ISSN 1870-4069

Inducción de árboles de decisión oblicuos utilizando dos variantes de evolución diferencial adaptiva

Miguel Angel Morales-Hernández[1], Rafael Rivera-López[2], Efrén Mezura-Montes[3]

[1] Laboratorio Nacional de Informática Avanzada, México

[2] Tecnológico Nacional de México, Instituto Tecnológico de Veracruz, México

[3] Instituto de Investigaciones en Inteligencia Artificial, Universidad Veracruzana, México

mangelmhdez@gmail.com, rafael.rl@veracruz.tecnm.mx, emezura@uv.mx

Resumen. Este artículo presenta un análisis comparativo del comportamiento de tres variantes del algoritmo de Evolución Diferencial para inducir árboles de decisión oblicuos. Tomando como base los resultados de una versión clásica del algoritmo, se implementaron dos de sus variantes adaptivas: JADE y SHADE, con el ánimo de observar su comportamiento de búsqueda en un espacio no continuo como es el de los árboles de decisión. Los resultados obtenidos sobre un conjunto de datos de prueba, indica que los algoritmos tienen un comportamiento similar, pero SHADE se comporta mejor construyendo árboles oblicuos para datasets con muchas etiquetas de clase.

Palabras clave: IA, aprendizaje automático, evolución diferencial, árboles de decisión, árboles de decisión oblicuos.

Oblique Decision Trees Induction Using Two Adaptive Differential Evolution Algorithms

Abstract. This article presents a comparative analysis of the behavior of three variants of the Differential Evolution algorithm for inducing oblique decision trees. Based on the results of a classic version of Differential Evolution, two self-adaptive algorithms were implemented: JADE and SHADE, with the aim of observing the behavior of these algorithms in non-continuos search spaces such as the space of decision trees. The results obtained on a test dataset indicate that the

algorithms have a similar behavior, but SHADE performs better in constructing oblique trees for datasets with many class labels.

Keywords: AI, machine learning, diferential evolution, decision trees, oblique decision trees.

1. Introducción

La Inteligencia Artificial (IA) se enfoca en el desarrollo de sistemas que pueden realizar tareas que requieren inteligencia humana, como el reconocimiento de voz, el procesamiento del lenguaje natural, la toma de decisiones, así como la resolución de problemas complejos [7]. La IA trata de emular las capacidades humanas, siendo el aprendizaje un tema de creciente interés.

El Aprendizaje Automático es la rama de la IA centrada en el desarrollo de algoritmos y modelos matemáticos que pueden aprender de los datos, sin ser programados explícitamente. Esta área cuenta con diferentes modelos de aprendizaje, destacándose aquellos de aprendizaje supervisado, que parten de un conjunto establecido de datos etiquetados (datasets) para generar un modelo que permita encontrar patrones y realizar tareas de clasificación o regresión [13].

En este contexto se puede decir que el desarrollo de análisis de datos es un tema importante porque con la información recolectada se pueden tomar mejores decisiones, así como obtener resultados más precisos, particularmente con la implementación y uso de diferentes técnicas de la minería de datos [17], con el objetivo de descubrir patrones en ellos.

1.1. Árboles de decisión

Los árboles de decisión (ADs) son modelos generados a partir de datos que cuentan con una estructura jerárquica similar a la de un árbol biológico, compuesto por ramas y hojas, entre otras características [6].

Ellos particionan un dataset "*de arriba hacia abajo*"distribuyendo los datos en los nodos internos del árbol, iniciando con el nodo raíz y ramificándose hacia los nodos hoja, dividiendo así el espacio de datos usando hiperplanos. En los ADs más populares, conocidos como paralelos a los ejes, cada nodo interno incluye la evaluación de un solo atributo del dataset.

Un aspecto importante al generar ADs es el inducir modelos compactos, ya que permiten tener una mejor interpretación y precisión con los resultados.

Los algoritmos tradicionales para inducir ADs como C4.5 [11] y CART [2] pueden producir árboles con muchas particiones, lo que los hace difíciles de interpretar, por lo que otros tipos de particiones se han estudiado, como aquellos donde se utiliza una combinación lineal de varios atributos en la condición evaluada por un nodo interno.

Estas combinaciones de atributos producen particiones con hiperplanos oblicuos, definidos como sigue:

$$\sum_{i=1}^{d} w_i x_i \leq \theta, \tag{1}$$

donde w_i es un coeficiente numérico aplicado al i-ésimo atributo x_i de un dataset con d atributos, y θ es el término independiente del hiperplano. Los árboles creados usando este enfoque, conocidos como árboles oblicuos, generalmente son más compactos y mas precisos que los tradicionales.

La inducción de este tipo de ADs suele hacerse con buscadores voraces, como el algoritmo CART con combinaciones lineales, y los métodos Linear Machine Decision Trees (LMDT) y Simulated Annealing of Decision Trees (SADT) [10].

1.2. Evolución diferencial

Evolución Diferencial (ED) es uno de los algoritmos más recientes dentro del campo de la computación evolutiva, que ha demostrado ser altamente competitivo para resolver problemas de búsqueda complejos [4], distinguiéndose por la simplicidad en su implementación y su habilidad para encontrar soluciones cercanas al óptimo.

Los elementos de ED provienen del esquema clásico de un algoritmo poblacional: inicialización, mutación, recombinación y selección. Todo empieza con un conjunto inicial de soluciones candidatas, también llamadas *individuos* o vectores, que se generan de forma aleatoria con una distribución uniforme.

Posteriormente se aplica una mutación diferencial y una cruza a cada solución (padre) para generar un nuevo individuo (descendiente), el cual permanecerá para la siguiente generación (iteración) si es mejor que su padre.

Todo lo anterior se repite hasta alcanzar una condición de paro, asociada usualmente a un número máximo de generaciones. La selección de los individuos se basa en la comparación de su aptitud, representada por la función objetivo del problema.

La variante más popular de ED es conocida como DE/rand/1 [15] , cuyo operador de mutación es el siguiente:

$$\mathbf{v}_{i,g} = \mathbf{x}_{r_0,g} + F \cdot (\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}), \tag{2}$$

donde g es la generación actual, $\mathbf{x}_{r_0,g}$, $\mathbf{x}_{r_1,g}$ y $\mathbf{x}_{r_2,g}$ son tres soluciones diferentes entre sí y diferentes al padre, escogidas al azar de la población actual, y $\mathbf{v}_i$ es el vector o solución mutante. $\mathbf{v}_i$ se recombina con la solución padre mediante una cruza discreta que puede ser binomial (bin) o exponencial (exp), para generar la solución descendiente.

Esta nueva solución compite contra su padre y el mejor de ellos, con base en aptitud, sobrevive para la siguiente generación y el otro es eliminado. ED usa tres parámetros: NP que indica el tamaño de la población, F que representa un factor de escala para el vector mutante, y CR, que se utiliza en la recombinación.

Existen otras variantes de ED, como la DE/best/1, que usa el siguiente operador de mutación:

$$\mathbf{v}_{i,g} = \mathbf{x}_{best,g} + F \cdot (\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}), \tag{3}$$

donde la diferencia es que el primer vector usado en la combinación, $\mathbf{x}_{best,g}$, es la mejor solución de la población actual.

1.3. Evolución diferencial para inducir árboles oblicuos

El utilizar algoritmos de búsqueda poblacional como los algoritmos evolutivos es de gran interés, pues la expectativa es que se pueden encontrar modelos (árboles en este caso) de alta calidad gracias a una búsqueda global en contraste con una búsqueda de trayectoria como la de los algoritmos tradicionales [12].

En la actualidad las versiones adaptivas de ED (aquellas que incluyen un autoajuste de los parámetros del algoritmo) han arrojado mejores resultados comparándolos con otros algoritmos evolutivos para la resolución de problemas de optimización numérica [3], ya que pueden adaptar automáticamente las estrategias de aprendizaje y la configuración de parámetros durante la evolución.

En la revisión de la literatura no se ha encontrado un estudio que haya probado la inducción de árboles de decisión oblicuos en un espacio de búsqueda continuo como el de los algoritmos de evolución diferencial adaptiva. Dado a lo anterior también es de suma importancia estudiar el comportamiento de los algoritmos con espacios de búsqueda que no son de naturaleza continua, tal es el caso de los AD oblicuos.

En este contexto se propone trabajar con los árboles de decisión oblicuos, ya que estos suelen mostrar un mejor desempeño y son más compactos que los árboles paralelos a los ejes [14]. El generar árboles oblicuos mediante algoritmos evolutivos adaptivos, particularmente las variantes de ED que adaptan los valores de sus parámetros, representa una oportunidad de desarrollar resultados con mejor precisión e interpretación.

De acuerdo con lo anterior, se plantea inducir AD oblicuos mediante los algoritmos de evolución diferencial adaptivos, y comparar su desempeño contra la versión rand/1/bin. Se propone utilizar los algoritmos adaptivos JADE y SHADE, que han demostrado tener un desempeño destacado al resolver problemas de optimización numérica, comparando los resultados con base en la precisión de clasificación.

El resto del trabajo se organiza de la siguiente manera. La sección 2 presenta de forma detallada la descripción de los algoritmos, las técnicas tradicionales que inducen los AD oblicuos y su comportamiento. La sección 3 muestra los experimentos y resultados obtenidos de cada AD generado, así como las técnicas y algoritmos implementados y una discusión de los resultados obtenidos. Por último en la sección 4 se presentan las conclusiones de los resultados obtenidos y el trabajo futuro.

2. Propuesta

En esta investigación, para que ED pueda inducir árboles de decisión, los AD se representan usando vectores numéricos. En los siguientes párrafos se describe la estrategia de representación y los algoritmos usados en la experimentación.

2.1. Esquema de codificación

En la Fig. 1 se muestran las etapas para la transformación de un vector de parámetros numéricos a un árbol de decisión oblicuo [13]. Este esquema se propone en un trabajo previo donde se utilizó la versión DE/rand/1/bin para inducir árboles oblicuos [14].

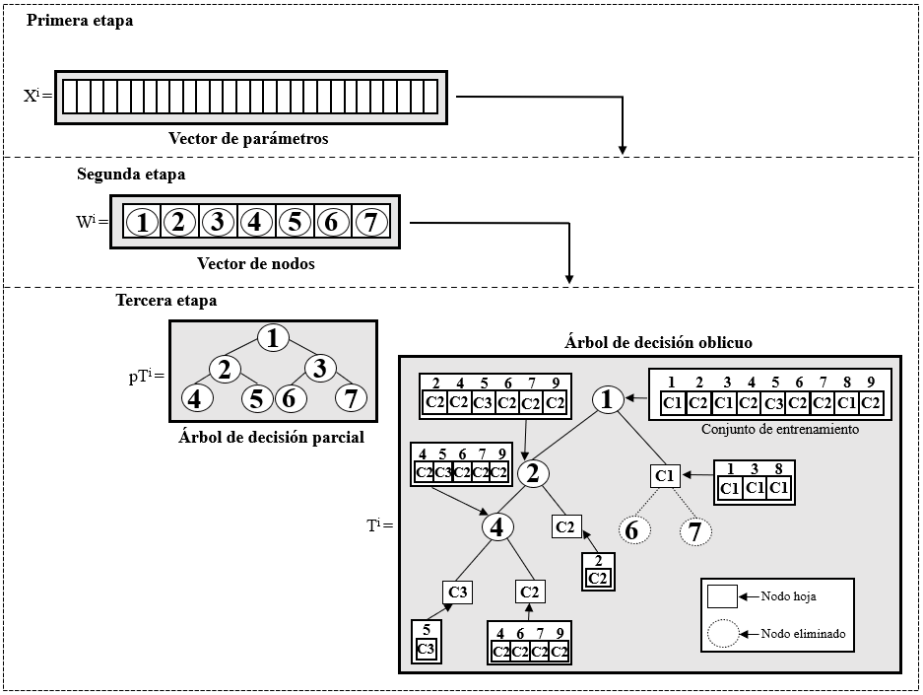

Fig. 1. Proceso de mapeo para la creación de un AD oblicuo.

Tamaño del individuo: La primera etapa consiste en determinar el tamaño del vector de parámetros, usado para crear los nodos internos del AD, mediante las fórmulas que estiman la profundidad de un árbol binario a partir del número de nodos internos y nodos hojas. En este caso se utiliza el número de atributos y el número de etiquetas de clase como un estimado de esos nodos:

$$H_i = \lceil log_2(d+1) \rceil, \tag{4}$$

$$H_l = \lceil log_2(s) \rceil, \tag{5}$$

donde H_i y H_l representan la profundidad del árbol, d es el número de atributos y s es el número de etiquetas de clase.

Posteriormente se aplica la siguiente fórmula para obtener el número estimado de nodos internos (n_e):

$$n_e = 2^{max(H_i, H_l)-1} - 1. \tag{6}$$

Finalmente, el tamaño del vector de parámetros (n) que representa la secuencia de hiperplanos usados en los nodos internos del árbol se obtiene usando la siguiente fórmula:

$$n = n_e(d+1). \tag{7}$$

Vector de nodos internos: En la segunda etapa se crea el vector de nodos internos que se usarán en el árbol. Cada nodo representa un hiperplano usando $d + 1$ valores del vector de parámetros.

Construcción de árbol: Usando el vector de nodos, se construye un árbol binario con los hiperplanos construidos previamente. Este es un árbol parcial que representa solo nodos internos.

El paso final es usar el dataset para particionar sus datos usando los hiperplanos. Si al evaluar un hiperplano los datos son de una misma clase, este nodo se convierte en un nodo hoja, podando todos los nodos descendientes de dicho nodo. Esto nos permite crear árboles de decisión usando hiperplanos en sus nodos internos.

En este trabajo se propone experimentar con dos algoritmos que son versiones adaptivas de ED conocidas como SHADE y JADE. A continuación se detallan ambos.

2.2. Evolución diferencial adaptiva

JADE: El algoritmo de evolución diferencial adaptativa con archivo externo opcional, nombrado JADE por sus autores [18], implementa la estrategia de mutación *DE/current-to-pbest*, junto al auto-ajuste de los parámetros F y CR.

El parámetro F se modifica de acuerdo a una regla de actualización que utiliza la mejor combinación de parámetros de mutación en la población actual, así como de la iteración anterior, mientras que CR se actualiza de forma similar, utilizando la mejor combinación de parámetros de cruza de ambas poblaciones.

En general ambos valores se ajustan utilizando distribuciones de probabilidad independientes en cada iteración, obteniendo el promedio de los vectores mutados mejor adaptados. La estrategia de mutación implementada en este algoritmo es la base principal para el rendimiento y confiabilidad del mismo.

En cada generación, la probabilidad de cruzamiento CR_i de cada individuo $\mathbf{x}_i$ es generada usando una distribución de probabilidad normal con media μ_{CR} y una desviación estándar de 0.1, como sigue:

$$CR_i = randn_i(\mu_{CR}, 0{,}1). \tag{8}$$

El valor μ_{CR} es ajustado en cada generación de acuerdo al siguiente criterio:

$$\mu_{CR} = (1 - c) \cdot \mu_{CR} + c \cdot mean_A(S_{CR}), \tag{9}$$

donde c es una constante positiva y $mean_A$ es la media aritmética de los valores de los CR_i exitosos en la población, almacenados en S_{CR}.

De forma similar, el factor de mutación F_i de cada individuo $\mathbf{x}_i$ es generado usando una distribución de Cauchy con un parámetro μ_F y un parámetro de escala 0.1, como sigue:

$$F_i = randc_i(\mu_F, 0{,}1). \tag{10}$$

El valor μ_F es ajustado en cada generación de acuerdo al siguiente criterio:

$$\mu_F = (1 - c) \cdot \mu_{CR} + c \cdot mean_L(S_F), \tag{11}$$

donde $mean_L$ es la media de Lehmer de los F_i exitosos en la población, almacenados en S_F.

Adicionalmente al autoajuste de parámetros, JADE introduce un nuevo operador de mutación, denominado *DE/current-to-pbest*, basado en combinar la aleatoriedad de las soluciones candidatas usadas en la mutación, que permite una mejor exploración del espacio de búsqueda, con la posibilidad de utilizar los mejores individuos para guiar la búsqueda en zonas prometedoras. El operador se representa como sigue:

$$\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F_i \cdot (\mathbf{x}^p_{best,g} - \mathbf{x}_{i,g}) + F_i \cdot (\mathbf{x}_{r1,g} - \tilde{\mathbf{x}}_{r2,g}), \tag{12}$$

donde $\mathbf{x}^p_{best,g}$ es seleccionado de entre un subconjunto de los mejores individuos de la población actual, $\mathbf{x}_{r1,g}$ es seleccionado aleatoriamente de la población actual, y $\tilde{\mathbf{x}}_{r2,g}$ es seleccionado de la unión de la población actual y un conjunto de soluciones desechadas previamente, almacenadas en un archivo externo.

La estructura de JADE se describe a continuación:

function $JADE$
  $\mu_{CR} = 0{,}5$; $\mu_F = 0{,}5$; $A = \emptyset$
  Creación de una población inicial aleatoria $\{\mathbf{x}_{i,0} | i = 1, 2, \ldots, NP\}$
  for $g \in \{1, \ldots, G\}$ **do**
    $S_F = \emptyset$; $S_{CR} = \emptyset$;
    for $i \in \{1, \ldots, NP\}$ **do**
      $CR_i = randn_i(\mu_{CR}, 0{,}1), F_i = randc_i(\mu_F, 0{,}1)$
      Selecciona al azar $\mathbf{x}^p_{best,g}$ entre el $100_p\,\%$ de mejores soluciones
      Selecciona al azar $\mathbf{x}_{rl,g} \neq \mathbf{x}_{i,g}$ de la población actual **P**
      Selecciona al azar $\tilde{\mathbf{x}}_{r2,g} \neq \mathbf{x}_{r1,g} \neq \mathbf{x}_{i,g}$ de $\mathbf{P} \cup \mathbf{A}$
      $\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F_i \cdot (\mathbf{x}^p_{best,g} - \mathbf{x}i, g) + F_i \cdot (\mathbf{x}_{r1,g} - \tilde{\mathbf{x}}_{r2,g})$
      $j_{rand} = randint(1, D)$
      for $j \in \{1, \ldots, D\}$ **do**
        if $j = j_{rand} \vee rand(0, 1) < CR_i$ **then**
          $\mathbf{u}_{j,i,g} = \mathbf{v}_{j,i,g}$
        else
          $\mathbf{u}_{j,i,g} = \mathbf{x}_{j,i,g}$
        end if
      end for
      if $f(\mathbf{x}_{i,g}) \leq f(\mathbf{u}_{i,g})$ **then**
        $\mathbf{x}_{i,g+1} = \mathbf{x}_{i,g}$
      else
        $\mathbf{x}_{i,g+1} = \mathbf{u}_{i,g}$; $\mathbf{x}_{i,g} \rightarrow \mathbf{A}$; $CR_i \rightarrow S_{CR}$; $F_i \rightarrow S_F$
      end if
    end for
    Elimina al azar soluciones en **A** tal que $|\mathbf{A}| \leq NP$
    $\mu_{CR} = (1 - c) \cdot \mu_{CR} + c \cdot mean_A(S_{CR})$
    $\mu_F = (1 - c) \cdot \mu_F + c \cdot mean_L(S_F)$
  end for
end function

SHADE: Este algoritmo surge de querer encontrar una mejora a la robustez del algoritmo de JADE, a través de una adaptación de sus parámetros basada en el historial de éxito [16].

SHADE contiene un conjunto de parámetros para guiar la actualizacion de CR y F, a medida que avanza la búsqueda, considerando la historia de cada generación llevada a cabo. Además, los vectores de prueba se generan para ser aplicados en la selección y la memoria histórica será actualizada. Lo anterior se repetirá hasta alcanzar algún criterio de terminación. SHADE se describe a continuación.

function $SHADE$
  $G = 0$;
  Creación de una población inicial aleatoria $\{\mathbf{x}_{i,0} \mid i = 1, 2, \ldots, NP\}$
  $M_{CR} = 0{,}5, M_F = 0{,}5$;
  A = $\emptyset$;
  $k = 1$;
  while No se alcance la condición de paro **do**
    $S_{CR} = \emptyset, S_F = \emptyset$;
    for $i \in \{1, \cdots, N\}$ **do**
      $r_i = randint(1, H)[1, H]$;
      $CR_{i,G} = randn_i(M_{CR,r_i}, 0{,}1)$;
      $F_{i,G} = randc_i(M_{F,r_i}, 0{,}1)$;
      $p_i, G = rand[p_{min}, 0{,}2]$;
      Crear $\mathbf{u}_{i,G}$ usando el operador *current-to-pbest/1/bin*;
    end for
    for $i \in \{1, \ldots, N\}$ **do**
      if $f(\mathbf{u}_{i,G}) \leq f(\mathbf{x}_{i,G})$ **then**
        $\mathbf{x}_{i,G+1} = \mathbf{u}_{i,G}$;
      else
        $\mathbf{x}_{i,G+1} = \mathbf{x}_{i,G}$;
      end if
      if $f(\mathbf{u}_{i,G}) < f(\mathbf{x}_{i,G})$ **then**
        $\mathbf{x}_{i,G} \rightarrow \mathbf{A}$;
        $CR_{i,G} \rightarrow S_{CR}, F_{i,G} \rightarrow S_F$;
      end if
    end for
    Si el tamaño de **A** excede $|\mathbf{A}|$,
    eliminar al azar individuos en **A** tal que $|\mathbf{A}| \leq |\mathbf{P}|$;
    if $S_{CR} \neq \emptyset \wedge S_F \neq \emptyset$ **then**
      Actualiza $M_{CR,k}, M_{F,k}$ usando S_{CR} y S_F;
      $k++$;
      if $k > H$ **then**
        $k = 1$;
      end if
    end if
  end while
end function

Tabla 1. Descripción de los datasets usados en el estudio experimental.

Dataset	Instancias	Atributos	Clases
glass	214	9	7
diabetes	768	8	2
australian	690	14	2
ionosphere	351	34	2
iris	150	4	3
balance-scale	625	4	3
ecoli	336	7	8
heart-startlog	270	13	2
liver-disorders	345	6	2
wine	178	13	3

SHADE depende del tamaño de memoria representado por H. Si este valor es pequeño, los valores tienen un uso frecuente, porque los valores más antiguos se sobrescriben rápidamente, favoreciendo una rápida convergencia de los valores de los parámetros de control. En caso contrario (valor grande), la tasa de control se espera hasta que la convergencia de parámetros disminuya, debido a que éstos seguirán teniendo influencia durante más tiempo.

3. Experimentos y resultados

3.1. Diseño experimental

Para poder evaluar si el uso de algoritmos adaptivos basados en DE ofrecen una ventaja sobre el trabajo existente, es necesario realizar un estudio experimental de su comportamiento. Los algoritmos a comparar en este estudio son DE/rand/1/bin, JADE y SHADE. Estos algoritmos fueron implementados en Java y utilizando la librería JMetal [5].

Primero se seleccionaron una serie de datasets, luego se definen los parámetros utilizados para ajustar las versiones de ED a comparar, y se determina la forma de usar los datos para construir los modelos de aprendizaje.

Datasets seleccionados. Para llevar a cabo el estudio experimental, se seleccionaron diez datasets del repositorio de aprendizaje supervisado (UCI) [9]. Debido a que las particiones que los nodos internos generan al evaluar el dataset representan una combinación lineal de valores de los atributos, todos los datasets utilizados tienen atributos numéricos solamente. La Tabla 1 describe las características de estos datasets.

Parámetros de los algoritmos. Tomando en cuenta lo descrito en la literatura, se han utilizado los siguientes parámetros para cada algoritmo.

- **rand/1/bin**: El factor de cruzamiento CR es 0.9, y el factor de escala es (F) es 0.9.
- **JADE:** La taza de cruzamiento promedio (μ_{CR}) es 0.5, el factor de escala promedio (μ_F) es 0.5, la tasa de mejores soluciones (p) es 0.05, y el valor del parámetro de balance (c) es 0.1.

Tabla 2. Desempeño de los algoritmos DE-ODT, JADE-ODT y SHADE-ODT en varios conjuntos de datos.

Dataset	DE-ODT	JADE-ODT	SHADE-ODT
glass	57.47 (3)	59.48 (2)	**61.02** (1)
diabetes	**74.21** (1)	73.54 (3)	74.14 (2)
australian	**79.71** (1)	75.23 (3)	76.71 (2)
ionosphere	90.79 (2)	89.08 (3)	**90.91** (1)
iris	**96.86** (1)	95.53 (3)	96.53 (2)
balance-scale	**90.41** (1)	90.12 (2)	90.09 (3)
ecoli	75.68 (3)	77.29 (2)	**77.52** (1)
heart-startlog	**83.18** (1)	70.33 (3)	78.25 (2)
liver-disorders	69.18 (3)	**70.00** (1)	69.65 (2)
wine	**89.15** (1)	72.35 (3)	85.73 (2)
Rank promedio	1.7	2.4	1.8

– **SHADE**: El tamaño de la memoria histórica (H) es 100, la proporción del tamaño del archivo externo (A) es 2.0 y la tasa de mejores soluciones (p) es 0.1.

El tamaño de población es de 100 individuos y el número máximo de evaluaciones es de $10,000 * n$, donde n es el tamaño del individuo.

Evaluación de los conjuntos de datos. El proceso evolutivo de cada algoritmo es guiado por la precisión de clasificación de los árboles de decisión como función de aptitud. Para obtener estimaciones fiables del rendimiento predictivo de los algoritmos implementados, se utilizó una validación cruzada (VC) de 10 folds [1]. VC es uno de los métodos de re-muestreo de datos más utilizados para estimar la predicción real [13].

En una validación cruzada de 10 folds, el conjunto de entrenamiento se divide aleatoriamente en diez folds disjuntos aproximadamente iguales. Para cada $k \in \{1, \ldots, 10\}$, se retiene el $k-$ésimo fold (el conjunto de prueba) y los folds restantes se utilizan para inducir un árbol de decisión. Una vez que se ha construido el árbol, el fold retenido se usa para calcular la precisión de la prueba.

Finalmente, cuando todos los folds se han usado en la fase de inducción, se calcula la precisión general de la prueba del modelo. Este esquema se repite diez veces, y al final se reporta la precisión de prueba promedio.

3.2. Resultados

La Tabla 2 muestra la precisión promedio de las diez ejecuciones para cada dataset y cada algoritmo. Los mejores resultados de cada dataset se resaltan en negritas y los números que se encuentran entre paréntesis, se refieren al ranking obtenido por el algoritmo al compararlo con los resultados de los otros algoritmos en un dataset particular.

3.3. Discusión

Como podemos observar en la Tabla 2, DE/rand/1/bin y SHADE obtienen mejores resultados por generar los promedios más bajos, *1.7* y *1.8* respectivamente, a

comparación de JADE que obtuvo *2.4*. En principio eso se esperaría al comparar JADE con SHADE, ya que es conocido que el comportamiento de SHADE en otros problemas es mejor al de JADE.

Sin embargo, se esperaría que SHADE superara en desempeño a las versiones tradicionales de DE, pero en estos resultados iniciales se encuentra lo contrario. Algo interesante que se observa es que SHADE obtiene mejores resultados para datasets con mayor número de clases (como en el caso de los datasets glass y ecoli), pero para dataset con dos clases, DE/rand/1/bin tiene mejor desempeño.

Se puede justificar este comportamiento debido a la forma de comparar los modelos generados, ya que es muy probable que SHADE obtenga mejor desempeño con los datos de entrenamiento, que se usan dentro del proceso evolutivo, y que al evaluar con los datos de prueba, su desempeño se degrade, lo que podría indicar que el modelo entrenado está sobreajustado.

En el caso de SHADE, se utiliza una memoria igual al tamaño de la población, y se observa que los parámetros varían lentamente, lo que sugiere evaluar el uso de una memoria de menor tamaño para acelerar la convergencia de las soluciones.

4. Conclusiones y trabajo futuro

Los resultados reportados en este trabajo son los primeros que se obtuvieron de una investigación en curso, y nos permiten reconocer que es importante seguir analizando el comportamiento de los algoritmos autoadaptables como JADE, SHADE y sus derivados, para obtener modelos de aprendizaje supervisado más precisos.

La métrica comparada inicialmente es la precisión de clasificación, para posteriormente analizar otras métricas como el tamaño del modelo y aquellas basadas en la matriz de confusión de los datos. Existen varios desafíos interesantes que se observan de estos resultados, como el poder generar árboles con menor número de nodos, y que el proceso evolutivo evite el sobre ajuste de los modelos.

También es interesante estudiar el comportamiento de otras variantes de los algoritmos autoadaptables conocidos en la literatura, tales como las versiones actualizadas de SHADE y aplicarlos a conjuntos de datos más robustos con otras características como su número de instancias y distribución de clases.

Agradecimientos. El primer autor agradece el apoyo de CONACyT mediante una beca para la realización de estudios de maestría en el Laboratorio Nacional de Informática Avanzada y la realización de la residencia profesional en el Instituto de Investigaciones en Inteligencia Artificial de la Universidad Veracruzana.

Referencias

1. Berrar, D.: Cross validation. Data Science Laboratory, Institute of Technology (2018) doi: 10.1016/B978-0-12-809633-8.20349-X
2. Breiman, L., Friedman, J. H., Olshen, R., Stone, C. J.: Classification and regression trees. Routledge (1984) doi: 10.1201/9781315139470

3. Brest, J., Bošković, B., Greiner, S., Žumer, V., Maučec, M.: Performance comparison of self-adaptive and adaptive differential evolution algorithms. Soft Computing, vol. 11, pp. 617–629 (2007) doi: 10.1007/s00500-006-0124-0
4. Coello, C.: Computación evolutiva. Academia Mexicana de Computación, Amexcomp (2019)
5. Durillo, J. J., Nebro, A. J.: jMetal: A Java framework for multi-objective optimization. Advances in Engineering Software, vol. 42, no. 10, pp. 760-–771 (2011) doi: 10.1016/j.advengsoft.2011.05.014
6. Bhargava. N., Sharma, G., Bhargava, R., Mathuria, M.: Decision tree analysis on J48 algorithm for data mining. In: Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, no. 6, pp. 1114–1119 (2013)
7. Grewal, D.: A critical conceptual analysis of definitions of artificial intelligence as applicable to computer engineering. IOSR Journal of Computer Engineering, vol. 16, no. 2, pp. 9–13 (2014) doi: 10.9790/0661-16210913
8. Hernández, S.: Mejoras al algoritmo de evolución diferencial para resolver problemas de optimización con restricciones. Universidad Nacional de la Patagonia Austral (2015)
9. Lichman, M.: UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences (2013)
10. Murthy, S., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Journal of Artificial Intelligence Research, vol. 2, pp. 1–32 (1994) doi: 10.1613/jair.63
11. Quinlan, J. R.: C4.5: Programs for machine learning. Machine Learning, vol. 16, pp. 235–240 (1994) doi: 10.1007/BF00993309
12. Rivera-Lopez, R., Canul-Reich, J., Mezura-Montes, E., Cruz-Chávez, M. A: Induction of decision trees as classification models through metaheuristics. Swarm and Evolutionary Computation, vol. 69 (2022) doi: 10.1016/j.swevo.2021.101006
13. Rivera-Lopez, R., Canul-Reich, J.: Construction of near-optimal axis-parallel decision trees using a differential-evolution-based approach. IEEE Access, vol. 6, pp. 5548-5563 (2018) doi: 10.1109/ACCESS.2017.2788700
14. Rivera-Lopez, R., Canul-Reich, J.: A global search approach for inducing oblique decision trees using differential evolution. In: Canadian Conference on Artificial Intelligence, vol. 10233, pp. 27–38 (2017) doi: 10.1007/978-3-319-57351-9_3
15. Storn, R., Price, K.: Differential evolution-A simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, vol. 11, no. 4, pp. 341–359 (1997) doi: 10.1023/A:1008202821328
16. Tanabe, R., Fukunaga, A.: Success-history based parameter adaptation for differential evolution. In: 2013 IEEE Congress on Evolutionary Computation, pp. 71–78 (2013) doi: 10.1109/CEC.2013.6557555
17. Witten, I., Frank, E., Hall, M.: Data mining practical machine learning tools and techniques. The Morgan Kaufmann Series in Data Management Systems (2005) doi: 10.1016/C2009-0-19715-5
18. Zhang. J., Sanderson, A.: JADE: Adaptive differential evolution with optional external archive. EEE Transactions on Evolutionary Computation, vol. 13, no. 5, pp. 945–958 (2009) doi: 10.1109/TEVC.2009.2014613