilísticos w_{θ_i} cada uno controlado por parámetros en el conjunto $\Omega_i \in \mathbb{R}$.

$$s_{\theta_s} : I^\lambda \cup I^{\mu+\lambda} \to I^\lambda \tag{2.26}$$

denota el operador de selección, este operador puede cambiar el número de individuos de λ o $\lambda + \mu$ a μ donde μ, $\lambda \in N$.

Este conjunto de operadores son la fuerza que forma las bases de los sistemas evolutivos. Además, son los operadores de selección los que actúan como la fuerza que aumenta la calidad media de las soluciones de la población.

Todos los algoritmos evolutivos comparten algunos elementos y solo varían en la forma de aplicar los operadores. En el **Algoritmo 4** se muestra el esquema general de un algoritmo evolutivo en forma de pseudocódigo.

Algoritmo 4 Esquema general de un algoritmo evolutivo [21]

```
1: procedimiento AE()
2:     Inicializar la población con soluciones candidatas aleatorias
3:     Evaluar cada solución candidata de la población
4:     mientras Condición de paro se cumpla hacer
5:         Seleccionar padres
6:         Recombinar pares de padres
7:         Mutar descendencia obtenida de 5
8:         Evaluar los nuevos candidatos
9:         Seleccionar individuos para la siguiente generación
10:    fin mientras
11: fin procedimiento
```

2.3.1. Conceptos y elementos

Tomando como base la metáfora de la evolución natural podemos tomar los conceptos básicos de ésta y mapearlos a nuestro contexto, que en este

caso es la búsqueda automatizada de soluciones a problemas de optimización. Primero, en el contexto de evolución, tenemos a un grupo de individuos, que como ya se ha mencionado, compiten en un entorno, que posee recursos limitados, por su supervivencia y reproducirse.

Trasladando este enfoque a la solución de problemas, podemos ver a este grupo de individuos como las soluciones candidatas que, válgase la redundancia, proporcionan una mejor solución al problema planteado. En la jerga utilizada por la comunidad, los individuos también se llaman soluciones candidatas o fenotipos, una vez que estos son codificados dentro del algoritmo evolutivo serán llamados genotipos o cromosomas. Ellos son usados para denotar los puntos en el espacio donde la búsqueda evolutiva esté tomando lugar. Al conjunto de individuos se le llama población y al entorno se llama problema.

Según Eiben, [21] un AE tiene los siguientes elementos:

- Representación (Definición de los individuos). La base para el éxito de un algoritmo evolutivo es su esquema de representación de las soluciones. Es la liga entre el mundo real y el mundo del AE. Implica abstraer algunos aspectos del mundo real para definir un problema bien definido dentro del cual las soluciones puedan existir y ser evaluadas. Así mismo, implica también como van a ser representadas las soluciones y como serán almacenadas de tal manera que puedan ser manipuladas por una computadora.

- Función de evaluación (Función de aptitud). Es un modelo matemático que representa los requerimientos que los individuos que conforman la población deben tener para adaptarse al contexto del AE. Desde la perspectiva de la solución del problema representa la tarea que debe ser resuelta. Técnicamente es una función que asigna a cada elemento de la población una medida de calidad que indique si este individuo es apto para resolver el problema o no.

- Población. Es la representación de las posibles soluciones al problema. Es un conjunto de genotipos. Generalmente se crean a partir de una heurística sencilla que genera soluciones aleatorias. Los individuos que pertenecen a la población son objetos estáticos que no cambian ni se adaptan, es la población quien lo hace.

- Mecanismos de selección de padres. La tarea principal de un mecanismo de selección de padres es diferenciar entre los individuos que tienen una mejor calidad y seleccionarlos para que generen nuevas soluciones en la siguiente generación. Bajo el principio de que si tienen una buena calidad, estos heredarán a sus hijos esta característica una vez que les sea aplicado un operador de variación. Generalmente este mecanismo es probabilístico y junto con el mecanismo de selección de

sobrevivientes son los responsables de mejorar la calidad de los individuos. Generalmente los individuos con mayor calidad tienen mejores oportunidades de convertirse en padres.

- Operadores de variación (mutación y recombinación). La principal tarea de estos operadores es crear nuevos individuos. Básicamente son de dos tipos: el operador de mutación, es siempre estocástico y provoca que algunos de los genes de un individuo sea perturbado de manera aleatoria bajo cierta probabilidad, provocando un cambio aleatorio e imparcial. Por otro lado, el otro operador es llamado recombinación o cruza, su función es la de fusionar o combinar información de dos genotipos primarios en uno o dos genotipos descendientes. De la misma forma que la mutación, es un operador estocástico. La forma de seleccionar las partes de los padres que serán combinadas y como realizar esta combinación son aleatorios.

- Mecanismo de selección de sobrevivientes (reemplazo). Es el encargado de seleccionar los mejores individuos descendientes en función a su calidad, y se encarga de mantener el tamaño de la población constante. Como su nombre lo indica, reemplaza los individuos ya sean padres o los de menor calidad en la población que pertenecen a la generación anterior por los descendientes de la actual generación, favoreciendo a los que tienen mejor calidad para realizar esta tarea.

- Inicialización. Este es un mecanismo que genera los primeros individuos que tendrá la población, generalmente se generan aleatoriamente.

- Condición de terminación. Es la forma que existe para detener un AE, se pueden usar distintas estrategias para hacerlo: después de un tiempo permitido por el procesador del equipo que esté ejecutando el algoritmo, una vez que se ha alcanzado un número máximo de evaluaciones, cuando una mejora se ha alcanzado un límite o umbral durante un periodo de tiempo, y cuando la diversidad de la población cae por debajo de un umbral permitido.

2.3.2. Tipos de algoritmos evolutivos.

En la literatura sobre algoritmos evolutivos se pueden encontrar diferentes paradigmas sobre AEs. Joshi *et al.* [36] distingue los siguiente paradigmas:

Algoritmos Genéticos (AGs). Los AGs son el tipo de algoritmo evolutivo más conocido. Originalmente fueron concebidos por Holland como una forma de estudiar el comportamiento adaptativo, sin embargo, la comunidad los ha considerado como métodos de optimización de funciones [21]. La estrategia general de esta técnica es mover de una población de soluciones

candidatas a otra nueva población aplicando una clase de selección natural y los operadores genéticos de cruza y mutación. [32]

Programación Genética (PG). Poli *et al.* [54] definen a la programación genética como una técnica de cómputo evolutivo que resuelve problemas automáticamente sin tener que decirle a la computadora como hacerlo. Es un tipo especial de algoritmo evolutivo (EA) donde los individuos se representan en forma de árbol. Durante este proceso, la PG aplica los operadores genéticos los cuales son especializados para actuar en programas de cómputo.

Estrategias Evolutivas (EE). Fueron desarrolladas por Rechenberg en 1971 [76] y son utilizados principalmente para modelar experimentos empíricos que son difíciles de modelar matemáticamente. La estrategia evolutiva simplemente se concentra en traducir los mecanismos fundamentales de la evolución biológica para problemas de optimización técnica. De manera similar a algunas estrategias de los algoritmos genéticos, las EE también usan vectores de números reales para representar los parámetros a optimizar, mientras que otro vector define los parámetros de la estrategia el cual controla la mutación de los parámetros objetivo. Tanto los parámetros de estrategia, como los del objetivo forman la estructura de datos para un simple individuo.

Programación Evolutiva (PE). Este paradigma fue ideado por Fogel [64], es muy parecido a la programación genética, a excepción de que aquí la estructura del programa es fijo y a sus parámetros numéricos se les permite evolucionar. Básicamente son modelos con reproducción asexual, su principal operador de variación es el de mutación, y posee un mecanismo de reemplazo típicamente es por torneo estocástico.

En años recientes aparece un nuevo modelo de algoritmo genético que ha sido conocido ampliamente:

Evolución Diferencial (ED). El algoritmo ED es un método evolutivo introducido originalmente por Storn y Price en 1995 para resolver problemas de optimización numérica [72]. La estrategia básica de este método es emplear la diferencia de dos individuos de la población, seleccionados aleatoriamente, como una fuente de variaciones aleatorias para producir un tercer vector o individuo. En su libro [56] lo describen detalladamente. Los individuos para este algoritmo particular son representados por vectores de valores continuos $x = (x_1, x_2, x_3, ..., x_n)$ con n parámetros, entonces, $x \in \mathbb{R}^n$.

Se pueden identificar tres fases en este algoritmo. La primera fase de inicialización contempla la selección y evaluación de la población inicial. Antes de ser inicializada la población se deben especificar los límites superior e inferior de cada parámetro. Estos valores, pueden ser almacenados en dos vectores, L_i y L_s, donde los subíndices i y s indican los limites inferior y superior respectivamente. Una vez que se especificaron estos límites, se puede inicializar la población de manera aleatoria dentro de un espacio de búsqueda finito $\epsilon \in \mathbb{R}^n$. Cada parámetro de cada vector se inicializa con

un valor dentro del rango de límites especificados. Entonces, el valor inicial en la generación cero del j-ésimo parámetro del vector i-ésimo, denotado por x_j^i es determinado por la siguiente ecuación:

$$x_j^i \ = \ alea_j(0,1) \cdot (L_{j,\ s} \ - \ L_{j,\ i}) \ + \ L_{j,i} \tag{2.27}$$

Donde $alea_j(0,1)$ genera un número aleatorio uniformemente distribuido entre cero y uno.

La fase evolutiva implementa un esquema iterativo para evolucionar la población inicial, donde cada iteración es una generación y una nueva población de individuos es generada. En ED, a diferencia de otros algoritmos evolutivos, el orden de aplicación de los operadores genéticos cambia. Primero se aplica el operador de mutación, posteriormente el de cruzamiento y al final el de selección para obtener el mejor individuo. Así mismo, el algoritmo no implementa los operadores de variación y selección tradicionales de los algoritmos evolutivos, en su lugar, crea cada individuo nuevo con una combinación lineal de los valores de dos o más individuos de la población actual seleccionados aleatoriamente. La forma en que se implementan estos operadores se utiliza para identificar las variantes que tiene el algoritmo.

Una notación aceptada para representar las variantes, y que suele encontrarse en la literatura, es con el siguiente formato: DE/A/B/C. Donde A y B determinan el tipo de mutación que se va a implementar y C especifica el tipo de cruzamiento. De manera específica, A representa el criterio de selección de uno de los individuos para construir el individuo mutado, y B el número de vectores diferencia usados para el operador de mutación. En la literatura se pueden encontrar 6 tipos diferentes de operadores de mutación [51] *et. al*:

$DE/rand/1 : v^i \ = \ x^{r_i} + F(x^{r_2} \ - \ x^{r_3})$,
$DE/best/1 : v^i \ = \ x^{best} + F(x^{r_1} \ - \ x^{r_2})$,
$DE/current-to-best/1 : v^i \ = \ x^i + F(x^{best} \ - \ x^{r_1}) \ + \ F(x^{r_2} \ - \ x^{r_3})$,
$DE/rand/2 : v^i \ = \ x^{r_1} + F(x^{r_2} \ - \ x^{r_3}) \ + \ F(x^{r_4} \ - \ x^{r_5})$,
$DE/rand/2 : v^i \ = \ x^{best} + F(x^{r_1} \ - \ x^{r_2}) \ + \ F(x^{r_3} \ - \ x^{r_4})$

Donde v^i es el vector mutado, x^{r_j} son individuos de la población seleccionados aleatoriamente, x^{best} es el individuo con la mejor aptitud de la población en la generación $g-1$, y F es un valor elegido por el usuario que representa un factor de escala usado para controlar la variación diferencial.

Por otro lado, el operador de cruzamiento se utiliza para combinar la información que tiene el vector mutado obtenido por alguno de los operadores descritos anteriormente y el vector $x^i \ \in \ X$ donde $i \ = \{0,\ 1,\ 2,\ ...,\ n-1\}$ llamado vector objetivo, el resultante es llamado vector de prueba t^i. Este operador de cruza se aplica por cada $j \ \in \{1,\ 2,\ 3,\ ...\ ,\ n\}$, es decir, por cada individuo x_j^i o v_j^i se seleccionan para formar el nuevo t^i usando un factor de cruzamiento $CR \ \in \ [0,1]$ especificado por el usuario.

Se pueden utilizar dos tipos de cruzamiento: binomial y exponencial:

$$binomial : t_j^i = \begin{cases} v_j^i & \text{si } r \leq CR \text{ o bien j} = 1, \\ x_j^i & \text{en otro caso} \end{cases} \tag{2.28}$$

donde r es un valor aleatorio entre 0 y 1.

$$exponencial : t_j^i = \begin{cases} v_j^i & \text{si } r \leq j \leq min(l + L - 1, n), \\ x_j^i & \text{en otro caso} \end{cases} \tag{2.29}$$

donde l representa la posición inicial de la secuencia en v^i y L es el número de intentos consecutivos $L < n$ en los cuales $r \leq CR$ que determina la posición final de la secuencia.

Por último, para seleccionar el individuo con la mejor aptitud se puede aplicar un torneo uno a uno entre x^i y t^i. Este individuo será nuevo miembro de la nueva población en la generación $g + 1$.

2.3.3. Algoritmos basados en la Física.

Este tipo de algoritmos son inspirados en fenómenos físicos en áreas tales como las leyes gravitacionales de Newton, la teoría cuántica, el electromagnetismo, entre otras. Biswas et al. [11] proponen una clasificación de estos métodos en diferentes categorías dependientes de las áreas a las que pertenecen las metáforas físicas que sirven como inspiración: leyes de gravitación de Newton, mecánica cuántica, teoría del universo, electromagnetismo, electrostática. Entre los algoritmos más conocidos a continuación se describen dos, el algoritmo de búsqueda gravitacional (GSA) y el algoritmo de centros evolutivos (ECA) que es eje central del presente trabajo de investigación.

Algoritmo de Búsqueda Gravitacional (GSA). El algoritmo GSA es inspirado en la ley de gravitación universal y movimiento de Newton. Fue introducido por Rashedi et al. [59]. El algoritmo toma la metáfora de que los objetos son atraídos unos a otros por la fuerza de gravedad, y esta fuerza causa un movimiento global de todos los objetos hacia los objetos con masas más pesadas. Para el algoritmo estas masas más pesadas representan las mejores soluciones o soluciones óptimas en el espacio de búsqueda.

Cada masa tiene cuatro especificaciones: la posición, su masa inercial, su masa gravitacional activa y la masa de gravitación pasiva. La posición de la masa corresponde a las soluciones del problema, mientras que las otras especificaciones se determinan usando la función de aptitud. Por lo tanto, cada masa representa una solución, entonces el algoritmo ajusta adecuadamente la fuerza de gravedad y la masa de inercia, entonces se espera que las masas más pesadas atraigan a las masas más ligeras (que representan soluciones de baja calidad).

Algoritmo de Centros Evolutivo (ECA). Es un algoritmo evolutivo basado en la Física, propuesto por Mejía-de-Dios y Mezura [44]. El algoritmo

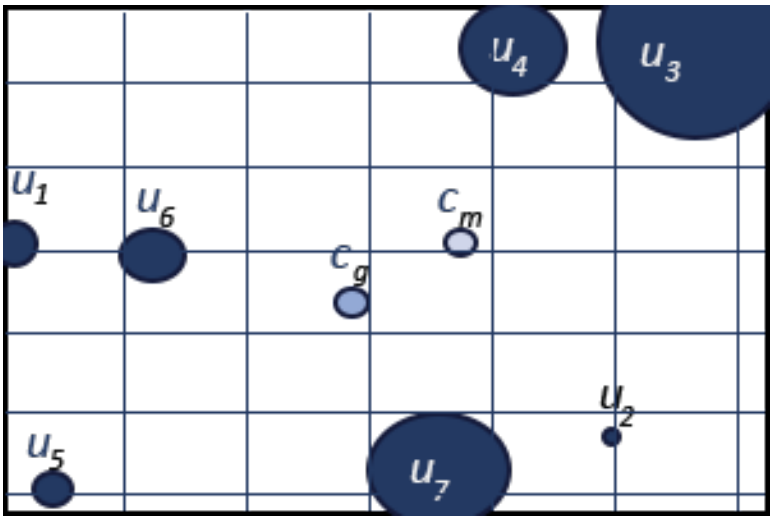

Figura 2.4: c_m es el centro de masa, c_g es el centro geométrico de los puntos negros. El radio del punto negro representa su masa. [44]

se basa en algunos principios físicos y mecánicos, de manera más específica utiliza el concepto de centro de masa para crear nuevas direcciones para mover los elementos de la población que no proporcionan soluciones de calidad hacia mejores regiones en el espacio de búsqueda. Para ello utiliza el concepto físico de centro de masa. Este punto es una propiedad física de todo cuerpo, representa la posición media del peso de un objeto.

De la figura 2.4 se puede apreciar que los puntos más grandes representan las masas más grandes, por lo que el sesgo de las nuevas direcciones tienden hacia estos puntos.

Se puede describir tanto el movimiento del cuerpo completamente, así como su rotación, si se le aplica una fuerza en este punto, de un lugar a otro. El algoritmo ECA utiliza este concepto para trasladar la población a lugares prometedores del espacio de búsqueda donde la masa de la población entera es máxima.

Debido a su naturaleza de algoritmo evolutivo, ECA posee sus elementos y solo cambia la manera de aplicar los operadores de variación y selección. A continuación se describe de manera detallada el algoritmo.

Sea $P = \{x_1, \ x_2, \ x_3 \ , ..., \ x_n\}$ una población generada de manera aleatoria de N soluciones. Por cada $x_i \in P$, seleccionar un subconjunto $U \subset P$ con K soluciones. De U se calcula el centro de masa con la siguiente ecuación:

$$c_i = \frac{1}{W} \sum_{u \in U} f(u) \cdot u, \qquad W = \sum_{u \in U} f(u) \tag{2.30}$$

donde W es la suma de los pesos del subconjunto U.

Después se selecciona una solución $u_a \in U$. Una vez generado el centro

de masa c, se genera una dirección para localizar una nueva solución h_i usando la siguiente estrategia:

$$h_i \ = \ x_i \ + \ \mu(c_i \ - \ u_r) \tag{2.31}$$

Donde μ es un número aleatorio entre 0 y $\mu_{max} = 2$.

Si el centro de masa coincide con el centro geométrico de U, entonces el centro de masa es:

$$c_i \ = \ \frac{1}{K\alpha} \sum_{u \ \in \ U} \alpha \cdot u \ = \ \frac{1}{K} \sum_{u \ \in \ U} u \tag{2.32}$$

El sesgo de las nuevas soluciones es proporcionado por la ecuación de c_i. Esto es debido a que para una solución con la masa más grande, la posición del centro de masa es más cercana a esta posición.

En el **Algoritmo** 5 se aprecia el pseudocódigo de ECA.

Algoritmo 5 Pseudocódigo del algoritmo ECA [44]

procedimiento ECA($K = 7, \mu_{max} = 2$)
    $N \leftarrow 2K * D$
    Generar y evaluar la población inicial P con N elementos
    mientras $r \neq criterioParo$ **hacer** ▷ Mientras no se cumpla el criterio de paro
        para cada x en P **hacer**
            Generar $U \in P$ tal que card(U) $= K$
            Calcular c usando U con 2.31
            $\varphi \leftarrow rand(0, \mu_{max})$
            $h \leftarrow x + \varphi * (c - u)$
            si $f(x) < f(h)$ **entonces**
                Agregar h a A
            fin si
        fin para
        $P \leftarrow mejores\ elementos\ en\ P \cup A$
    fin mientras
    regresar Mejor solución en P ▷ Reportar mejor solución en P
fin procedimiento

2.4. Algoritmos evolutivos para inducir árboles de decisión

Una técnica para realizar la tarea de inducción de árboles de decisión es generar todos los posibles árboles de decisión que clasifiquen correctamente el conjunto de entrenamiento y luego seleccionar el más simple de ellos, el

número de estas soluciones es finito pero muy grande [58]. Esto nos lleva a una búsqueda en el espacio de los árboles de decisión.

Los algoritmos evolutivos son metaheurísticas[1] basadas en población que usan un grupo de soluciones candidatas en cada paso de su proceso iterativo, su enfoque puede aplicarse para generar posibles árboles en el espacio de los árboles de decisión, tal como se ha mencionado en el párrafo anterior y después seleccionar el mejor de ellos. Las siguientes técnicas evolutivas se han utilizado para inducir árboles de decisión.

2.4.1. Enfoques basados en algoritmos genéticos

La mayoría de las técnicas tradicionales para inducir árboles de decisión están basados en una estrategia agresiva de particionamiento recursivo para el crecimiento del árbol. Usan diferentes variantes de medida de impureza como ganancia de información, índice de Gini, entre otros, para seleccionar el atributo de entrada que será asociado a un nodo. Una de sus desventajas es que guía a soluciones subóptimas, y al particionar el conjunto de datos recursivo puede resultar en subconjuntos muy pequeños lo cual puede llevar a un sobreajuste de datos. Estas técnicas realizan una búsqueda local en cada nodo.

Por otro lado, los algoritmos evolutivos realizan una búsqueda robusta global, es decir, buscan un equilibrio tanto el la exploración, así como en la explotación en el espacio de búsqueda de las soluciones candidatas. Este enfoque ha sido utilizado por los investigadores para diseñar técnicas basadas en algoritmos genéticos para inducir árboles de decisión.

Varias soluciones se han propuesto con este enfoque. Aitkenhead [53] aplica co-evolución competitiva entre el árbol de decisión y el conjunto de datos de entrenamiento. En [4] Utiliza un enfoque semejante para la inducción árboles de decisión. En [71] sugieren no utilizar cadenas binarias para representar puntos en el espacio de búsqueda, afirman que tales representaciones no son adecuadas para representar el espacio de los árboles de decisión y que los operadores de cruza y mutación pueden ser alterados eficientemente para que coincidan con la estructura del árbol. Bandar *et al* [5] proporcionan un método basado en AG para la inducción de múltiples árboles de decisión. Por otro lado, Smith [5] utiliza cromosomas lineales para

[1] La palabra heurística proviene del heuriskein, que significa "encontrar" o "descubrir". Wang *et al* [77] proporcionan las siguientes definiciones. Heurística, es una metodología de razonamiento en la resolución de problemas que permite una solución a un problema impulsada por prueba y error. Una metaheurística es una heurística genérica o de nivel superior que es más general en resolución de problemas. El cómputo metaheurístico es una computación adaptativa que aplica heurística general reglas para resolver una categoría de problemas computacionales. En otras palabras, una metaheurística es un procedimiento de búsqueda de alto nivel que aplica alguna regla o conjunto de reglas basado en una fuente de conocimiento, a fin de explorar el espacio de búsqueda de manera más eficiente que los algoritmos exactos.

inducir árboles de decisión para buscar genes RNA en datos genómicos, a pesar de que su estrategia es de dominio específico de la aceleración de la búsqueda de RNA puede ser extendido fácilmente a un enfoque genético.

2.4.2. Enfoques basados en programación genética

Taleby *et al* [2] describen la programación genética como una técnica que emplea métodos de optimización nuevos, específicamente de algoritmos evolutivos, para modificar programas de cómputo, imitando la forma en que los humanos desarrollan programas, al reescribirlos progresivamente para solucionar problemas automáticamente.

Para tareas de inducción de árboles de decisión se han utilizado las técnicas de GP como las que aplicaron Buontempo et al. [13] que utilizaron la programación genética para inducir árboles de decisión, como una solución a la pérdida de regiones del espacio de búsqueda que se generan cuando se utilizan técnicas de búsqueda voraces que se basan en las particiones de recursivas para elegir el mejor atributo y valor de división en cada nodo. Así mismo, Shali et al. [69] utilizaron GP para evolucionar y encontrar un criterio de prueba óptimo en cada nodo interno para el conjunto de ejemplos en ese nodo y de esta manera inducir árboles de decisión oblicuos. Folino et al [26] usaron una técnica de programación genética paralela para inducir árboles de decisión, en el que una población de árboles es evolucionada al utilizar los operadores genéticos y cada individuo es evaluado por una función objetivo. Krawiec [37] propone un método que gira en torno a C4.5 y la medida de precisión del algoritmo de inducción del árbol de decisión se utiliza como la aptitud. En su enfoque cada individuo es un bosque que tiene un cierto número de árboles, cada árbol codifica una nueva característica, una parte de estos árboles se utiliza como un almacén donde se guardan los mejores individuos, de este depósito se eligen los mejores sobre una base medida dada por el número de veces que el algoritmo de aprendizaje C.45 usa la característica construida. En [23] y [22] proponen soluciones parecidas.

2.4.3. Enfoques basados en evolución diferencial

Rivera [61] utilizó este algoritmo para inducir árboles de decisión, obteniendo resultados que mejoran los producidos con técnicas tradicionales como C4.5, CART, OC1 entre otros. En su trabajo utiliza una estrategia de búsqueda global para encontrar árboles de decisión paralelos a los ejes cercanos al óptimo. Para ello, codificaron los nodos internos del árbol en cromosomas de valores reales, y una población de ellos evoluciona usando la precisión del entrenamiento de cada uno de ellos como el valor de aptitud. Sus experimentos han demostrado que utilizar el método de ED, y en general los AEs, es una buena estrategia para inducir árboles de decisión, comparados con técnicas tradicionales.

2.4.4. Análisis de estrategias

Como resultado de la revisión de la literatura presentada en las secciones anteriores, es posible realizar una comparación breve de las diferentes estrategias para inducir árboles de decisión. Las técnicas clásicas, fundamentalmente realizan una búsqueda local a nivel de nodos. Esto es debido a que realizan las pruebas de cada atributo mediante una búsqueda voraz, utilizando herramientas estadísticas, y más específicamente, de teoría de la información, tales como la entropía y ganancia de información, para dividir el conjunto datos de entrenamiento.

Por otro lado, se han utilizado estrategias basadas en cómputo bioinspirado, que han resultado ser más eficientes para tareas de inducción de árboles de decisión. Estas técnicas realizan búsquedas globales en el espacio de los árboles de decisión. Como resultado de utilizar estos métodos se obtienen árboles más pequeños. Esto representa una ventaja importante respecto a las técnicas tradicionales, esto debido a que los árboles pequeños tienen un mejor nivel interpretativo.

Capítulo 3

ECA para inducir árboles de decisión

En este capítulo se describe un algoritmo basado en la Física para la inducción de árboles de decisión paralelos a los ejes. Primero se describe la representación de las soluciones, después se aplica ECA para encontrar un árbol de decisión cercano al óptimo que clasifique de manera adecuada el conjunto de entrenamiento.

3.1. Contexto de ECA

Los árboles de decisión se han utilizado ampliamente para resolver problemas de clasificación, y se han desarrollado muchas variantes de algoritmos o metodologías para su inducción, cada uno de ellos mejoran algunas deficiencias que otros tienen y resuelven mejor problemas que otros no. La mayoría de los enfoques que se les dan a los criterios de exploración en las investigaciones han sido restringidas a conjuntos de datos con valores simbólicos [58] o aquellas donde se prueban en cada nodo una sola variable (univariables) con valores numéricos, donde se comparan un simple atributo contra una constante o umbral. Estos enfoques son equivalentes a dividir el conjunto de datos con un hiperplano paralelo a alguno de los ejes, de ahí el nombre de árboles de decisión paralelos a los ejes.

Por otro lado, las técnicas bio-inspiradas, proporcionan un enfoque diferente, al explorar el espacio de búsqueda de las soluciones de los problemas. Rivera *et al* [62] han propuesto una solución elegante para representar las soluciones candidatas, y tomar como base el algoritmo OC1 [58], junto con su variante basada en GA y aplicando el algoritmo ED para encontrar el hiperplano oblicuo más cercano al óptimo en cada nodo interno de un árbol de decisión, para ello utilizaron un particionamiento recursivo. Del mismo modo, aplicaron una estrategia de búsqueda global, donde ED evoluciona una población de vectores de valores reales, que representan tanto los atribu-

tos como los valores umbral o límites asociados con el número de atributos evaluados en los nodos internos de un árbol de decisión, de esta manera obtienen árboles paralelos a los ejes.

En su trabajo, con estos métodos obtuvieron árboles de decisión tanto oblicuos como aquellos paralelos a los ejes muy competitivos. Sus experimentos en varios conjuntos de datos demostraron que su método encuentra árboles mucho más pequeños, comparables con los métodos univariables, incluso mejorando algunos resultados obtenidos con otros métodos clásicos de aprendizaje automático.

En este contexto de ubica la principal motivación del presente trabajo de investigación, al introducir un algoritmo evolutivo inspirado en la Física (ECA) que no ha sido utilizado para generar árboles de decisión, y que ha demostrado ser muy eficiente en pruebas de optimización numérica, incluso mejorando algunos resultados que otros algoritmos bio-inspirados.

3.2. Codificación de soluciones candidatas

Como se ha establecido previamente, la representación de las soluciones es un factor clave para el buen funcionamiento de cualquier algoritmo de aprendizaje automático. Es la relación entre el algoritmo que buscará la solución que lo resuelva y el dominio del problema en el mundo real. Si se elige una representación adecuada para las soluciones candidatas entonces esto conducirá a tener resultados de buena calidad. En esta sección se describe la representación que tendrán las soluciones en el presente trabajo de investigación. Esta representación, así como la estrategia para inducir los árboles de decisión que aquí se utilizan, se introducen en el trabajo realizado por Rivera y Canul [62].

3.2.1. Esquema de representación lineal de las soluciones candidatas

Similar a la propuesta de Rivera y Canul, en el presente trabajo de investigación los nodos internos de un árbol de decisión serán codificados en un vector de soluciones, y ECA evolucionará una población de éstos, usando el porcentaje de la precisión del árbol de decisión como criterio para encontrar el árbol más cercano al óptimo. Cada solución será representada y almacenada linealmente en un vector de valores reales de longitud fija donde se codificarán tanto los atributos como los valores de umbral asociados al número de atributos numéricos evaluados en los nodos internos del árbol de decisión. Esta representación lineal se puede usar sin modificaciones para codificar soluciones candidatas en los diferentes EAs. El presente trabajo de tesis aprovechará esta ventaja como base para representar las soluciones y les aplicará la estrategia evolutiva implícita en el algoritmo ECA.

Debido a que los EAs usan representaciones de longitud fija, es necesario definir a priori este tamaño. La estimación de este tamaño se calcula tomando en cuenta la estructura del conjunto de datos cuyo modelo de clasificación se desea construir. Una vez obtenido el vector que representa la solución candidata, se le aplica un esquema para mapear y obtener un árbol de decisión paralelo a los ejes factible, usando una regla heurística llamada Smallest Position Value (SPV) [74] y los ejemplos de entrenamiento.

En la sección 2.2 se mencionó que en un árbol univariable o paralelo a los ejes. Cada condición de prueba evalúa solo un atributo para dividir el conjunto de entrenamiento. En caso de ser atributos categóricos, el conjunto de entrenamiento es dividido en tantos subconjuntos como valores hay en el dominio del atributo. Por otro lado, si lo que se va a evaluar son atributos con valores numéricos, entonces es necesario un valor de umbral, límite o constante para dividir el conjunto de entrenamiento en dos subconjuntos. Entonces, el método para inducir este tipo de árboles debe determinar este valor de umbral adecuado optimizando algún criterio de división.

El vector que representará una solución candidata se obtiene asociando cada parámetro de un individuo con cada atributo y con cada valor de umbral usado en las condiciones de prueba de un árbol de decisión paralelo a los ejes, se denominará x^i a este vector. Entonces, si y^i y z^i son vectores que representan la secuencia de atributos y la secuencia de valores umbral respectivamente, para formar un individuo que codifique una solución se debe concatenar y^i seguido de z^i, esto es, $x_i = y^i \frown z^i$ como se puede apreciar en la figura 3.1

Si n_y es el número de elementos que hay en y^i y n_z el número de elementos que tiene z^i, el tamaño de x^i es $n = n_y + n_z$.

Debido a que no se conoce a priori la estructura del árbol que se va a generar, además de que solo se tiene un conjunto de datos a partir del cual se va a construir un árbol de decisión, y a que la altura de un árbol depende fuertemente del tamaño del conjunto de entrenamiento, en el presente trabajo se utiliza la altura de un árbol binario y la estructura del conjunto de datos para calcular el tamaño de x^i.

Para obtener la altura de un árbol de decisión completo se sigue el siguiente procedimiento. Si H representa esta altura, el número de nodos internos se puede determinar mediante $2^H - 1$, además, el número de nodos hoja de este mismo árbol es 2^H, se pueden obtener dos alturas mediante:

$$H_i = \lceil log_2(d+1) \rceil \tag{3.1}$$

y

$$H_l = \lceil log_2(s) \rceil \tag{3.2}$$

donde d es el número total de atributos del conjunto de entrenamiento, y s el número de etiquetas de clase usadas para inducir el árbol de decisión.

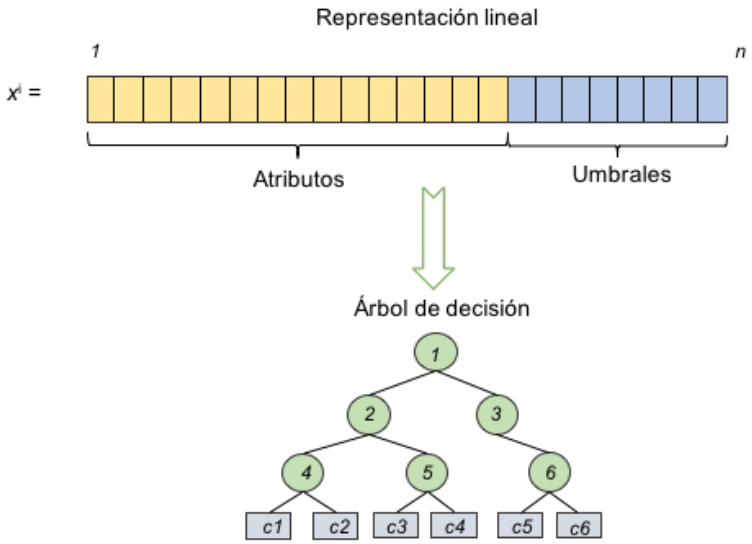

Figura 3.1: Representación lineal de un árbol de decisión [61].

Estas alturas representan la altura de los nodos internos (H_i) y la altura de los nodos hoja respectivamente (H_l).

Usando 3.1 y 3.2 para determinar el tamaño de n_y se obtiene la siguiente ecuación:

$$n_y = 2^{max(H_i,H_l)-1} - 1 \tag{3.3}$$

Sea $a = \{a_1, a_2, a_3, ..., a_n\}$ el vector de atributos extraído del conjunto de datos; dado que n_y no es más pequeño que d, cada atributo (a_i) en el conjunto de entrenamiento se puede asociar con uno o más elementos de y^i. Para realizar esta asociación, se utiliza un vector auxiliar llamado P que almacenará los índices de a. Entonces por cada $j \in \{1, 2, 3, ..., n_y\}$, la ubicación de cada atributo en el vector a es almacenado en el vector p de la siguiente manera:

$$p_j = j \ mod \ d \tag{3.4}$$

Por otro lado, como en z^i se almacenan los valores umbral o límite de los atributos numéricos asociados con y^i, entonces el tamaño de z^i depende del tamaño de y^i. El número de atributos numéricos en el conjunto de entrenamiento se calcula con la siguiente fórmula:

$$d_r = |\{k \in \{1, 2, 3, ..., d\} : D(a_k) \subseteq \mathbb{R}\}| \tag{3.5}$$

n_z se obtiene con:

$$n_z = d_z \left\lfloor \frac{n_y}{d} \right\rfloor + |\{k \in \{1, 2, 3, ..., d\} : D(a_k) \subseteq \mathbb{R} \ \wedge d \left\lfloor \frac{n_y}{d} \right\rfloor + k \leq n_y\}| \tag{3.6}$$

El primer término de 3.6 se refiere al número de atributos numéricos usados cuando a está completamente asociado con y^i, mientras que el segundo

término representa el número de atributos usados cuando a está parcialmente asociado con y^i.

A manera de ejemplo para ilustrar los conceptos y cálculos anteriores considérese el conjunto de datos Haberman's Survival Data Set que contiene los casos de estudio que se realizaron entre los años 1958 y 1970 en el Hospital Billings de la Universidad de Chicago sobre la supervivencia de los pacientes que se habían sometido a cirugía por cáncer mamario [40]. El conjunto de datos tiene los siguientes atributos:

Atributo	Tipo/valor
Age_of_patient_at_time_of_operation	*INTEGER*
Patients_year_of_operation *	*{58,59,60}*
Number_of_positive_axillary_nodes_detected	*INTEGER*
Survival_status	*{1,2}*

Tabla 3.1: Atributos del Haberman's Survival Data Set. (*) Sólo se tomaron tres valores de este atributo para efectos simplificar los cálculos del ejemplo.

El primer paso consiste en calcular el tamaño que tendrán los vectores solución. El método ECA-ADT utiliza la estructura del conjunto de datos para determinar la altura que tendrá el árbol de decisión y en función de ésta poder calcular el tamaño del vector solución.

Se puede apreciar que posee tres atributos, por tanto d=3. Hay tres atributos numéricos, por lo que $a = \{1, 2, 3\}$, y dos etiquetas de clase, por lo que $s = 2$. Debido a que la altura estimada de un árbol de decisión binario se puede calcular con: $hi = \lceil log_2(d + 1) \rceil$ para la altura considerando los nodos internos, entonces:

$$h_i = \lceil log_2(3 + 1) + 1 \rceil = 3 \tag{3.7}$$

además, dado que $h_l = \lceil log_2(s) \rceil$ sirve para calcular la altura tomando en cuenta el número de las etiquetas de clase, se tiene:

$$hl = \lceil log_2(2) + 1 \rceil = 2 \tag{3.8}$$

Entonces, el tamaño de y_i se calcula con $n_y = 2^{max\{hi,hl\}}-1$, sustituyendo valores en la anterior fórmula se tiene:

$$n_y = 2^{max\{3,2\}}-1 = 7 \tag{3.9}$$

Esto quiere decir que para 3 atributos están asociados con 7 elementos de y_i.

Ahora bien, para calcular el número de atributos numéricos que tiene el data set se usa la fórmula 3.5. Como se puede apreciar el conjunto de atributos del data set es $\{1, 2, 3\}$ de los cuales los atributos $\{1, 2, 3\}$ son

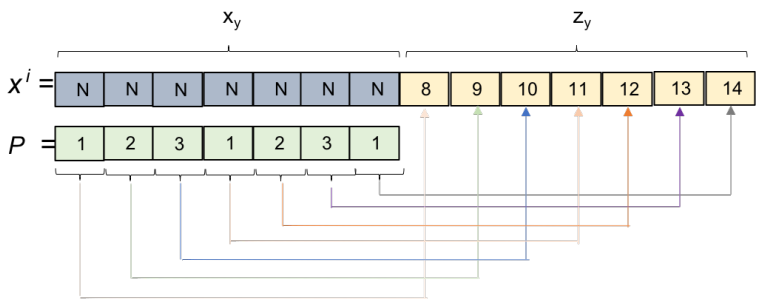

Figura 3.2: Ejemplo de representación

atributos numéricos, debido a que sus valores pertenecen al conjunto de los números reales, tal como lo indica la fórmula. Entonces la norma de este conjunto es tres, por lo que $d_r = 3$.

Dado que x^i tiene asignados 7 atributos numéricos, entonces $n_z = 7$.

Entonces el tamaño del vector solución que servirá para codificar un árbol de decisión binario completo es de $n = n_y + n_z$, esto es 14 elementos, tal como se puede apreciar en la figura 3.2

Luego, los atributos se asocian al individuo de la población por medio de un vector auxiliar de índices p. Estos índices pertenecen a las posiciones de los atributos dentro de otro vector.

3.2.2. Inducción de árboles de decisión paralelos a los ejes

Una vez que se ha estimado el tamaño de los vectores que representarán las posibles soluciones, ECA genera una población de soluciones aleatorias, pero estas no representan árboles de decisión, son solo vectores con parámetros numéricos aleatorios, éstos se deben transformar a árboles de decisión utilizando la regla SPV y el conjunto de datos, lo realiza por medio de un mapeo.

ECA-ADT implementa un proceso de tres etapas para mapear un árbol de decisión paralelo a los ejes a partir de la representación de una solución de la población. Primero se usa x^i para construir un vector auxiliar w^i el cual codifica una secuencia de nodos candidatos del DT. Posteriormente w^i es utilizado para crear un árbol de decisión parcial pT^i que contiene únicamente nodos internos. En la última etapa, para completar el DT, un conjunto de nodos hoja se agregan a pT^i usando el conjunto de entrenamiento.

1) Etapa 1: Construcción del vector de nodos

Para construir el vector de nodos el método ECA-ADT utiliza la regla SPV para obtener un vector con atributos con una secuencia ordenada a partir de x^i. La regla SPV genera un vector o^i basándose de los elementos de y^i siguiendo el siguiente procedimiento: la localización del elemento con valor más bajo en y^i se almacena en la primera posición de o^i; la localización

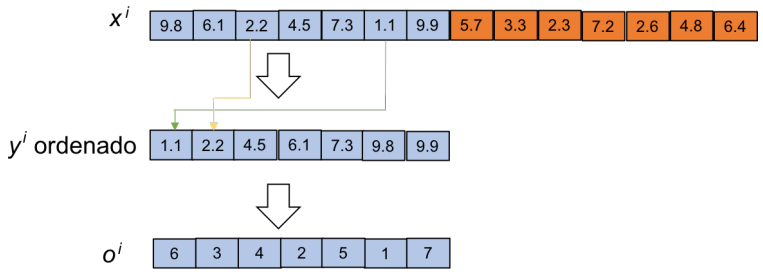

Figura 3.3: Ejemplo de aplicación de la regla SPV para obtener o^i

del siguiente elemento más pequeño de y^i se almacena en la segunda posición de o^i; el procedimiento se repite recursivamente por cada elemento de y^i. Entonces como resultado obtenemos un vector de posiciones ordenadas o^i.

Con el objetivo de ilustrar el procedimiento, se retoma el ejemplo que se utilizó para estimar el tamaño del árbol. Suponiendo que ECA generó un vector solución aleatorio como el que se muestra el la figura 3.3. El elemento más pequeño de x^i es 1.1, por lo tanto se coloca en la primera posición, el segundo más pequeño es 2.2, que se coloca en la segunda posición, y así sucesivamente. Después se coloca en el vector o^i las posiciones respectivas que ocupan estos valores, para este caso, la posición del valor 1.1 es la 6, la posición del valor 2.2 es 3, y así sucesivamente.

Por otro lado, para ajustar los elementos del vector de umbrales z^i de tal manera que éstos se encuentren dentro del dominio de los atributos numéricos que se encuentren en a, se utiliza otro vector auxiliar t^i.

Para calcular la posición en la que se encuentra el umbral en x^i del atributo numérico $a_{p_{o^i_j}}$ se utiliza la ecuación 3.10

$$q = d_r \left\lfloor \frac{j}{d} \right\rfloor + |k \in 1, ..., d : D(a_k) \subseteq \mathbb{R} \wedge k \leq p_{o^i_j}| \tag{3.10}$$

Luego t^i_q representa el valor de umbral de $a_{p_{o^i_j}}$, que se obtiene aplicando la siguiente ecuación:

$$t^i_q = minD(a_{p_{o^i_j}}) + \frac{(z^i_j - x^{min}_j)(max\{D(a_{p_{o^i_j}})\} - min\{D(a_{p_{o^i_j}})\})}{x^{max}_j - x^{min}_j} \tag{3.11}$$

Una vez que o^i contiene los índices ordenados de y^i, y t^i contiene los valores umbral asociados con los atributos numéricos codificados en y^i, esos vectores son usados para construir el vector w^i que representará la secuencia de los nodos internos candidatos de un árbol parcial.

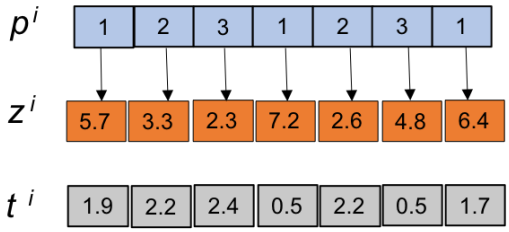

Figura 3.4: Ejemplo de construcción del vector t^1

Para cada $j \in \{1, ..., n_y\}$, el elemento j-ésimo de w^i es calculado con la siguiente ecuación 3.12:

$$w_j^i = \begin{cases} (a_{p_{o_j^i}}, t_q^i) & \text{si } D(a_{p_{o_j^i}}) \subseteq \mathbb{R} \\ (a_{p_{o_j^i}}) & \text{de otra manera} \end{cases} \tag{3.12}$$

En la figura 3.4 se muestra el proceso para construir el vector t^i a partir de z^i. Suponiendo que los dominios hipotéticos de los datos en nuestro conjunto de datos de ejemplo sean: $a_1 = \{1.2, 2.5\}$, $a_2 = \{1.5, 4.3\}$, $a_3 = \{2.2, 3.0\}$, $a_4 = \{0, 3.6\}$, $a_5 = \{1, 2.6\}$, $a_6 = \{0.2, 1.8\}$ y $a_7 = \{0.4, 3.3\}$, calculando q con la ecuación 3.10 y t_q^i con la ecuación 3.11 podemos usar p para asociar cada valor de umbral para asociarlo a su correspondiente valor numérico.

A continuación se describe de manera general el proceso. Del vector p obtenido y mostrado en la figura 3.2 se toma el primer elemento del arreglo y dado que es un atributo numérico, se calcula su valor correspondiente para t^i como sigue: el primer término de 3.11 se refiere al valor del límite inferior del dominio de éste primer atributo numérico, en el ejemplo tiene el valor 1.2. x_j^{min} y x_j^{max} son los valores mínimo y máximo de y^i del individuo i-ésimo del cual se está construyendo el árbol de decisión, en el ejemplo $x_j^{min} = 1.1$ y $x_j^{max} = 9.9$. Sustituyendo estos valores en 3.11 se tiene:

$$t_i^i = (1.2) + \frac{(5.7 - 1.1) * ((2.5) - (1.2))}{9.9 - 1.1} = 1.9 \tag{3.13}$$

y así sucesivamente se calculan los restantes valores de t^i

2) Etapa 2: Construcción del árbol de decisión parcial. Para la construcción del árbol parcial se emplea un vector auxiliar al que se llamará w^i que servirá para colocar tanto los atributos, como sus valores respectivos cuando se trate de atributos numéricos. Debido a que puede contener atributos categóricos y numéricos el número de nodos sucesores b se puede calcular mediante el dominio de los atributos usados en la condición de prueba utilizando la siguiente ecuación 3.14:

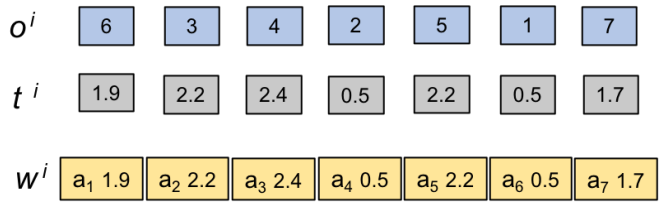

Figura 3.5: Ejemplo de construcción del vector w^i

$$b = \begin{cases} 2 & \text{si } D(\alpha) \subseteq \mathbb{R} \\ |D(\alpha)| & \text{en otro caso} \end{cases} \tag{3.14}$$

Para evitar que el árbol posea nodos redundantes se aplican las siguientes reglas en el momento de realizar la construcción:

- Regla 1: Un atributo categórico solo puede ser evaluado una sola vez en cada rama del árbol
- Regla 2: Un atributo numérico se puede evaluar varias veces en una misma rama del árbol, siempre y cuando éste utilice un valor de umbral coherente
- Regla 3: Los nodos sucesores de un nodo interno con dos ramas no pueden usar el mismo atributo categórico.

Si un elemento del vector w^i no satisface una de las anteriores reglas, no se usa para crear el árbol.

La figura 3.5 muestra el procedimiento para la construcción de w^i a partir de o^i y de t^i. En ella se puede apreciar como el primer atributo es numérico, por lo que se relaciona con el primer elemento del vector t^i, el segundo atributo es categórico, por lo que se coloca en la segunda posición del vector, y así sucesivamente.

La figura 3.6 muestra el proceso de construcción del árbol parcial a partir de w^i. Para su construcción se toman en cuenta las reglas previamente descritas. Se toma cada elemento de w^i y se verifica si cumple cada una de las reglas, en nuestro ejemplo se toma el primer elemento, que sería el nodo raíz del árbol se coloca en el árbol, como es un atributo numérico solo puede tener dos ramas, se coloca el segundo elemento de vector y se verifica que es un atributo categórico, tendrá tres nodos sucesores, debido a que el dominio de los valores del atributo es de tres y pasa la prueba de las tres reglas. Se puede apreciar del vector w^i que los elementos w_5^i no cumple la regla 1, debido a que ya se consideró el elemento w_2^i como el nodo raíz del árbol, por lo tanto no se considera como un nodo del árbol. El elemento w_4^i no cumple la regla 2, ya que $a_1 > 1.9$ no es congruente con $a_1 > 0.3$ por lo que no se

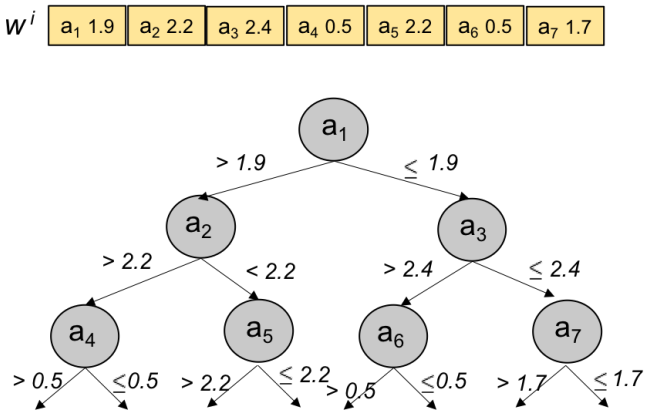

Figura 3.6: Ejemplo de construcción del vector w^i

considera como nodo del árbol. El elemento w_6^i si cumple la regla 2 ya que $a_3 > 1.3$ sí es congruente con $a_3 > 1.6$. Para el último elemento se cumple la regla 2, ya que $a_1 > 1.9$ es congruente con $a_1 > 1.9$.

3) Etapa 3: Construcción del árbol de decisión completo. Para construir el árbol de decisión completo se utiliza un mecanismo de mapeo usando el conjunto de datos del que se desea construir el árbol. El Algoritmo 6 muestra el proceso de construcción, mientras que la figura 3.7 describe la construcción del árbol del ejemplo que se ha usado en este documento. En esta etapa se agregan varios nodos hoja al árbol parcial pT^i al evaluar el conjunto de entrenamiento. Un conjunto de entrenamiento ϕ es asignado a un nodo interno ω (ι al nodo raíz), y al evaluar cada elemento en ϕ con la condición de prueba en el nodo ω se crean dos subconjuntos y se asignan a los nodos sucesores de ω. Se consideran dos casos:

- Si ω es localizado al final de la rama de pT^i, entonces dos nodos son creados y son designados como nodos hoja de ω. Cada subconjunto de ejemplos es asignado a cada nodo creado, y cada nodo es etiquetado como nodo hoja usando como su etiqueta de clase aquella que tiene el número más grande de ocurrencias en el subconjunto de ejemplos asignado a él.

- Si el número de ejemplos asignados a ω es menor que el número previamente definido al valor de umbral τ, o si todas sus ejemplos asignados pertenecen a la misma clase, luego ω es etiquetado como un nodo hoja. La clase mayoritaria ς de esos ejemplos es asignada como la etiqueta de clase del nodo hoja, y sus nodos sucesores son removidos si ya existen.

Para explicar el proceso previamente descrito se retoma el ejemplo que se ha venido utilizando en este documento. Dado que el conjunto de datos

Algoritmo 6 Construcción de Árbol completo

function DTCompleto(pT^i, ι, τ) ▷ Entrada: Árbol parcial, conjunto de entrenamiento, valor umbral para asignar un nodo hoja
  $\alpha \leftarrow$ Atributo asignado a al nodo raíz de pT^i
  $\omega' \leftarrow \alpha, \iota$
  $V' \leftarrow \omega'$
  $E' \leftarrow \emptyset$
  $Q \leftarrow$ Cola vacía
  AgregarCola(Q, ω')
  mientras $!Q = \emptyset$ **hacer**
    $\omega \leftarrow$SacarCola(Q)
    $\alpha \leftarrow$ Atributo asignado a *omega*
    $\phi \leftarrow$ Conjunto de instancias asignados en ω
    $\varphi \leftarrow \mid \mathbb{N}^+(\omega \in V) \mid$ ▷ Número de nodos sucesores asignados a ω en pT^i
    $b \leftarrow$Número de nodos sucesores de ω usando 3.14
    $\Phi \leftarrow$ particionDeConjuntoInstancias(ϕ, α)
    para $j \in 1, ..., b$ **hacer**
      $\xi \leftarrow$ ClaseMayoritaria(ϕ^j)
      $\psi \leftarrow$ El número de instancias en ϕ^j con ξ como etiqueta de clase
      si $j \leq \varphi \wedge \mid \phi^j \mid \neq \psi \wedge \mid \Phi^j \mid > \tau$ **entonces**
        $\alpha \leftarrow$ El atributo usado por $\mathbb{N}_j^+(\omega)$
        $\omega_j \leftarrow (\alpha, \phi^j)$ AgregarCola(Q, ω_j)
      si no
        $\omega_j \leftarrow (\xi, \phi^j)$
      fin si
      $V' \leftarrow V' \cup (\omega_j)$
      $E' \leftarrow \cup(\omega, \omega_j)$
    fin para
  fin mientras
  $T^i \leftarrow (G(V', E'), \omega')$
  regresar T^i
fin function

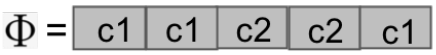

Figura 3.7: Conjunto de etiquetas de clase hipotético para el ejemplo

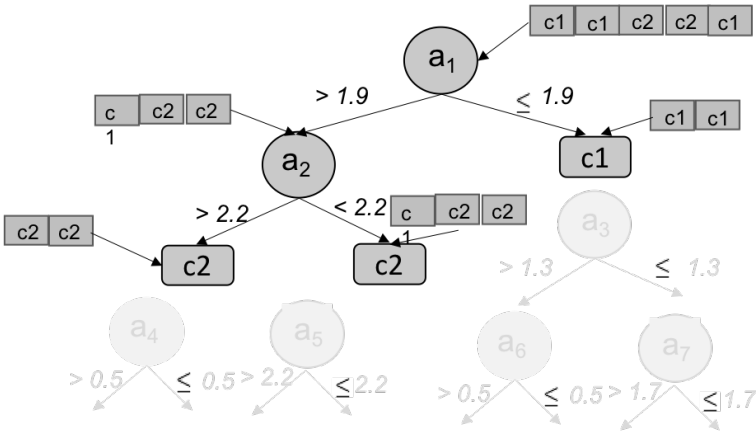

Figura 3.8: Conjunto de etiquetas de clase hipotético para el ejemplo

tiene dos etiquetas de clase, se utiliza el siguiente vector hipotético con las etiquetas de clase:

Este conjunto de datos se asocia al nodo raíz, se compara con la condición que se encuentra en este nodo y se divide en dos subconjuntos que se asignan a sus nodos sucesores, a su vez cada uno de ellos realiza el mismo procedimiento, en el caso de que en el subconjunto todos las etiquetas de clase sean iguales se reemplaza ese nodo por un nodo hoja y se coloca la etiqueta de esa clase mayoritaria. Para continuar con el proceso se siguen las reglas previamente mencionadas. En la figura 3.8 se muestra este procedimiento.

3.2.3. Método general de búsqueda de árboles de decisión paralelos a los ejes mediante ECA

En el presente trabajo de tesis se propone el siguiente método para generar árboles de decisión. La estructura del método ECA-ADT se muestra en el pseudocódigo del **Algoritmo 7**.

Inicialmente, el método lee el conjunto de datos, determina el número de atributos y el número de etiquetas de clase, calcula un tamaño estimado que tendrán los vectores solución. ECA genera una población aleatoria, evoluciona esta población y retorna el mejor individuo, con este individuo el método ECA-ADT construye el árbol parcial y posteriormente el árbol

competo. Se utiliza un proceso de refinamiento [1] del árbol y uno de poda [2], con el objetivo de quitar nodos redundantes y por ende un árbol más pequeño.

Algoritmo 7 Pseudocódigo para generar árboles de decisión paralelos a los ejes, método ECA-ADT

procedimiento ECA-ADT($DataSet, Umbral$) ▷ Input: Conjunto de datos, valor de umbral usado para asignar un nodo hoja
  $(a, c, e) \leftarrow$ *leer conjunto de datos*$(DataSet)$
  $d \leftarrow \mid a \mid$ ▷ Número de atributos
  $s \leftarrow \mid c \mid$ ▷ Número de etiquetas de clase
  $n_y \leftarrow$ *Calcular número estimado de nodos internos*
  $n_z \leftarrow$ *Calcular número estimado de umbrales*
  para cada $j \in \{1, ..., n_y\}$ **hacer**
    $n_z \leftarrow$ *Posición un atributo calculado usando* $p_j = j \ mod \ d$
  fin para
  $n_z \leftarrow n_y + n_z$
  $n^{mejor} \leftarrow ECA()$
  $w \leftarrow$ *Construcción de nodo vector*(x^{mejor})
  $Ap \leftarrow$ *Construcción de árbol parcial*(w)
  $A \leftarrow$ *Construcción de árbol completo*(Ap)
  $A \leftarrow$ *Refinamiento del árbol*(A)
  $A \leftarrow$ *Poda del árbol*(A)
fin procedimiento

[1] En esta tesis, el refinamiento es un procedimiento utilizado para mejorar el porcentaje de clasificación del mejor árbol obtenido por el proceso evolutivo (ECA), de tal manera que los nodos hoja que pudieran ser redundantes o que no estén clasificando correctamente sean reemplazados por sub-árboles.

[2] La poda es una técnica de compresión de datos en algoritmos de búsqueda y aprendizaje automático que reduce el tamaño de los árboles de decisión al eliminar secciones del árbol que no son críticas y son redundantes para clasificar instancias. La poda reduce la complejidad del clasificador final y, por lo tanto, mejora la precisión predictiva al reducir el sobreajuste.

Capítulo 4

Experimentos y Resultados

En este capítulo se describe el estudio experimental para analizar el desempeño del método ECA-ADT, objeto de la presente tesis. En primer lugar se proporciona una descripción de los conjuntos de datos usados en el proceso de experimentación. De la misma manera se definen los parámetros de ECA-ADT y el esquema de validación. Se describirán (1) el algoritmo $ECA\text{-}ADT_{SPV}$, el cual se adopta para efectos de comparación, (2) las medidas de desempeño utilizadas, y (3) las pruebas estadísticas para validar los resultados obtenidos. Se utiliza también el algoritmo clásico j48 para efectos de comparación en este capítulo.

4.1. Marco experimental

El algoritmo ECA-ADT se encuentra implementado en el lenguaje Java usando el framework jMetal [17]. Para validarlo se compara con el método $j48$ [79], cuya implementación en WEKA es el algoritmo C4.5, y con el método $DE\text{-}ADT_{SPV}$ con base en la validación estadística de los resultados de las ejecuciones y contrastando las diferencias encontradas entre ellos. La prueba estadística no-paramétrica a utilizar será Friedman [27], [28].

En la Tabla 4.1 se describen los parámetros usados en los experimentos.

El tamaño de población del algoritmo ECA-ADT se determina en función del número de elementos que integrarán los subconjuntos U y la dimensión del vector solución x^i. Para el caso del factor de mutación del método $DE\text{-}ADT_{SPV}$, su valor se irá decrementando linealmente desde un valor inicial de 1.0 a uno final de 0.3, mientras que el factor de cruza se mantendrá fijo en 0.9. Se puede apreciar que el valor de cruza es alto, lo que favorece a una buena capacidad de explotación del algoritmo y de manera simultánea permite una gran diversidad de las soluciones. Por otro lado, si en etapas tempranas del proceso evolutivo el factor de escala de mutación es alto, entonces también se favorece la exploración del espacio de búsqueda debido a que los vectores mutados se pueden localizar en diferentes áreas del espacio

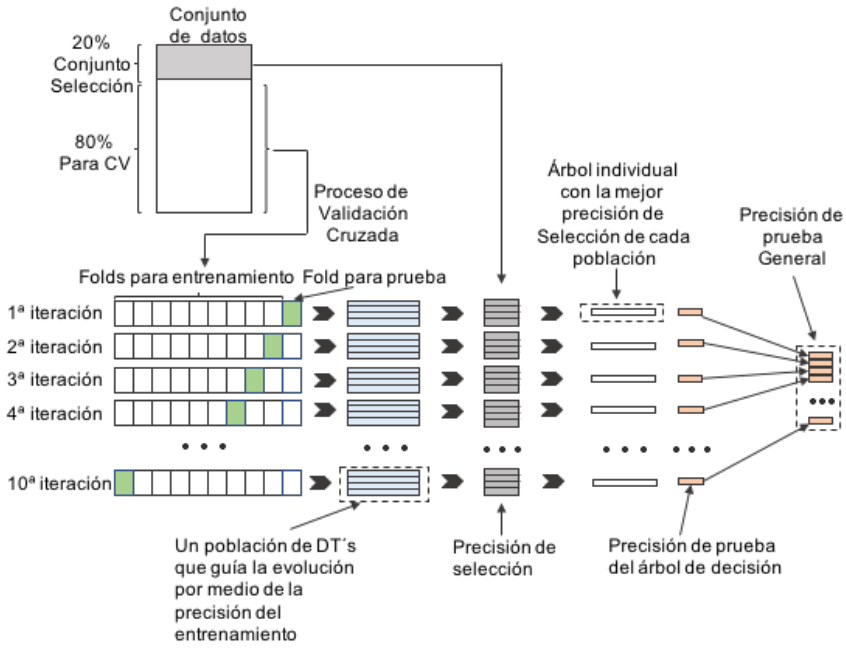

Figura 4.1: Proceso de CV 10-Fold implementado en ECA-ADT [61]

de búsqueda. A medida que el proceso avanza, los vectores se acercan entre sí, de tal manera que un factor reducido permite una mejor explotación de las áreas prometedoras del espacio de búsqueda.

Parámetro	DE-ADT	ECA-ADT
Generaciones	100	100
Valor de aptitud	Accuracy	Accuracy
Número de vecinos U		K = 7
Dimensión del vector solución		D
Paso		$\eta_{max} \leq 5.6184$
Tamaño de población	$250 \leq 5n \leq 250$	2K * D
Factor de escala de mutación	[0.3,1.0]	
Porcentaje de cruza	0.9	
Método de poda	Poda basada en error	Poda basada en error
Valor de umbral usado para etiquetar los nodos hoja	2 instancias	2 instancias

Tabla 4.1: Parámetros usados en los experimentos. Los parámetros de DE-ADT son los sugeridos por sus autores. Los parámetros de ECA-ADT fueron calibrados mediante la herramienta IRACE

Se adopta la sugerencia de Storn y Price [72] de utilizar el tamaño de la

población ajustada a $5n$, pero dentro del intervalo de 250 y 500 individuos como límite inferior y superior, respectivamente, para asegurar que la población no sea muy pequeña como para no permitir el proceso de exploración del espacio de búsqueda, y de manera simultánea que no sea tan grande como para afectar el tiempo de ejecución del algoritmo.

Las funciones de aptitud que se usan en los algoritmo tanto de ECA-ADT como de *DE-ADT*$_{SPV}$ calculan la precisión de entrenamiento de cada árbol de decisión en la población.

Por otro lado, debido a que tanto los métodos ECA-ADT y *DE-ADT*$_{SPV}$ obtienen los mejores árboles de decisión por medio de una búsqueda global, los árboles obtenidos se refinan con un procedimiento recursivo, el cual consiste de una poda para evitar posibles sobreajustes. Se aplica el enfoque de Poda Basado en error (EBP) [66], el cual produce árboles de decisión con una precisión mejorada usando únicamente el conjunto de entrenamiento.

Con el objetivo de obtener resultados confiables del rendimiento predictivo y asegurarse de que el modelo obtenido sea robusto, se hará uso de la técnica de validación cruzada 10-fold Cross Validation, la cual permite estimar el error de predicción y probar la precisión de ambos clasificadores que se van a comparar. En esta técnica de validación los conjuntos de datos se dividen aleatoriamente en diez subconjuntos disjuntos (folds) del mismo tamaño. Por cada $k \in \{1, 2, \ldots, 10\}$, el *k-ésimo* subconjunto es usado como el conjunto de prueba, éste se almacena, y los restantes son usados para inducir un árbol de decisión. Una vez que el árbol de decisión fue obtenido, el subconjunto que se encuentra guardado se utiliza para calcular la precisión (accuracy) del árbol construido. Este proceso se repite diez veces, una vez por cada fold, y finalmente cuando este proceso iterativo termina en la fase de inducción de los árboles, la precisión de la prueba general del modelo es calcula.

Tanto el método ECA-ADT como el *DE-ADT*$_{SPV}$ realizan una búsqueda global dentro del espacio de los árboles de decisión, por lo que usan la precisión del entrenamiento de cada árbol como el valor de aptitud (fitness) y existe la probabilidad de que los árboles inducidos estén sobreajustados al conjunto de entrenamiento, y entonces el árbol con mejor precisión en la fase de entrenamiento podría tener una precisión baja en la fase de prueba. Para tratar este problema, se utiliza un subconjunto de instancias del conjunto de datos para determinar una precisión independiente por cada árbol de decisión en la población final y para seleccionar el mejor entre ellos. A este valor Rivera en su tesis doctoral [61] le llama precisión de selección. De esta manera, el árbol de decisión con la mejor precisión de selección que se encuentre en la población final se utiliza para calcular la precisión de la prueba del subconjunto o fold. Para implementar esta estrategia, el 20 % de los ejemplos en el conjunto de datos se utilizan para calcular la precisión de la selección, y el resto se utilizan en el procedimiento de Validación Cruzada (CV, por sus siglas en inglés), ver figura 4.1

En el presente trabajo se toma el mismo procedimiento usado por Rivera-López y Canul-Reich, el cual está basado en el método propuesto por Murthy *et al.* [46] que consiste en que, por cada subgrupo (fold) se calcula la precisión de selección de cada árbol de decisión en la población, y el árbol de decisión con la mejor precisión de selección se utiliza para calcular el número ejemplos de prueba correctamente clasificados. El promedio que existe entre las clasificaciones correctas de todos los folds y el número de ejemplos o instancias de entrenamiento es la precisión de prueba general del clasificador, y el tamaño del árbol de decisión es el promedio de los nodos hoja del árbol construidos por todos los folds.

4.2. Selección de datos de prueba

Para realizar el estudio experimental se han utilizado un grupo de 20 conjuntos de datos (datasets), seleccionados del repositorio de aprendizaje automático UCI [39]. Todos estos conjuntos de datos han sido elegidos debido a que poseen atributos tanto numéricos, categóricos, o bien, una combinación de ambos. Así mismo, sus instancias son clasificadas en dos o más clases, y la mayoría de ellos son conjuntos de datos no equilibrados, lo que los hace ideales para ser utilizados en el estudio experimental. Estos datos se muestran en la Tabla 4.2.

En el presente estudio se aseguró de que los conjuntos de datos no contengan valores nulos, lo anterior con el objetivo de asegurar que tanto los resultados obtenidos por ECA-ADT, así como por $DE\text{-}ADT_{SPV}$ se puedan comparar y no se vean afectados por el tratamiento de los datos.

4.3. Sintonización de parámetros de ECA-ADT

El propósito de la presente sección es mostrar, de manera breve, los enfoques principales para configurar los parámetros de un algoritmo evolutivo. Se describe el marco que hay detrás de los métodos para calibrar sus parámetros. Se mencionan algunos enfoques que se encuentran en la literatura especializada para realizar esta tarea, y debido a que en el presente trabajo utiliza el método de calibración de parámetros, se pone especial interés en este enfoque, haciendo énfasis en el uso de la herramienta IRACE para la calibración automática.

4.3.1. Sintonización de parámetros para el análisis de algoritmos evolutivos

El tema de la configuración de parámetros en un algoritmo evolutivo es de suma importancia en su diseño, pues su buen desempeño depende fuertemente de los valores que tengan precisamente sus parámetros.

Conjunto de datos	Instancias	Atributos numéricos	Atributos categóricos	Clases	Distribución de clase
Conjunto de datos sólo con atributos numéricos:					
australian	690	14	0	2	307|383
balance-scale	625	4	0	3	288|49|288
glass	214	9	0	7	70|76|17|0|13|9|29
heart-statlog	270	13	0	2	150|120
ionosphere	351	34	0	2	126|225
iris	150	4	0	3	50|50|50
liver-disorder	345	6	0	2	145|200
page-blocks	5473	10	0	5	4913|329|28|88|115
pima-diabetes	768	8	0	2	500|268
sonar	208	60	0	2	97|111
vehicle	846	18	0	4	212|217|218|199
wine	178	13	0	3	59|71|48
Conjuntos de datos solo con atributos categóricos:					
car	1728	0	6	4	1210|384|69|65
molecular-p	106	0	57	2	53|53
tic-tac-toe	958	0	9	2	332|626
Conjuntos de datos tanto con atribtos numéricos y categóricos:					
cmc	1473	2	7	3	629|333|511
credit-g	1000	7	13	2	700|300
dermatology	366	1	33	6	112|61|72|49|52|20
haberman	306	2	1	2	225|81
lymph	148	3	15	4	2|81|61|4

Tabla 4.2: Descripción de los conjuntos de datos usados para la experimentación. [61]

La tarea de configurar parámetros no es una tarea trivial, existen un número grande de opciones posibles y muchas veces no se tiene conocimiento del efecto que puedan tener los parámetros en el desempeño del algoritmo. Más aún, dependiendo del dominio del problema que se pretende resolver con este algoritmo, los valores de los parámetros podrían cambiar, de tal manera que diferentes problemas podrían tener una configuración diferente de sus parámetros, en otras palabras, la configuración que funciona bien para un tipo de problemas podría no ser eficiente para otro tipo de problema.

Eiben *et al.* [20] dividen la configuración de parámetros en dos clases: **control de parámetros** y **calibración de parámetros**. Algunos autores también les llaman *en línea* y *fuera de línea*, respectivamente [7]. La primera

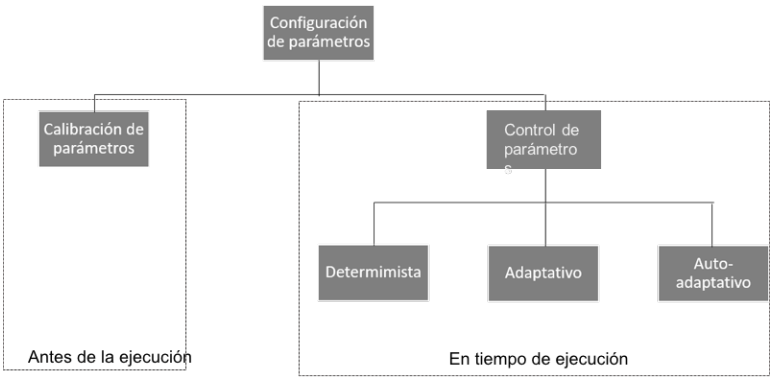

Figura 4.2: Enfoques para la configuración de parámetros

clase se refiere a que los valores de los parámetros cambian durante la ejecución del algoritmo, por lo que se necesitan valores iniciales para cada uno de ellos, así como algún mecanismo de control adecuado, de tal manera que le permitan al algoritmo adaptarse automáticamente ante cualquier cambio que ocurra en el problema. Por otro lado, la calibración de parámetros requiere de un proceso previo de ejecución del algoritmo y así encontrar valores adecuados para los parámetros que se mantienen fijos durante la ejecución final del algoritmo.

Por otro lado, es común encontrar en la literatura una clasificación de las estrategias para el control de parámetros en tres enfoques: **determinista**, **adaptativo** y **auto-adaptativo** [19].

En la presente investigación se opta por la calibración de parámetros como método para sintonizar los parámetros de ECA-ADT. Por ende, se explica con más detalle el marco conceptual que existe detrás de este enfoque.

Uno de los principales retos que enfrenta el diseñador de un algoritmo evolutivo son los valores de los parámetros que debe tener, lo anterior debido a la gran influencia que tienen sobre el desempeño del algoritmo. Si el investigador se toma el tiempo para realizar una buena calibración, el algoritmo puede producir resultados de buena calidad, frente a uno del cual no se tuvo el cuidado de elegir buenos valores para sus parámetros. El proceso de calibración de parámetros puede ser computacionalmente costoso, pero el diseñador debe llevarlo a cabo o de lo contrario su trabajo podría ser deficiente y poco confiable.

La calibración de parámetros puede visualizarse como un problema de optimización. Para comprender las similitudes entre la optimización de un problema y en calibración de parámetros se pueden distinguir dos momentos: uno cuando se pretende optimizar una solución y el otro cuando se pretende optimizar un grupo de parámetros. En Eiben *et al.* [20], con el objetivo de mostrar el proceso de calibración de parámetros y la similitud que existe con la optimización de problemas, identifican tres capas: la capa de algoritmo,

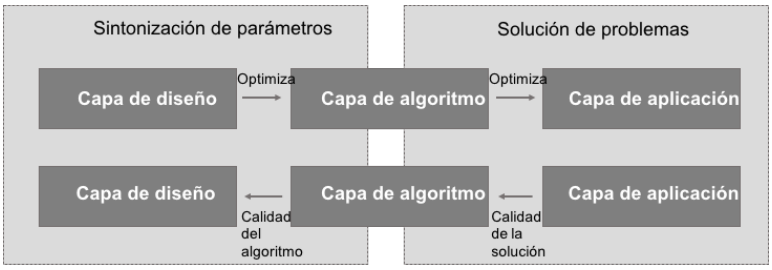

Figura 4.3: Flujo de control (arriba) y flujo de información (abajo) a través de las tres capas en la jerarquía de calibración de parámetros

capa de diseño y capa de aplicación, ver Figura 4.3.

Simultáneamente, dividen el esquema general en dos problemas de optimización: (1) solución de problemas y (2) sintonización de parámetros. La primera división consiste de un problema que se desea resolver en la capa de aplicación con un algoritmo evolutivo en la correspondiente capa de algoritmo. En el contexto del problema a resolver, como con cualquier problema de optimización, se utilizan los componentes de un EA, es decir, el método de trabajo es un algoritmo evolutivo, el espacio de búsqueda es el espacio de los vectores solución, y existe una función que mide la calidad de una solución llamada función de aptitud, y la valoración se realiza mediante evaluaciones de esta función de aptitud con las soluciones encontradas.

Por otro lado, en el contexto de la sintonización o calibración de parámetros, en lugar de usar un algoritmo evolutivo como método de trabajo, se utiliza un procedimiento de calibración donde en el espacio de búsqueda se encuentran vectores de parámetros; para medir la calidad de estos vectores se adopta el término utilidad; y para la valoración se utilizan pruebas.

Entonces, cuando se pretende encontrar la solución al problema, en la división que comprende a la solución de problemas, el EA iterativamente genera soluciones candidatas y trata de buscar una con la máxima calidad. Lo anterior corresponde, en la parte de la calibración de parámetros, a un método de calibración que intenta encontrar los valores más adecuados de los parámetros para el algoritmo evolutivo en la capa del algoritmo. De manera similar a la parte de la solución de problemas, el método de calibración iterativamente genera vectores de parámetros candidatos y busca aquel que tenga la máxima calidad, pero a diferencia de la parte de solución de problemas, la calidad del vector de parámetros está basada en el rendimiento del EA usando los valores en dicho vector.

En la literatura especializada sobre la calibración de parámetros se puede encontrar una gran variedad de métodos. Nuevamente, retomando a Eiben y a sus colegas, proponen una clasificación basada a su vez en tres taxonomías:

- *Taxonomía* T_1 Los métodos de esta categoría pueden ser de dos tipos: *calibradores no iterativos e iterativos*. Para los calibradores no itera-

tivos la etapa de generar vectores de parámetros se ejecuta una sola vez (al inicio del algoritmo), por lo tanto, se crea un conjunto fijo de vectores. En la etapa de prueba cada uno de estos vectores se prueba con el objetivo de encontrar el mejor, el proceso es simple, se inicializa y se prueba. La inicialización de los vectores se puede hacer aleatoria. Ejemplos de este tipo de métodos se pueden encontrar en Latin-Square [48] y Taguchi Orthogonal Arrays [73]. Por otro lado, la segunda categoría incluye aquellos algoritmos calibradores que no mantienen fijo el conjunto de vectores de parámetros iniciales. Estos métodos crean un pequeño conjunto de vectores iniciales y después crean nuevos durante la ejecución. Ejemplos de este tipo de métodos se pueden encontrar en Greffenstette [30], y FocusedILS [34], entre otros.

- *Taxonomía* T_2. En esta clasificación se encuentran los métodos que se enfocan en el esfuerzo de la búsqueda, es decir, tratan de encontrar vectores de parámetros de buena calidad con el menor esfuerzo posible. Un buen algoritmo de calibración de parámetros entonces, es aquel que realiza el menor esfuerzo posible para encontrar el vector de parámetros de buena utilidad.

  Teniendo como base el esfuerzo, se pueden dividir los algoritmos que pertenecen a esta clasificación en cuatro categorías: aquellos métodos que enfocan sus esfuerzos en minimizar el número de vectores de parámetros a ser probados en el EA, por ejemplo, meta-GA [30] y el método REVAC [49]. En la segunda categoría se encuentran los algoritmos que pretenden minimizar el número de ejecuciones del EA al que se le calibran sus parámetros, ejemplo de este tipo pueden ser ANOVA [68], y racing [42]. Existen también métodos que intentan minimizar ambos esfuerzos, ejemplos de estos ser: Sequential Parameter Optimization [6] y REVAC++ [70].En la tercera categoría se encuentran los métodos que intentan reducir el número de evaluaciones de la función de aptitud por el EA al que se le pretende calibrar sus parámetros. En la cuarta categoría se pueden agrupar los método que intentan minimizar tanto el número de vectores de parámetros, así como ahorrar el número de pruebas.

- *Taxonomía* T_3. Existen varios indicadores que pueden usarse para evaluar a los algoritmos evolutivos, entre ellos se pueden mencionar las medidas de rendimiento del algoritmo y también su robustez. De este último indicador se puede dividir en aquel que mide los cambios en la especificación del problema y en otro que mide los cambios en los valores de los parámetros. Los métodos que pertenecen a esta taxonomía no sólo son capaces de encontrar los mejores valores para los parámetros, sino que también proporcionan información sobre los diferentes indicadores que permiten medir el rendimiento del algorit-

mo. Los métodos que pertenecen a esta taxonomía tienen dos tareas principales. La primera se puede considerar como una actividad de explotación, ya que debe encontrar el vector de parámetros con la más alta calidad de rendimiento posible explotando una zona prometedora del espacio de los parámetros. La segunda tarea se considera una actividad de exploración del espacio de los parámetros y su función es adquirir información sobre la robustez. Cada método tiene un equilibrio diferente entre exploración y explotación, y se enfocan en diferentes tipos de robustez. Ejemplos de estos métodos se pueden encontrar en REVAC[49], I/F-RACE [9] y CALIBRA [1].

4.3.2. Método IRACE para calibración automática de parámetros de algoritmos evolutivos

En la sección previa se mencionó que los diferentes métodos que existen para calibrar parámetros de un AE se pueden clasificar en tres taxonomías. El método que se utiliza en el presente trabajo de tesis para la calibración automática de ECA-ADT se llama IRACE y pertence a la taxonomía T_3.

La configuración automática de algoritmos se puede describir desde la perspectiva del aprendizaje automático, como el problema de encontrar una buena configuración de parámetros para resolver instancias no conocidas o no vistas mediante el aprendizaje de conjunto de instancias ya conocidas en un conjunto de entrenamiento [8]. De aquí se pueden distinguir dos fases: la fase de calibración, en esta fase se elige un algoritmo, dado un conjunto de ejemplos de entrenamiento; y, una fase de prueba, donde se utiliza el algoritmo de configuración para resolver un conjunto de ejemplos del mismo problema, no conocidos por el algoritmo. El objetivo de la configuración automática es encontrar durante la fase de calibración una configuración del algoritmo que minimice alguna medida de costo sobre un conjunto de ejemplos que serán conocidos durante la fase de producción [41].

El paquete IRACE es una implementación del método racing iterado, del cual I/F-RACE [9] es un caso especial que usa el análisis bidireccional no paramétrico de la varianza de Friedman por ranking. El racing iterado es un método de configuración automática de tres pasos: (1) muestrear nuevas configuraciones de acuerdo a una distribución particular, (2) seleccionar las mejores configuraciones de las nuevas muestras por medio de carreras o racing, y (3) actualizar la distribución de los ejemplos en función de los sesgos de las mejores configuraciones. Esos pasos se repiten hasta que se alcanza un criterio de minimización.

No. Iteración	K	η_{max}
Primera iteración:		
Mejor configuración	8	7.3181
Peor configuración	5	4.2486
Segunda iteración:		
Mejor configuración	7	5.6184
Peor configuración	4	6.7066
Tercera iteración:		
Mejor configuración	7	5.6184
Peor configuración	4	6.7066

Tabla 4.3: Valores de los parámetros para ECA-ADT obtenidos en la calibración automática con IRACE

El racing iterado es implementado en la herramienta IRACE. Cada parámetro que se desea configurar tiene asociado una distribución de muestreo independiente de los otros parámetros, además de las restricciones y condiciones entre éstos.

El proceso de racing se puede describir como una competencia de carreras, la cual inicia con un conjunto de configuraciones candidatas. En cada paso (iteración del algoritmo) de la carrera las configuraciones son evaluadas en cada instancia I_j, después de un número determinado de pasos, aquellas configuraciones que no tuvieron una buena calidad son eliminadas, la carrera continúa con las configuraciones sobrevivientes.

La actualización de las distribuciones consiste en modificar tanto la media como la desviación estándar en la distribución normal, o los valores de las probabilidades de las distribuciones discretas. Las actualizaciones sesgan las distribuciones de tal manera que se incrementa, para futuras iteraciones, la probabilidad de la muestra para los valores de los mejores valores de los parámetros encontrados hasta el momento.

4.3.3. Resultados obtenidos con IRACE

Para calibrar los parámetros de manera automática con IRACE, se ha tomado un conjunto de 5 datasets o conjuntos de datos representativos: iris, ionosphere, vehicle, glass y dermatology. Se han elegido estratégicamente debido a sus características (ver Tabla 4.2), con los objetivos tanto de minimizar el tiempo de ejecución, así como de la obtención de resultados que pudieran funcionar para todos los datasets que se listan en dicha tabla.

Así mismo, se utilizó un equipo de cómputo con las siguientes especificaciones: CPU 2 x 3.8 GHz, Memoria RAM 8 GB y Sistema operativo basado en Linux UBUNTU 18.

Se han utilizado tres iteraciones de IRACE y rangos de valores para $3 \leq K \leq 10$ y $0.1 \leq \eta_{max} \leq 8$.

En la Tabla 4.3 se resumen los resultados obtenidos por el paquete IRACE.

Pese a haber obtenido con la herramienta IRACE los valores tanto para η_{max} como para K que se muestran en la Tabla 4.3, se toma la decisión de utilizar los valores de $K = 3$ y $\eta_{max} = 1$. Estos valores se obtuvieron en base a la experiencia de manera manual, dado que las limitaciones de cómputo para ejecutar la herramienta no eran las adecuadas para calibrar con un mayor número de conjuntos de datos. Al realizar los experimentos se observó que estos valores producían mejores resultados que las obtenidas en las primeras iteraciones de IRACE.

4.4. Diseño experimental

En el presente estudio experimental del trabajo de tesis, cuyo objetivo principal es el de validar el método propuesto ECA-ADT, se aplica la prueba no paramétrica Friedman [27, 28] con 95 % de confianza, y de esta manera realizar el análisis estadístico de los resultados que se obtuvieron de las ejecuciones del método sobre los 20 conjuntos de datos descritos en la Tabla 4.2. Las medidas de desempeño del algoritmo serán el promedio de clasificación de los árboles obtenidos (accuracy), así como su respectivo promedio de tamaño (size).

La prueba estadística no paramétrica Friedman se utiliza para comparar simultáneamente varios algoritmos del estado del arte, versus el algoritmo que se desea validar.

De manera general, lo que hace esta prueba es asignar rangos (rankings) r_{ij} a los resultados que se obtuvieron de los experimentos al algoritmo j del conjunto de algoritmos de tamaño k que se desean comparar y que fueron aplicados para resolver cada problema i. En otras palabras, para cada problema, se le asigna un rango que va de $r_j^i (1 \leq r_{rj} \leq k)$. Los rangos se asignan de manera ascendente a cada resultado, es decir, se le asocia a un valor de 1 a la mejor observación, 2 a la segunda, y así sucesivamente. Si dos observaciones tienen el mismo valor, entonces se asigna el valor del promedio de los rangos.

Una vez identificada cada observación y asignada su valor de rango, se realiza el cálculo de los rangos medios de los algoritmos tomando en cuenta los n problemas, es decir, se aplica la Ecuación 4.1, de esta manera clasifica los algoritmos para cada problema por separado, el algoritmo que tiene el mejor rendimiento tiene el rango 1, el segundo mejor el 2, y así sucesivamente.

$$R_j = \frac{\sum_{i=1}^{n} r_{ij}}{n} \tag{4.1}$$

Sean H_0 y H_1 las hipótesis nula y alterna respectivamente, entonces:

- H_0: Todos los algoritmos se comportan similarmente, por lo tanto sus rangos R_j deben ser similares.
- H_1: Todos los algoritmos no tienen comportamiento similar, por lo que sus rangos R_j no son similares.

Si el estadístico de Friedman sigue la hipótesis nula, que establece que todos los algoritmos tienen un comportamiento similar, es decir, todos sus rangos deberían ser iguales, el estadístico de Friedman se calcula con 4.2:

$$F_F = \frac{12n}{k(k-1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{4.2}$$

la cual se distribuye de acuerdo a una distribución χ^2 con $k-1$ grados de libertad.

En caso de que se rechace la hipótesis nula, entonces se puede proceder con un test *post-hoc* para encontrar diferencias a posteriori. Gracias a estas pruebas se podrán encontrar las diferencias que pudieran existir entre los algoritmos a comparar. Este grupo de algoritmos restantes están asociados a un grupo de hipótesis de comparación. El p-valor de cada hipótesis en el grupo (también llamadas p-valores desajustados) se puede obtener a través de la conversión de los rangos calculados por el test de Friedman usando una aproximación normal.

Para comparar el *i-ésimo* algoritmo versus el *j-ésimo* algoritmo, es decir, aproximar el estimador z de una distribución normal a partir de las diferencias que se encontraron entre dos rangos se debe utilizar la Ecuación 4.3.

$$z = (R_i - R_j)/\sqrt{\frac{k(k-1)}{6n}} \tag{4.3}$$

Donde R_i y R_j son los promedios de rankings o rangos obtenidos del la prueba de Friedman de los algoritmos comparados.

Después de haber calculado el valor z se podrá obtener un p-valor no ajustado. Estos procedimientos post-hoc son capaces de calcular un p-valor Ajustado considerando una familia de hipótesis. En el presente trabajo de tesis se aplica el test post-hoc de Nemenyi [50]. Esta prueba ajusta el valor de significancia α en un solo paso, al dividirlo por el número de algoritmos comparados [29]. El objetivo de utilizar el test de Nemenyi es analizar y detectar las diferencias significativas entre todos los pares existentes del conjunto de algoritmos que se están comparando. Para hacerlo, se ha implementado en la librería *scmamp* de R [14].

En el presente trabajo de investigación se utilizarán dos versiones del método ECA-ADT, con el objetivo de mejorar la precisión del árbol obtenido:

- **ECA-ADT**^{B}. Esta versión del algoritmo no implementa un proceso de refinamiento, únicamente retorna como resultado la precisión de selección y el tamaño del árbol generado, pero sin refinar los nodos hoja. Con este algoritmo se corre el riesgo de inducir árboles de decisión con nodos hoja que contengan más de una instancia en la etiqueta de clase, por lo que la clasificación no sería correcta en estos nodos.

- **ECA-ADT**^{R}. Esta variante da como resultado una versión refinada del árbol, además de la mejor precisión de selección en la población y su tamaño. Con este refinamiento se buscan nodos hoja que clasifiquen correctamente las instancias del conjunto de datos, en otras palabras, se busca que los árboles inducidos no contengan nodos hoja que tengan dos o tres instancias diferentes en una misma etiqueta de clase. Para lograr lo anterior se toma el mismo criterio que utiliza $j48$, se usa un valor umbral para determinar el número máximo de instancias con etiqueta de clase deferente a la predominante, WEKA [3] usa 2 o 3 como valor de umbral.

Estas dos versiones del método son comparadas con sus métodos homólogos $DE\text{-}ADT^B_{SPV}$ y $DE\text{-}ADT^R_{SPV}$ en [62] y con $j48$.

4.5. Resultados y pruebas estadísticas

En esta sección se describen los resultados obtenidos y se detallan las estadísticas que se emplearon para comparar ECA-ADT con otros métodos para inducir árboles de decisión.

Las observaciones pertenecen a los resultados obtenidos sobre un conjunto de diez ejecuciones realizadas en cada uno de los conjuntos de datos de la Tabla 4.2, mientras que las muestras de $j48$, $DE\text{-}ADT^B_{SPV}$ y de $DE\text{-}ADT^R_{SPV}$ se tomaron de [62]. Primero, el porcentaje de clasificación de los árboles obtenidos, son comparados con $j48$, $ECA\text{-}ADT^B$ y $ECA\text{-}ADT^R$. Se aplica el estadístico de Friedman, tal como se ha mencionado en la sección previa.

La Tabla 4.4 muestra los resultados de los promedios de los porcentajes de clasificación de los mejores árboles inducidos. Cada valor tiene asociado su respectivo valor del rango (en paréntesis) que le fue asignado para aplicar al método Friedman. En negrita se pueden apreciar los resultados de los métodos que obtuvieron mejores desempeños. En la última fila se puede apreciar el promedio de rankings de cada método evaluado.

Conjunto de datos	*J48*	*DE-ADT*$^{R}_{SPV}$	*DE-ADT*$^{B}_{SPV}$	*ECA-ADT*^{R}	*ECA-ADT*^{B}
australian	84.35(5)	86.42(3)	85.57(4)	**99.57**(1.5)	**99.57**(1.5)
balance-scale	77.82(4)	**80.00**(1)	75.70(5)	79.59(2)	78.06(3)
glass	67.62(4)	**71.67**(1)	67.82(3)	68.18(2)	66.16(4)
heart-statlog	78.15(5)	82.92(2)	**88.81**(1)	79.74(4)	79.99(3)
iris	94.73(5)	**96.95**(1)	95.92(4)	96.20(3)	96.53(2)
ionosphere	89.74(5)	**92.88**(1)	89.97(4)	91.71(2)	91.48(3)
liver-disorders	65.83(4)	**70.33**(1)	66.96(2)	66.15(3)	64.26(5)
page-blocks	**96.99**(1)	96.74(3)	95.10(4)	96.90(2)	94.60(5)
pima-diabetes	74.49(3)	**75.48**(1)	74.73(2)	73.07(5)	74.03(4)
sonar	73.61(4)	**78.74**(1)	74.19(2)	73.95(3)	72.65(5)
vehicle	**72.28**(1)	72.21(2)	65.49(5)	70.50(3)	68.51(4)
wine	93.20(5)	**94.37**(1)	93.47(4)	93.59(3)	93.71(2)
car	**92.22**(1)	90.53(2)	81.19(5)	83.92(3)	82.56(4)
molecular-p	79.06(5)	**84.88**(1)	80.12(2)	79.72(4)	80.00(3)
tic-tac-toe	**85.28**(1)	84.46(2)	75.49(5)	83.66(3)	78.13(4)
cmc	51.44(5)	**55.57**(1)	54.18(2)	52.99(4)	53.31(3)
credit-g	71.25(5)	**73.65**(1)	71.63(4)	72.23(3)	72.34(2)
dermatology	94.10(4)	95.70(3)	92.81(5)	**95.99(1)**	95.96(2)
haberman	72.16(5)	**75.76**(1)	74.29(2)	73.14(4)	73.72(3)
lymph	75.81(3)	**78.83**(1)	76.50(2)	74.26(5)	74.51(4)
Promedio de rangos	3.75	1.50	3.35	3.02	3.37

Tabla 4.4: Promedios de porcentajes de clasificación de los DT obtenidos por los métodos comparados para inducir árboles de decisión

Por otro lado, la Tabla 4.6 muestra los tamaños de los árboles con los mejores porcentajes de clasificación obtenidos. Similar a la tabla 4.4 de los porcentaje de clasificación, entre paréntesis se muestra el rango, en negrita los tamaños de los árboles más compactos, mientras que la última fila muestra los promedios de rangos calculados. En este contexto, se puede apreciar que los métodos DE-ADT como ECA-ADT, ambos sin refinamiento, obtienen los árboles más compactos y de la Tabla 4.4 se observa que la diferencia en sus promedios de rendimiento es muy pequeña, solo 0.02, por lo que se puede afirmar que tienen un comportamiento muy parecido.

Del mismo modo, se aprecia que el método con el enfoque basado en *ECA-ADT*^{R}, que posee el comportamiento similar a *j*48, ofrece la ventaja de inducir árboles más compactos en el 75 % de los datasets utilizados en los experimentos, incluso mejora el nivel interpretativo de los árboles del 25 % de los obtenidos con el mejor método (*DE-ADT*$^{R}_{SPV}$).

Se puede observar que los enfoques basados en metaheurísticas (DE y ECA en este caso) proporcionan mejores porcentajes de clasificación que el

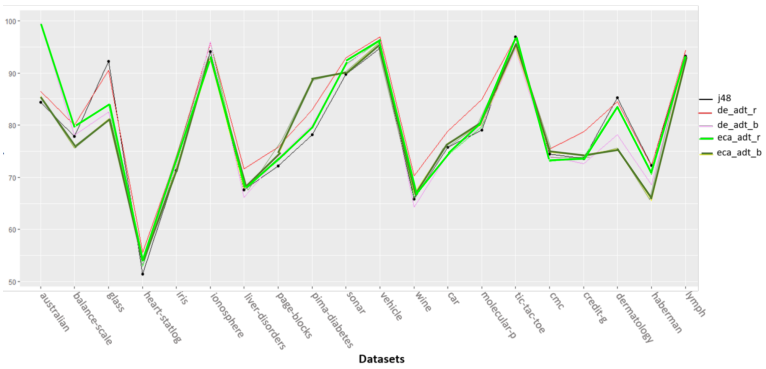

Figura 4.4: Gráfica descriptiva de los rendimientos de los cinco métodos comparados

algoritmo $j48$, que se tiene como referente para la solución de muchos problemas de clasificación. El método basado en DE con refinamiento, de acuerdo a los rangos de la prueba de Friedman, proporciona mejores resultados con respecto a porcentaje de clasificación (Tabla 4.4). El enfoque basado en la Física $ECA\text{-}ADT^R$, con un promedio de rango de 3.02, queda en segundo lugar en el presente estudio experimental y se puede apreciar que mejora el desempeño en el 70 % de los árboles inducidos en comparación con $j48$ y en un 60 % con respecto a la versión de DE sin refinamiento.

La gráfica de la Figura 4.4 muestra de manera visual los resultados de la Tabla 4.4.

El estadístico de Friedman calculado mediante el análisis de los resultados para los cinco métodos es 24.61, mientras que su correspondiente p-valor es 6.026 x 10^{-05}. Evaluando este p-valor con un nivel de significancia $\alpha = 5\,\%$ se puede afirmar con seguridad que la hipótesis nula H_0 es rechazada.

Después, el test de Nemenyi se aplica como prueba post-hoc para encontrar las posibles diferencias entre los métodos comparados y poder determinar si los métodos con enfoque en ECA son competitivos o no. La Tabla 4.5 muestra las diferencias de promedios de rangos entre pares de algoritmos que se están comparando, tal como está descrito en Demšar [15].

Método	$j48$	$DE\text{-}ADT^{R}_{SPV}$	$DE\text{-}ADT^{B}_{SPV}$	$ECA\text{-}ADT^{R}$	$ECA\text{-}ADT^{B}$
$j48$	0.000	2.250	0.400	0.275	0.375
$DE\text{-}ADT^{R}_{SPV}$	2.350	0.000	-1.850	-1.525	-1.875
$DE\text{-}ADT^{B}_{SPV}$	0.400	-1.850	0.000	0.325	-0.025
$ECA\text{-}ADT^{R}$	0.275	-1.525	0.325	0.000	-0.350
$ECA\text{-}ADT^{B}$	0.375	-1.875	-0.025	-0.350	0.000

Tabla 4.5: Diferencias de promedios de rangos entre los cinco métodos a comparar

El test indica que el desempeño de dos algoritmos que se están comparando es significativamente diferente si el promedio correspondiente a los rangos difiere en al menos la diferencia crítica obtenida por la Ecuación 4.4.

$$CD = q_\alpha \sqrt{\frac{k(k-1)}{6n}} \tag{4.4}$$

donde n es el número de datasets utilizados para realizar los experimentos, k es el número de algoritmos a comparar y α es el nivel de significancia.

De acuerdo a la tabla valores críticos para el test de dos algoritmos de Nemenyi en [15], para $\alpha = 0.05$ y k = 5 es: $q_{0.05} = 2.728$. Reemplazando este valor en la Ecuación (4.4) y realizando los cálculos correspondientes se obtiene $CD = 1.114$. Debido a que la diferencia entre el mejor y peor algoritmo es mayor a este valor, podemos decir que la prueba post-hoc tiene suficiente evidencia para detectar diferencias significativas entre los algoritmos comparados.

En el presente trabajo de tesis, se decidió, además, usar la prueba de Bonferroni-Dunn para reducir la tasa de error por experimento que se obtiene en pruebas de múltiples hipótesis, como en el del presente estudio experimental, y de esta manera reducir el porcentaje de obtener descubrimientos falsos (tasa de error familiar) que el test de Nemenyi no puede reducir. Para lograr controlar la tasa de error familiar Bonferroni-Dunn divide α por el número de comparaciones hechas ($k - 1$ en nuestro caso).

De acuerdo a lo anterior, y para poder determinar si el método con enfoque $ECA\text{-}ADT^{R}$ es mejor que el método $j48$ se utiliza el test de Bonferroni-Dunn tal como se aplica en [15], obteniendo el valor de $q_{0.05}$ de la tabla para el valor crítico para la prueba Bonferroni-Dunn en [15] se observa que $q_{0.05} = 2.498$ para 5 algoritmos. Nuevamente utilizando la Ecuación 4.4 se obtiene $CD = 1.019$. Debido a que la diferencia de promedios de rangos entre $j48$ y $ECA\text{-}ADT^{R}$ es menor a 1.019 no se puede rechazar H_0. Por lo que se puede decir con seguridad que estos dos enfoques poseen un comportamiento similar. Análogamente, al compararlo con el mejor algoritmo, se puede apreciar que la diferencia de promedios de rangos es mayor a CD, por

lo que la hipótesis nula se rechaza y se puede decir que $ECA\text{-}ADT^R$ y $DE\text{-}ADT^R_{SPV}$ tienen comportamientos diferentes, y se concluye que el algoritmo con enfoque en DE es mejor que el de enfoque en ECA.

Comparando $ECA\text{-}ADT^R$ versus $DE\text{-}ADT^B_{SPV}$ se puede apreciar que su diferencia también es menor a 1.019, por lo que tampoco se puede rechazar H_0, y por lo tanto, estos dos algoritmo poseen comportamientos similares, la balanza se inclina ligeramente hacia el método con enfoque en ECA.

Conjunto de datos	*J48*	*DE-ADT^R_{SPV}*	*DE-ADT^B_{SPV}*	*ECA-ADT^R*	*ECA-ADT^B*
australian	25.75(5)	13.54(4)	7.47(3)	**2.0**(1.5)	**2.0**(1.5)
balance-scale	41.60(5)	18.43(3)	10.04(1)	21.63(4)	**14.03**(2)
glass	23.58(5)	15.93(2)	8.88(1)	19.28(4)	**16.45(3)**
heart-statlog	17.82(5)	9.77(2)	9.60(1)	15.15(4)	**14.35**(3)
iris	4.64(3)	4.35(2)	**4.10**(1)	5.82(4)	5.87(5)
ionosphere	13.87(5)	**7.79**(1)	7.86(2)	12.68(4)	10.50(3)
liver-disorders	25.51(5)	14.29(3)	7.58(1)	20.20(4)	**8.98**(2)
page-blocks	42.91(5)	40.45(4)	8.04(2)	28.93(3)	**7.45**(1)
pima-diabetes	22.20(4)	18.55(3)	**6.71**(1)	32.81(5)	14.58(2)
sonar	14.45(3)	10.40(2)	**10.34**(1)	15.09(4)	16.71(5)
vehicle	69.50(5)	61.84(4)	**11.50**(1)	50.35(3)	33.78(2)
wine	**5.30(1)**	5.25(2)	5.39(3)	7.16(4)	7.49(5)
car	122.05(5)	105.33(4)	15.57(1)	63.49(3)	**25.37**(2)
molecular-p	16.90(5)	11.62(2)	**11.29(1)**	15.55(4)	15.40(3)
tic-tac-toe	88.04(5)	79.80(4)	27.12(1)	69.38(3)	**31.96**(2)
cmc	149.75(5)	38.87(3)	15.61(1)	77.63(4)	**29.61**(2)
credit-g	90.18(5)	55.57(3)	35.67(1)	62.53(4)	**47.10**(2)
dermatology	27.06(5)	24.07(3)	**19.14**(1)	24.29(4)	22.68(2)
haberman	15.32(5)	**9.09**(1)	10.16(2)	14.56(4)	10.72(3)
lympph	17.30(5)	13.14(2)	**13.01**(1)	18.68(4)	16.40(3)
Promedio de rangos	4.65	2.70	1.4	3.75	2.67

Tabla 4.6: Promedios de los tamaños de DT obtenidos de los métodos para inducir árboles de decisión comparados

La Figura 4.5 muestra gráficamente lo tamaños obtenidos por los métodos comparados.

En el presente capítulo se presentan los resultados obtenidos del experimento realizado con el método propuesto ECA-ADT. Tal como se puede apreciar, pese a haber obtenido resultados competitivos, los promedios de rangos de porcentajes de clasificación obtenidos al comparar los cinco métodos, muestran poca diferencia entre ellos (Tabla 4.4). Para poder validar si estos resultados tienen diferencias estadísticas significativas y de esta mane-

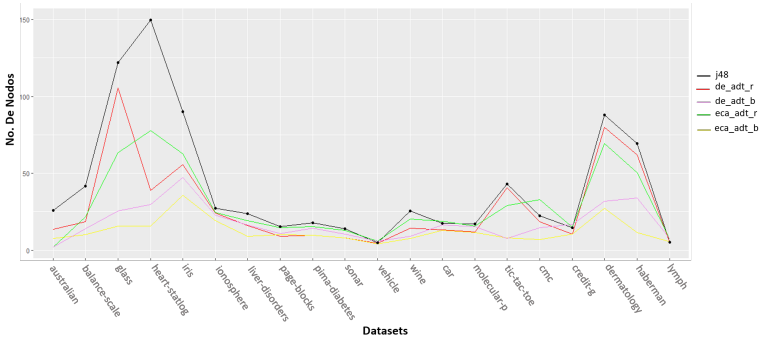

Figura 4.5: Gráfica descriptiva de los tamaños de los árboles obtenidos por cada uno de de los cinco métodos comparados

ra poder afirmar que los métodos $j48$, $DE-ADT^{B}_{SPV}$, y ambas versiones de $ECA-ADT$ son diferentes y alguno resulta mejor que otro, fue necesario realizar pruebas post-hoc.

La prueba utilizada fue el test de Nemenyi y posteriormente Bonferroni-Dunn. Con los resultados obtenidos se pudo tener la certeza, de que los algoritmos tienen suficiente evidencia para poder detectar diferencias significativas. Dado que la diferencia en promedios de rangos entre el mejor método en esta comparación ($DE-ADT^{R}_{SPV}$) y el segundo mejor método ($ECA-ADT^{R}$) es significativamente suficiente para afirmar lo anterior. Ahora interesa comparar el segundo lugar, con el método que quedó en tercer lugar, es decir, $j48$, y poder afirmar que el método con enfoque evolutivo es mejor. Dado los resultados de las pruebas, no se puede rechazar la hipótesis H_0 entre éstos dos métodos, por lo que tienen un comportamiento similar.

A pesar de tener comportamientos similares $ECA-ADTSPV^{R}$ y $j48$, se puede observar de la Tabla 4.5 que el método con enfoque ECA obtiene árboles más compactos que $j48$, por lo que puede representar como su principal ventaja frente al algoritmo que se tiene como referente para la solución de muchos problemas de clasificación. Además, en algunos datasets mejora el porcentaje de clasificación, incluso al que quedó en primer lugar, por lo que se puede tomar como un método alternativo para la solución de problemas y con buenas probabilidades de mejorar los resultados que se podrían obtener con $j48$.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

En el presente trabajo de tesis se propuso un método alterno a los que se pueden encontrar el la literatura existente en el estado del arte, en el área de aprendizaje automático, para la inducción de árboles de decisión usando un enfoque inspirado en la Física, se utilizó el algoritmo ECA para realizar la búsqueda en el espacio de los árboles de decisión y poder generar árboles más compactos y con un competitivo porcentaje de clasificación.

Dado que ECA ha mostrado ser competitivo para la solución de problemas de optimización numérica [44], incluso mejorando los resultados obtenidos con algoritmos de referencia como DE en espacios continuos, este trabajo se dedicó a analizar su desempeño en un tipo particular de espacio discreto, como lo es el relativo a los árboles de decisión.

Los experimentos realizados sobre un conjunto de datasets demostraron que el método $ECA\text{-}ADT^R$ generó árboles de menor tamaño en comparación a aquellos obtenidos con el algoritmo j48, que se tiene como referente para inducir árboles de decisión. Por otro lado, $DE\text{-}ADT^B_{SPV}$ y $ECA\text{-}ADT^B$, que estadísticamente obtuvieron desempeños semejantes, fueron los métodos que obtuvieron los árboles más compactos, lo que favorece su interpretación.

Se observó que, si bien $ED\text{-}ADT^R_{SPV}$ induce árboles con mejor porcentaje de clasificación, $ECA\text{-}ADT^R$ ocupó el segundo lugar en este sentido. Además, en general, el ECA fue particularmente competitivo en conjuntos de datos de mayor tamaño y cuando estos poseen atributos tanto numéricos como categóricos, mejorando, incluso, algunos de los resultados obtenidos por aquellos basados en DE.

Después de los experimentos realizados, de los resultados obtenidos y de los análisis estadísticos aplicados, se puede validar la hipótesis de investigación planteada en el Capítulo 1, pues ECA logró inducir árboles de

decisión con similar precisión de clasificación, pero con un tamaño menor que aquellos generados por los algoritmos comparados. De hecho, ECA fue particularmente competitivo en conjuntos de datos grandes, con atributos tanto numéricos como categóricos.

5.2. Trabajo futuro

A continuación se mencionan algunas líneas para trabajos futuros que se derivan del presente trabajo de tesis:

- Realizar comparaciones con otras técnicas de aprendizaje automático que se encuentran en la literatura del estado del arte sobre la inducción de árboles de decisión, con el objetivo de determinar el desempeño de ECA.
- Estudiar el comportamiento de ECA en conjuntos de datos de mayor tamaño, evaluar el tiempo para la obtención de los clasificadores, así como el nivel de precisión.
- Analizar la capacidad del método de refinamiento, pues es probable que se pueda mejorar o utilizar otra técnica (tal vez otra metaheurística).
- Estudiar el proceso de refinamiento de los nodos hoja que no son necesarios, que fue utilizado en el presente trabajo de tesis, dado que fue utilizado $j48$ para realizarlo, se debe analizar si otro método puede mejorar los resultados obtenidos, esto con el objetivo de obtener árboles con mejor rendimiento de clasificación.
- Estudiar un enfoque multi-objetivo (clasificación VS tamaño) para inducir árboles de decisión.
- Así mismo, se deja como trabajo futuro la calibración del parámetro del tamaño de la población, pues en el presente trabajo de tesis se utilizó el valor recomendado por el autor del algoritmo ECA. De manera simultánea investigar si este parámetro influye o no para mejorar los resultados obtenidos, tanto en el tamaño, como el porcentaje de clasificación.

Bibliografía

[1] B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1):99–114, 2006.

[2] M. T. Ahvanooey, Q. Li, M. Wu, and S. Wang. A survey of genetic programming and its applications. *TIIS*, 13(4):1765–1794, 2019.

[3] V. Autores. Archivo situacionista hispano. urlhttp://sindominio.net/ash, 1999.

[4] Z. Bandar, H. Al-Attar, and D. McLean. Genetic algorithm based multiple decision tree induction. In *ICONIP'99. ANZIIS'99 & ANNES'99 & ACNN'99. 6th International Conference on Neural Information Processing. Proceedings (Cat. No. 99EX378)*, volume 2, pages 429–434. IEEE, 1999.

[5] R. C. Barros, M. P. Basgalupp, A. C. De Carvalho, and A. A. Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3):291–312, 2011.

[6] T. Bartz-Beielstein, K. E. Parsopoulos, M. N. Vrahatis, et al. Analysis of particle swarm optimization using computational statistics. In *Proceedings of the international conference of numerical analysis and applied mathematics (ICNAAM 2004)*, pages 34–37, 2004.

[7] R. Battiti, M. Brunato, and F. Mascia. *Reactive search and intelligent optimization*, volume 45. Springer Science & Business Media, 2008.

[8] M. Birattari and J. Kacprzyk. *Tuning metaheuristics: a machine learning perspective*, volume 197. Springer, 2009.

[9] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, et al. A racing algorithm for configuring metaheuristics. In *Gecco*, volume 2, 2002.

[10] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York, New York, NY, softcover

reprint of the original 1st edition 2006 (corrected at 8th printing 2009) edition, 2016. OCLC: 1031889079.

[11] A. Biswas, K. Mishra, S. Tiwari, and A. Misra. Physics-inspired optimization algorithms: a survey. *Journal of Optimization*, 2013, 2013.

[12] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees*. CRC press, 1984.

[13] F. V. Buontempo, X. Z. Wang, M. Mwense, N. Horan, A. Young, and D. Osborn. Genetic programming for the induction of decision trees to model ecotoxicity data. *Journal of chemical information and modeling*, 45(4):904–912, 2005.

[14] B. Calvo and G. Santafe. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, Accepted for publication, 2015.

[15] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.

[16] D. Dua and C. Graff. UCI machine learning repository, 2017.

[17] J. J. Durillo and A. J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[18] A. Ehrenfeucht and D. Haussler. Learning decision trees from random examples. *Information and Computation*, 82(3):231–246, 1989.

[19] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms*, pages 19–46. Springer, 2007.

[20] A. E. Eiben and S. K. Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[21] A. E. Eiben, J. E. Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[22] C. Estébanez, J. M. Valls, and R. Aler. Gppe: a method to generate ad-hoc feature extractors for prediction in financial domains. *Applied Intelligence*, 29(2):174–185, 2008.

[23] C. Estébanez, J. M. Valls, R. Aler, and I. M. Galván. A first attempt at constructing genetic programming expressions for eeg classification. In *International Conference on Artificial Neural Networks*, pages 665–670. Springer, 2005.

[24] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

[25] P. Felgaer. Optimización de redes bayesianas basado en técnicas de aprendizaje por inducción. *Reportes Técnicos en Ingeniería del Software*, 6(2):64–69, 2004.

[26] G. Folino, C. Pizzuti, and G. Spezzano. Parallel genetic programming for decision tree induction. In *Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001*, pages 129–135, Dallas, TX, USA, 2001. IEEE Comput. Soc.

[27] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701, 1937.

[28] M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[29] S. Garcia and F. Herrera. An extension on"statistical comparisons of classifiers over multiple data sets"for all pairwise comparisons. *Journal of machine learning research*, 9(Dec):2677–2694, 2008.

[30] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on systems, man, and cybernetics*, 16(1):122–128, 1986.

[31] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[32] J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[33] B. Hssina, A. Merbouha, H. Ezzikouri, and M. Erritali. A comparative study of decision tree id3 and c4. 5. *International Journal of Advanced Computer Science and Applications*, 4(2):13–19, 2014.

[34] F. Hutter, H. H. Hoos, and T. Stützle. Automatic algorithm configuration based on local search. In *Aaai*, volume 7, pages 1152–1157, 2007.

[35] A. Ittner and M. Schlosser. Non-linear decision trees-ndt. In *ICML*, pages 252–257. Citeseer, 1996.

[36] K. D. Joshi and D. C. Vakaskar. Evolutionary Algorithms: Symbiosis of its Paradigms. page 6, 2011.

[37] K. Krawiec. [No title found]. *Genetic Programming and Evolvable Machines*, 3(4):329–343, 2002.

[38] A. Krenker, J. Bešter, and A. Kos. Introduction to the artificial neural networks. *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pages 1–18, 2011.

[39] M. Lichman et al. Uci machine learning repository, 2013.

[40] T. Lim. Haberman's survival data set. *UCI Machine Learning Repository, University of California, School of Information and Computer Science, Irvine, CA*, 1999.

[41] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[42] O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1-5):193–225, 1997.

[43] J.-A. Mejía-de Dios. A metaheuristic based on the center of mass. Master's thesis, Centro de Investigaciǿn en Inteligencia Artificial, Universidad Veracruzana, 2017.

[44] J.-A. Mejía-de Dios and E. Mezura-Montes. A new evolutionary optimization method based on center of mass. In *Decision Science in Action*, pages 65–74. Springer, 2019.

[45] T. M. Mitchell. *Machine Learning*. McGraw-Hill series in computer science. McGraw-Hill, New York, 1997.

[46] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.

[47] S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel. Oc1: A randomized algorithm for building oblique decision trees. In *Proceedings of AAAI*, volume 93, pages 322–327. Citeseer, 1993.

[48] R. Myers and E. R. Hancock. Empirical modelling of genetic algorithms. *Evolutionary computation*, 9(4):461–493, 2001.

[49] V. Nannen and A. E. Eiben. Efficient relevance estimation and value calibration of evolutionary algorithm parameters. In *2007 IEEE congress on evolutionary computation*, pages 103–110. IEEE, 2007.

[50] P. Nemenyi. Distribution-free multiple comparisons (doctoral dissertation, princeton university, 1963). *Dissertation Abstracts International*, 25(2):1233, 1963.

[51] H. S. Noghabi, H. R. Mashhadi, and K. Shojaei. Differential evolution with generalized mutation operator for parameters optimization in gene selection for cancer classification. *arXiv preprint arXiv:1510.02516*, 2015.

[52] J. D. Olden and D. A. Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1-2):135–150, 2002.

[53] A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. In *ICML*, volume 1, pages 393–400, 2001.

[54] R. Poli, W. B. Langdon, N. F. McPhee, and J. R. Koza. Genetic programming: An introductory tutorial and a survey of techniques and applications. *University of Essex, UK, Tech. Rep. CES-475*, 2007.

[55] A. Pradhan. Support vector machine-a survey. *International Journal of Emerging Technology and Advanced Engineering*, 2(8):82–85, 2012.

[56] K. Price, R. M. Storn, and J. A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[57] J. L. Puga. Cómo construir y validar redes bayesianas con netica. *REMA*, 17(1):1–17, 2012.

[58] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[59] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi. Gsa: a gravitational search algorithm. *Information sciences*, 179(13):2232–2248, 2009.

[60] M. A. Razi and K. Athappilly. A comparative predictive analysis of neural networks (nns), nonlinear regression and classification and regression tree (cart) models. *Expert Systems with Applications*, 29(1):65–74, 2005.

[61] R. Rivera-Lǿpez. *Differential-Evolution-based methods for inducing Decision Trees*. PhD thesis, 2017.

[62] R. Rivera-Lopez and J. Canul-Reich. Construction of near-optimal axis-parallel decision trees using a differential-evolution-based approach. *IEEE Access*, 6:5548–5563, 2018.

[63] L. Rokach and O. Z. Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

[64] G. Rozenberg, T. Bäck, and J. N. Kok. *Handbook of natural computing*. Springer, 2012.

[65] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda. The cart decision tree for mining data streams. *Information Sciences*, 266:1–15, 2014.

[66] S. L. Salzberg. C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993, 1994.

[67] G. Sathyadevi. Application of cart algorithm in hepatitis disease diagnosis. In *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, pages 1283–1287, 2011.

[68] J. D. Schaffer, R. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the 3rd international conference on genetic algorithms*, pages 51–60, 1989.

[69] A. Shali, M. R. Kangavari, and B. Bina. Using genetic programming for the induction of oblique decision trees. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 38–43, Cincinnati, OH, USA, Dec. 2007. IEEE.

[70] S. K. Smit and A. E. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *2009 IEEE congress on evolutionary computation*, pages 399–406. IEEE, 2009.

[71] S. F. Smith. Rna search acceleration with genetic algorithm generated decision trees. In *2008 Seventh International Conference on Machine Learning and Applications*, pages 565–570. IEEE, 2008.

[72] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[73] G. Taguchi, Y. Yokoyama, et al. *Taguchi methods: design of experiments*, volume 4. Amer Supplier Inst, 1993.

[74] M. F. Tasgetiren, M. Sevkli, Y.-C. Liang, and G. Gençyilmaz. Particle swarm optimization algorithm for single machine total weighted tardiness problem. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 2, pages 1412–1419. IEEE, 2004.

[75] F.-Y. Tzeng and K.-L. Ma. *Opening the black box-data driven visualization of neural networks*. IEEE, 2005.

[76] W. Vent. Rechenberg, Ingo, Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. 170 S. mit 36 Abb. Frommann-Holzboog-Verlag. Stuttgart 1973. Broschiert. *Feddes Repertorium*, 86(5):337–337, Apr. 2008.

[77] Y. Wang. A sociopsychological perspective on collective intelligence in metaheuristic computing. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1(1):110–128, 2010.

[78] I. H. Witten and E. Frank. Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

[79] I. H. Witten, E. Frank, L. E. Trigg, M. A. Hall, G. Holmes, and S. J. Cunningham. Weka: Practical machine learning tools and techniques with java implementations. 1999.

[80] O. T. Yıldız and E. Alpaydın. Univariate and multivariate decision trees, 2000.