Algoritmos Evolutivos
Informe Final

Eugenia Miranda
Facultad de Ingeniería
Universidad de la República
Email: maria.eugenia.miranda@fing.edu.uy

Andrés Vidal
Facultad de Ingeniería
Universidad de la República
Email: andres.vidal@fing.edu.uy

***Resumen*—La neuroevolución consiste en el empleo de técnicas de computación evolutiva para construir redes neuronales especializadas en resolver problemas determinados. Si bien el entrenamiento de redes neuronales es un problema resuelto eficientemente por el algoritmo de *backpropagation*, el atractivo de los Algoritmos Evolutivos en este contexto radica en la posibilidad de manipular la topología de las redes participantes en el proceso. En este proyecto, se busca diseñar un algoritmo evolutivo capaz de optimizar la estructura y los pesos de redes neuronales utilizadas en problemas de clasificación.**

1. Introducción

Este trabajo es un informe sobre el desarrollo del proyecto final de la asignatura de Algoritmos Evolutivos de la Facultad de Ingeniería de la Universidad de la República. El proyecto aborda el problema de la neuroevolución, que consiste en la generar redes neuronales a través de algoritmos evolutivos. En particular, se busca diseñar un algoritmo que construya redes neuronales para clasificación multiclase que tengan buen desempeño según la medida F_1.

En este contexto, se definen formalmente tanto el problema de clasificación multiclase y el problema de optimización asociado, como el modelo de redes neuronales considerado para generar la representación de las soluciones. Además, se detalla la implementación del algoritmo evolutivo con énfasis en los operadores evolutivos diseñados y en los parámetros que controlan su comportamiento.

Luego, se realiza un proceso de configuración paramétrica en el cual participan tres parámetros: el tamaño de la población, la probabilidad de mutación y la probabilidad de cruzamiento. De esta etapa resulta una configuración destacada, que se utiliza para desarrollar la evaluación del algoritmo y su comparación con redes estáticas entrenadas con *backpropagation*. Se utilizan tres *datasets* para configuración y tres *datasets* para evaluación.

En general, el algoritmo diseñado no presentó resultados positivos para esta tarea. Especialmente, su desempeño varía demasiado entre instancias del problema y los altos tiempos de ejecución imposibilitaron tanto la evaluación estadística de la eficiencia computacional como la realización de pruebas iterativas que guiasen el perfeccionamiento del diseño. De todas formas, fue posible realizar un análisis estadístico consistente, diagnosticar los problemas encontrados y generar hipótesis sobre cómo mejorar el algoritmo.

2. Problema

El problema considerado es el de la neuroevolución, que consiste en utilizar técnicas de aprendizaje evolutivo para construir y entrenar redes neuronales. Éstas son estructuras utilizadas para calcular funciones complicadas, que pueden representar la solución a varios problemas de aprendizaje. Estas redes están compuestas por neuronas interconectadas, que se disponen en varias capas.

La cantidad de capas, bien como la cantidad de neuronas por capa, definen la topología de la red. Existen dos tipos de capas: las explícitas y las implícitas. Las explícitas corresponden a la capa de entrada (la primera) y a la capa de salida (la última). Las implícitas son las capas intermedias, que agregan capacidad de representación a la red. Las capas se disponen en orden creciente según la distancia a la capa de entrada y son responsables por procesar un vector de entrada provisto por la capa anterior y propagar la información a la capa siguiente.

Las neuronas son unidades de procesamiento que se activan al recibir una señal (un valor de entrada) y la propagan a las neuronas de la siguiente capa. Este fenómeno, llamado activación, tiene asociada una función de activación que procesa el valor de entrada antes de propagarlo. En general, cada neurona de la capa actual recibe como valor de entrada una combinación lineal de los elementos del vector de entrada $\vec{e} = \left[\begin{array}{cccc:c} e_1 & e_2 & \dots & e_n & 1 \end{array}\right]$. Esta combinación lineal es el producto escalar de $\vec{e}$ con un vector de pesos $\vec{W}_i$ exclusivo de cada neurona i. Definimos:

$$\vec{W}_i = \begin{bmatrix} w_{1i} & w_{2i} & \dots & w_{ni} \end{bmatrix}$$

En el modelo considerado, todas las neuronas de una capa está conectadas con todas las neuronas de la capa siguiente. Asimismo, no existen conexiones entre las neuronas de una misma capa. La desconexión de neuronas se representa con un peso nulo.

Esto permite definir un vector de transición $\vec{t}$, cuya i-ésima posición corresponde al valor de entrada de la i-ésima

neurona. Entonces, se define la matriz de transición T que permite calcular $\vec{t} = T\vec{e}$ como:

$$T = \begin{bmatrix} \vec{W}_1 \\ \vec{W}_2 \\ \vdots \\ \vec{W}_m \end{bmatrix} = \left[\begin{array}{cccc|c} w_{11} & w_{21} & \dots & w_{n1} & b_1 \\ w_{12} & w_{22} & \dots & w_{n2} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{1m} & w_{2m} & \dots & w_{nm} & b_m \end{array}\right]$$

Dónde n es la cantidad de neuronas de la capa anterior y m es la cantidad de neuronas de la capa actual. El vector $\vec{B} = \begin{bmatrix} b_1 & b_2 & \dots & b_m \end{bmatrix}$ contiene el sesgo de cada neurona, que se puede ver como un término de la combinación independiente de las neuronas de la capa anterior. Además, si a_i es la función de activación de la i-ésima neurona de la capa actual, se define la función de activación $h : \mathbb{R}^m \to \mathbb{R}^m$ de esta capa como:

$$h(\vec{x}) = \begin{bmatrix} a_1(x_1) & a_2(x_2) & \dots & a_m(x_m) \end{bmatrix}$$

En una red neuronal de k capas, al considerar h_i como la función de activación de la capa i y T_i como la matriz de transición entre la i-ésima capa y la siguiente, dado un vector de entrada $\vec{e}$ es posible calcular el vector de salida $\vec{s}$ de la red como:

$$\vec{s} = h_k(T_k h_{k-1}(\dots h_2(T_2 h_1(T_1 \vec{e}))))$$

Esta expresión permite evaluar la función representada por la red neuronal en tiempo de ejecución constante, considerando que la topología de la red es fija y que las matrices de transición están dadas.

Sin embargo, la complejidad computacional presentada por las redes neuronales radica en su entrenamiento, esto es, en el ajuste de las matrices de transición de acuerdo a cierto conjunto de datos de ejemplo. De todas formas, el algoritmo de propagación hacia atrás (*backpropagation*) es una solución que utiliza descenso por gradiente para ajustar estos valores de forma eficiente.

El principal problema en el desarrollo de redes neuronales consiste en la definición de la topología, dado que es difícil establecer criterios que equilibren la complejidad de la red y su capacidad de aprendizaje. En general, la topología se define en base a la experiencia de los investigadores que la desarrollan. Por este motivo, si bien el uso de técnicas evolutivas para entrenar a la red perdió espacio frente al algoritmo de *backpropagation*, éstas son útiles para optimizar la topología de la red. Este es el foco de la neuroevolución.

En este proyecto, se busca optimizar la topología de una red neuronal, abstraída como el número de capas y el número de neuronas por capa, así como las funciones de activación utilizadas y los valores de las matrices de transición. Se optó por usar técnicas evolutivas para lo último, en vez de descenso por gradiente, con el objetivo de homogeneizar las técnicas de aprendizaje. De esta forma, en la etapa de evaluación se compara una red generada totalmente por un algoritmo evolutivo con otra entrenada con *backpropagation*.

Dado que el objetivo del trabajo se enfoca en las técnicas de aprendizaje evolutivo, se optó por simplificar la tarea a la que se aplicará la red neuronal. Por este motivo, se considerará un problema de clasificación. En este tipo de problema se plantea un universo Ω compuesto por instancias clasificables en varias categorías.

Cada instancia $\omega \in \Omega$ es descripta por un conjunto de variables explicativas $a_1, a_2, \dots, a_n$. En este contexto, si $a_i \in A_i$, entonces $\Omega = A_1 \times A_2 \times \dots \times A_n$ y la clasificación de las instancias es generada por una función $c : \Omega \to C$, dónde C es el conjunto de categorías.

El problema de clasificación consiste en encontrar, con base en un conjunto de ejemplos $\Omega^* \subset \Omega$, una función c^* que aproxime c. En este caso, c^* es la función expresada por la red neuronal. Una instancia del problema es caracterizada por Ω, Ω^*, $A_1, \dots, A_n, C$ y los valores de $c(\omega)$ conocidos para los elementos de Ω^*. Al cambiar cualquiera de estos elementos, se genera un nueva instancia del problema.

En este caso, se utiliza la medida F_1 para determinar el desempeño de una red en la tarea de clasificación. Esta medida es una función de otras dos medidas de desempeño: la precisión y la exhaustividad. En un problema de clasificación binaria (con 2 clases), la precisión estima la probabilidad de que una instancia clasificada como positiva sea efectivamente positiva. Entretanto, la exhaustividad estima la probabilidad de que una instancia efectivamente positiva sea clasificada como positiva.

En general, la precisión se utiliza cuando los falsos positivos resultan en altos costos para las partes interesadas en la solución, mientras que la exhaustividad se utiliza cuando los falsos negativos son más costosos [2]. Como la decisión entre una u otra medida requiere conocimiento específico sobre el propósito de resolver el problema de clasificación propuesto, se optó por utilizar la medida F_1, que busca un equilibrio entre las anteriores [2]. Sean p y r la precisión y la exhaustividad calculadas, definimos:

$$F_1 = 2\frac{pr}{p+r}$$

Considerando que se propone resolver el problema de clasificación para cualquier cantidad de clases y que F_1 es una medida para clasificación binaria, se tomará como función de evaluación $f(red)$ el promedio de la medida F_1^i para cada clase i ponderado por la cantidad c_i de ejemplos pertenecientes a la clase i. Si consideramos un problema de clasificación con C clases:

$$f(red) = \frac{\sum_{i=1}^{C} c_i F_1^i}{\sum_{i=1}^{C} c_i}$$

Considerando el modelo de redes neuronales definido, así como los detalles del problema de optimización, se puede resumir el problema abordado en este proyecto como "*encontrar redes neuronales para clasificación multiclase, que presenten buen desempeño en términos de la medida* F_1".

3. Solución

Una vez definidos el modelo de redes neuronales utilizados y el problema de optimización abordado, es posible proponer una solución para el mismo utilizando algoritmos evolutivos. Para esto, es necesario determinar la representación (genotipo) de los individuos en el proceso evolutivo, la función de evaluación, la función de *fitness* y los operadores evolutivos que serán utilizados (selección, cruzamiento y mutación), además del esquema de reemplazo. También se presenta una tabla con los parámetros del algoritmo evolutivo.

3.1. Representación

El genotipo de una red neuronal en este contexto se define como un conjunto de capas. La representación de estas capas son pares (h, T), dónde h es una función de activación y T es una matriz de transición. En particular, si se quiere representar una red con k capas, el genotipo contiene k pares numerados $\{(h_1, T_1), (h_2, T_2), \ldots, (h_k, T_k)\}$. A su vez, la representación completa de un capa i con m neuronas, siendo que la capa anterior tiene n neuronas, es:

$$(h_i, T_i) = \left(\begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_m \end{bmatrix}, \left[\begin{array}{cccc:c} w_{11} & w_{21} & \ldots & w_{n1} & b_1 \\ w_{12} & w_{22} & \ldots & w_{n2} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{1m} & w_{2m} & \ldots & w_{nm} & b_m \end{array}\right] \right)$$

Del modelo explicado anteriormente, específicamente de la expresión presentada para calcular un resultado utilizando la red neuronal, es posible establecer las condiciones de factibilidad de un genotipo:

1. Las neuronas de la primera capa deben tener conexiones para recibir todos los valores del vector de entrada. Sea v la cantidad de variables explicativas del problema de clasificación:
   $T_1 \in \boldsymbol{M}_{n\times(v+1)}$

2. La última capa debe tener tantas neuronas cuantas categorías haya en el problema de clasificación, para generar un vector de salida con el tamaño requerido. Sea c la cantidad de categorías del problema de clasificación:
   $T_k \in \boldsymbol{M}_{c\times n}$

3. La función de activación de la última capa debe ser $softmax$, para garantizar que el vector de salida represente una distribución de probabilidad. Sea c la cantidad de categorías del problema:
   $h_k(\vec{x}) = softmax(\vec{x})$ dónde $softmax : \mathbb{R}^c \Rightarrow [0, 1]^c$

4. Las matrices de transición de dos capas consecutivas tienen que ser multiplicables en orden decreciente, esto es, se debe poder calcular $T_{i+1} * T_i$.
   $T_i \in \boldsymbol{M}_{m\times n} \Leftrightarrow T_{i+1} \in \boldsymbol{M}_{d\times(m+1)}$
   $\forall i \in \{1, \ldots, k-1\}$

5. En una misma capa, la cantidad de filas de la matriz de transición y el tamaño del vector de activación deben coincidir, puesto que ambos representan la cantidad de neuronas.
   $T_i \in \boldsymbol{M}_{m\times n} \Leftrightarrow h_i : \mathbb{R}^m \to \mathbb{R}^m$
   $\forall i \in \{1, \ldots\ k\}$

Recordando que a los vectores de salida (excepto al de la última capa) se les concatena el valor 1 al final para representar el término independiente de las neuronas en la combinación lineal de la siguiente capa.

En la práctica, se utilizan las estructuras de datos de Python para implementar la representación propuesta. Las redes son listas de tuplas, ambas estructuras nativas de Python. De esta forma, cada tupla representa una capa, conteniendo en la primera posición a la función de activación y en la segunda a la matriz de transición.

Tanto la función de activación de una capa como su matriz de transición se implementan con arreglos Numpy. La primera es un arreglo de funciones y la segunda es un arreglo de números en punto flotante. Las posibles funciones para componer el arreglo se definen en un archivo de configuración y son: arcotangente (`atan` de `math`); tangente hiperbólica (`tanh` de `math`); dos variantes del rectificador lineal (relu y lrelu) y la identidad. Para las últimas tres funciones se utilizan implementaciones propias. Además, se utiliza `softmax` de `scipy.special` para las capas de salida.

3.2. Función de *fitness*

Durante el proceso evolutivo, las redes neuronales obtenidas serán evaluadas mediante su desempeño para clasificar las instancias del problemas, el cual se quiere maximizar. Para determinar el desempeño de una red en la tarea de clasificación se utiliza la medida F_1.

Como se considera un problema de maximización, la función de *fitness* corresponde a la función $f(red)$ expresada anteriormente. Entonces, el proceso de evaluación de una red consiste en, dado un conjunto de instancias de ejemplo:

1. calcular el resultado de evaluar la red en cada instancia;
2. para cada instancia, elegir la categoría más probable;
3. comparar los resultados con las clasificaciones reales;
4. calcular la medida F_1^i para cada categoría i; y
5. calcular el *fitness* utilizando la expresión de $f(red)$

Recordando, en el paso 2, que la función $softmax$ puede ser utilizada para representar la distribución de probabilidad sobre las diferentes salidas.

3.3. Inicialización

El proceso de inicialización de una población consiste en la generación de una cantidad prefijada de individuos generados aleatoriamente. Con el objetivo de limitar el tamaño

de los individuos generados en este proceso, se definen una cantidad máxima de capas por red y una cantidad máxima de neuronas por capa. Estas cotas son respetadas únicamente en la inicialización, por lo que el proceso evolutivo puede resultar en redes con más capas o más neuronas en alguna capa.

La creación de un individuo aleatorio se genera secuencialmente, desde la primera hasta la última capa. La cantidad de capas de la red neuronal se define sorteando, con probabilidad uniforme, un número entre 2 y la cantidad máxima de capas permitida. El número sorteado incluye, además de las capas implícitas, a las dos explícitas (entrada y salida).

Las dimensiones de las matrices y de las funciones de activación de las capas explícitas, (h_1, T_1), (h_k, T_k) están fijas para garantizar la factibilidad de las soluciones. Ambas matrices de transición son cuadradas, siendo que las dimensiones de la primera coinciden con la cantidad de variables explicativas de la instancia del problema, mientras que las dimensiones de la última se corresponden con la cantidad de categorías.

Los pares (h_i, T_i) se generan secuencialmente para garantizar la compatibilidad de las dimensiones entre capas. El algoritmo consiste en sortear la cantidad de neuronas de la capa actual (entre 1 y el máximo permitido, con probabilidad uniforme); generar las estructuras necesarias (vector de activación y matriz de transición) según el requerimiento de conexiones propagado de la capa anterior; y transmitir el nuevo requerimiento de conexiones para la capa siguiente. El requerimiento de conexiones de la capa siguiente coincide con la cantidad de neuronas generadas en la capa actual.

Los valores de las matrices de transición son números reales generados con distribución normal estándar. Las componentes de los vectores de activación se eligen con probabilidad uniforme dentro de un conjunto de funciones posibles, excepto para la última capa, que utiliza siempre activación por $softmax$.

3.4. Operadores evolutivos

3.4.1. Cruzamiento. El cruzamiento entre dos individuos está basado en el implementado por el algoritmo *Neuroevolution of Augmenting Topologies* (NEAT) [10] y se realiza de forma secuencial, desde la primera hasta la última capa. Al cruzar dos matrices de transición es necesario tener en cuenta las condiciones de factibilidad, de modo que la cantidad de columnas de la matriz sea consistente con la dimensión del vector de entrada. Sea $k \in \mathbb{N}$ la cantidad de columnas que deben tener los descendientes, se define la intersección entre dos matrices de transición como la región común a las dos, acotada en la segunda dimensión por el valor de k.

Esta noción permite identificar neuronas en común entre las redes neuronales que participan del cruzamiento, las cuales son base para la generación de nuevos individuos. Formalmente, la intersección entre dos matrices $A \in \boldsymbol{M}_{n\times m}$ y $B \in \boldsymbol{M}_{r\times s}$ corresponde a la región compuesta por las entradas (i, j) tales que $i < min\{n, r\}$ y $j < min\{m, r, k\}$.

Las regiones de cada matriz que complementan la intersección se denominan excesos. Las columnas a la derecha de la intersección representan exceso de conexiones y las filas abajo de la intersección representan exceso de neuronas. Es necesario definir estos conceptos para expresar formalmente el cruzamiento entre matrices. Se destaca que, dadas dos matrices y su intersección, a lo sumo una tendrá exceso de conexiones y a lo sumo una tendrá exceso de neuronas.

El cruzamiento entre dos matrices genera otras dos matrices de transición, cada una de las cuales aporta a la generación de un individuo diferente. Las matrices generadas tienen dimensiones menores o iguales a las de la mayor matriz ancestral.

La base para la construcción de las matrices descendientes es la intersección de las matrices ancestrales. Los valores de las entradas pertenecientes a esta región se eligen con equiprobabilidad entre los valores presentes en los padres. Una vez finalizado este proceso, independiente entre ambos descendientes, se procede de manera diferente para cada uno de ellos:

1. Para el primer descendiente, se completa la matriz utilizando los excesos de la matriz ancestral perteneciente al individuo con mayor *fitness*. Si este es vacío, no se agregan nuevas neuronas. Respecto al exceso de conexiones, si la intersección tiene n columnas, se sortean $k - n$ columnas del exceso de la matriz mencionada. El exceso de filas es agregado en su totalidad desde la misma matriz.
2. Para el segundo descendiente, se completa la matriz utilizando los excesos de ambas matrices ancestrales. Si la intersección tiene n columnas, se sortean $k - n$ columnas de la matriz que contenga exceso de conexiones. Respecto al exceso de neuronas, para cada fila de la matriz que tenga filas sobrantes, se sortea con probabilidad $p < 0{,}5$ si la neurona se incluirá en la nueva matriz.

Es necesario observar que debido a la independencia entre los procedimientos de generación de ambos descendientes, es probable que el valor k de columnas requeridas en la matriz de una determinada capa difiera entre los hijos generados por un mismo cruzamiento.

Las regiones de las matrices que no se puedan heredar por falta de información en los padres se inicializan en 0. Las matrices generadas por el primer procedimiento se disponen en orden de generación para componer las capas de la primer red neuronal descendiente. Aquellas generadas por el segundo procedimiento se disponen análogamente para componer las capas de la segunda red neuronal descendiente.

Las funciones de activación de todas las capas (exceptuando la última) se generan con los mismos procedimientos, respectivamente, correspondiéndose con las filas agregadas a las matrices descendientes.

Si las redes neuronales ancestrales difieren en la cantidad de capas, en el caso del primer descendiente se optará por la cantidad de capas del padre de mayor *fitness*. Entretanto,

para el segundo descendiente se hereda con distribución uniforme un tramo de la secuencia de capas sobrantes respecto al padre de menor tamaño. Es necesario destacar que este proceso implica ajustar las dimensiones de las matrices de transición en los bordes de este tramo.

3.4.2. Mutación. Se definen cinco mutaciones independientes para un individuo, asociadas a diferentes partes o niveles de su estructura:

1. Generación de una neurona
2. Eliminación de una neurona
3. Modificación de función de activación
4. Modificación de una conexión
5. Eliminación de una conexión

Las primeras dos mutaciones se definen a nivel de capa, mientras que la tercera ocurre a nivel de neurona y las últimas dos a nivel de conexión. El nivel estructural dónde ocurre una mutación define dónde y cuantas veces esta puede ser aplicada a un mismo individuo durante el proceso de mutación en general. Esto es, una mutación a nivel de capa podría modificar todas las capas de una red, mientras que una mutación a nivel de conexión podría generar cambios en todas las conexiones. Por ejemplo, la mutación de generación de neurona (a nivel de capa), podría generar a lo sumo una neurona en cada capa de una red.

Contar con varias mutaciones en diferentes niveles de la estructura del individuo es una decisión de diseño para localizar las operaciones y definir parámetros con mayor control sobre las mismas. A modo de ejemplo, en vez considerar la probabilidad de que un individuo tenga alguna conexión modificada y sortear uniformemente cuál será esa conexión, se define un parámetro que determina la probabilidad de que una conexión específica sea modificada.

Si bien esto genera parámetros con más influencia sobre el comportamiento del algoritmo, también aumenta la complejidad computacional de las operaciones de mutación. Esto se debe a que todos los elemento del nivel estructural referido (por ejemplo, todas las capas, o todas las conexiones) deben ser recorridos para sortear si ocurre o no la mutación. Además, aumenta significativamente la probabilidad en general de que un individuo sea mutado, puesto que esta se calcula como la suma de las probabilidades involucradas en cada sorteo.

Para suavizar el efecto en el tiempo de ejecución, bien como la probabilidad general de mutación, se define un sorteo previo que determina si el individuo sufrirá o no una mutación. Este sorteo ocurre a nivel de individuo, según un parámetro de mutación general. Idealmente, se debería definir al menos un parámetro general para cada opción de mutación. Sin embargo, para simplificar el conjunto de parámetros del algoritmo se optó por definir un único parámetro de mutación general.

Entonces, el sorteo referente a la mutación de un individuo ocurre en dos etapas. Primero, se sortea según la probabilidad de mutación general si el individuo será mutado. Luego, se recorren los diferentes niveles estructurales para aplicar (con sus respectivas probabilidades) las cinco mutaciones. Los procesos seguidos para aplicar cada mutación depende de cada caso.

Para la generación de neurona, se recorren todas las capas de la red neuronal y, en cada una, se sortea si una nueva neurona será agregada a la capa. En caso positivo, se agrega una nueva a fila al final de la matriz de transición, para representar las conexiones con la capa anterior. Además, se agrega una nueva columna al final de la matriz de transición de la siguiente capa, para representar las conexiones con la capa siguiente.

Los pesos de todas las nuevas conexiones se generan según una distribución normal estándar. Finalmente, se sortea una función de activación con probabilidad uniforme, que es concatenada al final del vector de activación de la capa. Cuando la capa considerada es la última, el procedimiento es el mismo, salvo que la nueva neurona se agrega a una nueva capa, generada en la penúltima posición de la red.

En el caso de la eliminación de una neurona, se recorren todas las capas de la red. Para cada capa, se sortea si una neurona será eliminada. En el caso positivo, se sortea una neurona con probabilidad uniforme y se elimina la fila correspondiente de la matriz de transición de la capa. De la misma forma, se elimina la columna correspondiente en la matriz de transición de la siguiente capa y la posición correspondiente en el vector de activación.

Cuando la capa tiene una única neurona, es necesario eliminar toda la capa para mantener la factibilidad de la red. En este caso, se ajusta la matriz de la capa siguiente para que mantenerla multiplicable con la matriz de la capa anterior. Los pesos se calculan sumando las conexiones con la capa anterior con las conexiones con la capa siguiente. De este modo, una topología como la siguiente:

$$n_1 \xrightarrow{w_1} n_2 \xrightarrow{w_2} n_3$$

Resulta, al eliminar la neurona n_2 y, consecuentemente, toda la capa, en la siguiente topología:

$$n_1 \xrightarrow{w_1+w_2} n_3$$

La modificación y la eliminación de una conexión consisten en recorrer toda las conexiones de la red sorteando si la mutación debe ser aplicada o no. En el caso de la modificación, el nuevo valor del peso se genera con distribución normal estándar. En el caso de la eliminación, se asigna peso 0 al peso de la conexión.

La modificación de una función de activación se realiza con un procedimiento análogo al de la modificación de conexión, excepto que se recorren todos los vectores de activación de la red. En caso de que haya que aplicar la

mutación, los nuevos valores se eligen uniformemente del conjunto de funciones de activación disponibles.

3.4.3. Selección y reemplazo. Se utiliza un esquema de selección por torneo y reemplazo $\mu + \lambda$.

3.5. Parametrización

En la Tabla 1 se presentan todos los parámetros del Algoritmo Evolutivo, con sus respectivos valores iniciales. En el caso de μ, cx_1 y mut_0, los valores iniciales no se definen porque estos parámetros serán definidos mediante configuración paramétrica.

Tabla 1: Parámetros del Algoritmo

Parámetro	Descripción	Valor
μ	Tamaño de la población	-
λ	Cantidad de descendientes por generación	2μ
$ngen$	Cantidad de generaciones	30
mut_0	Probabilidad de mutación	-
mut_1	Probabilidad de generación de neurona	0,05
mut_2	Probabilidad de eliminación de neurona	0,1
mut_3	Probabilidad de modificación de conexión	0,07
mut_4	Probabilidad de eliminación de conexión	0,07
cx_1	Probabilidad de cruzamiento	-
cx_2	Probabilidad de heredar neurona	0,4
$init_1$	Cant. máxima de capas al inicializar	10
$init_2$	Cant. máxima de neuronas al inicializar	5
sel_1	Cant. de individuos participantes por torneo	3
sel_2	Cant. de individuos seleccionados por torneo	1

4. Evaluación experimental

El proceso de evaluación experimental se divide en tres etapas: la configuración de parámetros, en la que se busca optimizar los valores de los parámetros del algoritmo evolutivo; la evaluación del algoritmo, en la que se reportan datos sobre el comportamiento del algoritmo con la configuración escogida; y la de análisis comparativo, en la que se caracteriza la calidad de la solución tomando como referencia otro algoritmo cuya calidad ya fue relevada.

Las instancias del problema utilizadas en esta etapa se presentarán en la sección donde sean relevantes. Cada instancia es un *dataset* y el único requerimiento impuesto al seleccionarlas fue que sus atributos sean númericos y que su propósito sea la clasificación de instancias. La excepción es el *dataset* mushrooms, que debió ser preprocesado para convertir datos categóricos definidos como texto a datos numéricos.

4.1. Configuración paramétrica

En esta etapa se busca analizar y comparar el desempeño del algoritmo con distintas combinaciones de valores para los parámetros configurados. Estos parámetros son el tamaño de población μ, la probabilidad de cruzamiento cx_1 y la probabilidad de mutación mut_0. Los valores considerados para cada parámetros se exponen en la Tabla 2.

Tabla 2: Parámetros que serán configurados.

Parámetro	Descripción	Valores		
μ	Tamaño de población	50	75	100
mut_0	Probabilidad de mutación	0,2	0,4	0,6
cx_1	Probabilidad de cruzamiento	0,4	0,6	0,8

Como es posible observar, se consideran tres opciones de valores para cada uno de los parámetros. Esto resulta en un total de 27 configuraciones posibles cuyo desempeño debe ser comparado en tres instancias (*datasets*) de configuración. Las medidas muestrales que se toman en cuenta durante el proceso de configuración son la media, la mediana, la desviación estándar y el promedio. Se realizan test de hipótesis para inferir sobre la distribución en probabilidad del máximo *fitness* alcanzado por el algoritmo en cada caso.

Debe destacarse que los valores considerados para la probabilidad de mutación son más altos que lo convencional en este tipo de estudio, donde se toma como regla que la mutación ocurre con probabilidad baja. La decisión de considerar valores de probabilidad relativamente alta (especialmente $0{,}4$ y $0{,}6$) considera que, en realidad, el algoritmo diseñado tiene 5 mutaciones independientes entre si. En este contexto, el parámetro configurado caracteriza una variable Bernoulli que condiciona la ocurrencia de cada mutación. Por lo tanto, la probabilidad de mutación configurada se multiplica por las definidas en la Tabla 1.

Los *datasets* utilizados en esta etapa, denominados instancias o *datasets* de configuración, son los siguientes:

1. **Iris:** refiere a la clasificación de plantas iris en 3 especies diferentes en función de cuatro atributos. Posee un total de 150 instancias, donde hay 50 instancias para cada una de las categorías posibles. [3]
2. **Glass**: clasifica tipos de vidrio en 7 categorías diferentes en función de 10 atributos. Posee un total de 214 instancias. [4]
3. **Wine**: representa los resultados de un análisis químico de vinos que provienen de deferentes cultivos (3 posibles categorias)., en función de 13 atributos. Posee un total de 178 instancias. [5]

El proceso de configuración paramétrica está compuesto por cinco etapas:

1. Muestreo
2. Prueba de normalidad
3. Análisis poblacional no paramétrico del máximo *fitness*
4. Reporte de datos muestrales sobre el máximo *fitness*
5. Análisis de la eficiencia computacional

4.1.1. Muestreo. En la etapa de muestreo se define la variable aleatoria X, correspondiente al máximo *fitness* obtenido por el algoritmo en una ejecución. Dado que en la configuración paramétrica se consideran tres parámetros, cada uno con tres valores posibles, se consideran 27 posibles configuraciones: $c_1, \ldots, c_{27}$. En este contexto, es posible definir las variables aleatorias independientes $X_1, \ldots, X_{27}$, dónde X_i corresponde al máximo *fitness* obtenido por el algoritmo en una ejecución con la configuración c_i.

En esta etapa se observan 30 resultados de cada variable X_i, lo que corresponde a ejecutar el algoritmo 30 veces con la configuración c_i. Como las ejecuciones son independientes y este procedimiento se repite sobre cada instancia de configuración, se generan 27 muestras aleatorias de tamaño 30 para cada par $(c_i, dataset)$. El objetivo del proceso de configuración es, entonces, analizar la información obtenida de estas muestras para comparar el desempeño del algoritmo con cada opción de configuración.

Es necesario resaltar que la muestra aleatoria obtenida para la i-ésima configuración y un *dataset* d tiene la forma $X_{i_1}(d), \ldots, X_{i_n}(d)$ $iid \sim X_i(d)$. La inferencia estadística se realiza sobre la variable poblacional $X_i(d)$ correspondiente a un *dataset* específico. Luego, las poblaciones generales X_i se comparan combinando los resultados específicos para cada *dataset*.

4.1.2. Prueba de normalidad. El primer paso para estudiar el comportamiento en población del algoritmo es realizar una prueba de distribución que determina la naturaleza del análisis poblacional a ser realizado. La prueba utilizada fue el test de Anderson-Darling, que considera la hipótesis nula de que una muestra proviene de una población Normal.

La implementación utilizada (`anderson`, de `scipy.stats`) devuelve el valor del estadístico $A(\vec{x_i})$ y un conjunto de valores críticos para diferentes niveles de significación. Para todas las muestras obtenidas, A excede el valor crítico incluso para un nivel de significación de $15\,\%$, por lo que no se puede afirmar con confianza al $85\,\%$ que alguna de las $X_i(d)$ distribuye con normalidad.

4.1.3. Análisis poblacional. La falta de normalidad en los datos establece la necesidad de realizar un análisis no paramétrico de la población, para lo que se emplean los contrastes de hipótesis de Kruskal-Wallis y de Mann-Whitney. El nivel de significación considerado a partir de esta etapa es de $5\,\%$.

El contraste de Kruskal-Wallis toma en consideración un conjunto de muestras independientes y considera la hipótesis nula de que no existe dominancia estocástica entre ninguna de ellas. La dominancia estocástica establece un órden parcial sobre las variables aleatorias, de modo que, si X domina a Y, la distribución de X se acumula sobre valores mayores a aquellos sobre los que se acumula la distribución de Y. Esto introduce, a su vez, una noción de orden sobre las medianas de X e Y que puede ser utilizada para inferir sobre sus distribuciones.

La implementación de Kruskal-Wallis utilizada (`kruskal` de `scipy.stats`) resulta en el valor del estadístico $H(\vec{x_1}, \ldots, \vec{x_{27}})$ y en el p-valor asociado a la muestra (en este caso, el conjunto de muestras $\vec{x_1}, \ldots, \vec{x_{27}}$). En la Tabla 3 se reportan los p-valores calculados para cada *dataset*.

Tabla 3: Resultados del contraste de Kruskal-Wallis

Dataset	**p-valor**	**Resultado**
iris	0,000000086	Significativo
wine	0,000588544	Significativo
glass	0,000000207	Significativo

Como es posible observar, los p-valores resultantes del test de Kruskal-Wallis para todos los *dataset* son aproximadamente 0, por lo que las muestras proveen evidencia suficiente para rechazar la hipótesis nula con el nivel de significación de $5\,\%$ definido previamente. Esto quiere decir que con confianza $95\,\%$, o incluso muy cercana a $100\,\%$, es posible afirmar que existen relaciones de dominancia estocástica entre las muestras.

Una vez interpretado este resultado, se utiliza el contraste de Mann-Whitney para probar la dominancia estocástica entre los pares $\{(X_i(d), X_j(d)) : i \neq j\}$. Se optó por utilizar este test, en vez de Kruskal-Wallis con solamente dos muestras, porque la implementación utilizada (`mannwhitneyu` de `scipy.stats`) permite modificar la hipótesis alternativa de modo a determinar cual de las dos variables aleatorias es la dominante.

El resultado de la prueba de Mann-Whitney es el valor del estadístico $U(\vec{x_i}, \vec{x_j})$ que caracteriza la región crítica y el p-valor que cuantifica la evidencia aportada por la muestra para rechazar la hipótesis nula de que $X_i(d)$ domina estocásticamente a $X_j(d)$. Sea $p(\vec{x_i}, \vec{x_j})$ el p-valor asociado a las muestras $\vec{x_i}$ y $\vec{x_j}$, como se exige un nivel de significación de $5\,\%$, podemos afirmar que X_i domina a X_j si $p(\vec{x_i}, \vec{x_j}) \leq 0{,}05$.

La interpretación del contraste de Mann-Whitney permite ordenar las variables aleatorias $X_i(d)$ según la dominancia estocástica, por lo que es posible, también, encontrar los supremos de la relación de orden establecida. El conjunto de supremos de la relación corresponde a las variables aleatorias $X_i(d)$ que, en el *dataset* d, acumulan más probabilidad sobre valores más altos del máximo *fitness* alcanzado.

Como fue mencionado anteriormente, la dominancia estocástica introduce una noción para la comparación de medianas en probabilidad. Se resalta que la propiedad "X domina a Y entonces $m_X > m_Y$", siendo estas últimas las medianas de X e Y, es cierta únicamente al asumir que ambas distribuciones tienen la misma forma. De todos modos, se adopta esta noción reconociendo que, *a priori*, bajo las condiciones de este experimento, $m_X > m_Y$ no es seguro pero ocurre con probabilidad alta.

Entonces, el procedimiento diseñado emplea el contraste de Mann-Whitney para determinar un conjunto de configuraciones que, para cada *dataset*, es probable que generen resultados de máximo *fitness* con medianas más altas. Una virtud de utilizar la mediana para realizar la comparación es

Tabla 4: Datos muestrales de las configuraciones dominantes (máximo *fitness* alcanzado)

Configuración			iris				wine				glass			
μ	cx_1	mut_0	$x_{(30)}$	$\overline{x}$	s_{30}	m	$x_{(30)}$	$\overline{x}$	s_{30}	m	$x_{(30)}$	$\overline{x}$	s_{30}	m
75	0,4	0,4	**0.993**	0,921	0,100	0,967	**0.894**	0,676	0,079	0,692	0,845	**0.721**	**0.052**	0,719
75	0,6	0,6	0,987	0,951	**0.026**	0,960	0,753	0,687	**0.045**	0,692	0,772	0,683	0,056	0,686
100	0,4	0,4	0,987	0,925	0,086	0,967	0,770	0,674	0,060	0,686	0,813	0,704	0,078	**0.727**
100	0,4	0,6	0,987	0,953	0,035	0,960	0,787	**0.698**	0,055	**0.721**	0,836	0,716	0,063	0,734
100	0,6	0,2	0,987	0,953	0,048	0,967	0,876	0,696	0,078	0,714	0,838	0,693	0,077	0,691
100	0,6	0,4	**0.993**	**0.957**	0,041	**0.970**	0,791	0,683	0,066	0,688	0,793	0,703	0,066	0,729
100	0,8	0,2	**0.993**	0,952	0,054	0,967	0,758	0,681	0,059	0,696	**0.923**	0,705	0,079	0,712

que es una medida de tendencia central más robusta que el promedio, que a su vez puede ser menos adecuado cuando la distribución es asimétrica.

4.1.4. Reporte de datos muestrales. Obtenidas las distribuciones dominantes para un determinado *dataset*, es necesario generalizar el criterio de selección para que no dependa de la instancia de configuración considerada. Se procede intersectando el conjunto de supremos en dominancia hayado para cada *dataset*. La Tabla 4 contiene las configuraciones resultantes de este proceso, así como estadísticos calculados de sus respectivas muestras en cada *dataset*. En este contexto, $x_{(30)}$ corresponde al máximo muestral, $\overline{x}$ es el promedio de la muestra, s_{30} es su desvío estándar y m su mediana.

Dado que el procedimiento anterior resultó en un conjunto de configuraciones cuyas medianas no presentan diferencias estadísticamente significativas, es necesario establecer un criterio de desempate para seleccionar una única configuración definitiva. Entonces, se retoma la definición de la variable X como el máximo *fitness* resultante de ejecutar una vez el algoritmo sobre algún *dataset*.

Son características deseables de X, además de que su mediana sea alta, que su media sea alta y que su desvío sea bajo. Se toma en consideración que el valor máximo que puede tomar la función de *fitness* es 1, dato que podría ser utilizado para interpretar el valor del máximo de la muestra.

Si bien desconocer la distribución de X imposibilita la estimación por intervalos de confianza de la media $\mathbb{E}(X)$, la Ley de los Grandes Números garantiza que el promedio muestral $\overline{X}$ es un estimador puntual consistente para la misma. Esto, combinado con que el tamaño muestral de 30 observaciones es razonablemente grande, permite utilizar el valor de $\overline{x}$ como un criterio complementario al de la comparación de medianas.

Los datos reportados en la Tabla 4 son consistentes con el criterio de comparación utilizado para seleccionar el conjunto de configuraciones, puesto que hay poca diferencia entre los valores de las medianas. También es posible observar que, para un mismo *dataset*, el algoritmo desempeña de forma similar en promedio y desviación al variar la configuración.

Las principales diferencias de desempeño se observan al cambiar de *dataset*. En general, el algoritmo presenta desempeño excelente sobre iris, aunque la configuración ($\mu = 100,\ cx_1 = 0{,}6,\ mut_0 = 0{,}4$) destaca según el promedio y presenta un desvío considerablemente bajo, aunque no es el menor. Sobre los otros *datasets*, el desempeño promedio es considerablemente menor, pero el algoritmo presenta robustez según los desvíos.

El *dataset* iris es conocido por la calidad de su datos, lo que explica la diferencia de desempeño respecto a las otras instancias, en las que el algoritmo presenta resultados homogéneos, a pesar de que sean peores.

4.1.5. Análisis de eficiencia computacional. En general, es esperable que las configuraciones seleccionadas mediante la comparación de medianas no difieran demasiado en desempeño. Por lo tanto, otras medidas, como el tiempo de ejecución o el tamaño promedio de los individuos generados, serían de utilidad para el desempate. En este caso, se muestreó el tiempo de ejecución del algoritmo empleando un procedimiento semejante al utilizado para obtener datos sobre el máximo *fitness* alcanzado para cada configuración.

Tabla 5: Especificaciones de la plataforma de pruebas.

Elemento	**Descripción**
Procesador	Intel Core i7-3630QM. Quad-core. 2.40GHz
Memoria RAM	8GB
Sistema operativo	Ubuntu 18.04.3 LTS
Lenguaje	Python 3.6.9

En este contexto, se consideran siete variables aleatorias $T_1(d), \ldots, T_7(d)$ correspondientes a los tiempos de ejecución del algoritmo para un determinado *dataset*, en cada una de las configuraciones dominantes obtenidas mediante la comparación de medianas. Estos datos se exponen en la Tabla 6 para los *datasets* de configuración. El tamaño de cada muestra es 30 y los estadísticos $t_{(0)}$, $\overline{t}$, s_{30} y m corresponden al mínimo, el promedio, el desvío y la media, respectivamente.

Las especificaciones técnicas de la plataforma computacional sobre los cuales se realizó el muestreo se especifican en la Tabla 5. Se distribuyó la evaluación de las instancias en todos los núcleos del procesador empleado, con auxilio de la librería SCOOP (Scalable COncurrent Operations in Python). Algunas bibliotecas utilizadas relevantes para este análisis son Numpy 1.15.1, para la representación de matrices, SCOOP 0.7, para la distribución de la evaluación y DEAP 1.3, para la implementación del ciclo evolutivo.

Tabla 6: Datos muestrales de las configuraciones dominantes (tiempo de ejecución en segundos)

Configuración			**iris**				**wine**				**glass**			
μ	cx_1	mut_0	$t_{(0)}$	$\bar{t}$	s_{30}	m	$t_{(0)}$	$\bar{t}$	s_{30}	m	$t_{(0)}$	$\bar{t}$	s_{30}	m
75	0,4	0,4	72	**96**	18	**93**	**172**	**254**	40	250	153	**208**	**28**	**202**
75	0,6	0,6	100	128	29	122	312	369	39	373	**147**	271	43	278
100	0,4	0,4	**40**	908	4366	117	281	348	42	339	253	289	37	273
100	0,4	0,6	130	170	34	161	388	472	47	478	291	359	52	345
100	0,6	0,2	82	100	**16**	96	195	227	**22**	**221**	191	230	32	218
100	0,6	0,4	116	145	26	138	308	348	27	343	278	321	34	315
100	0,8	0,2	45	102	24	105	204	299	76	286	156	245	29	245

Los datos de la Tabla 6 destacan el desempeño en tiempo de ejecución de de la configuración ($\mu = 75,\ cx_1 = 0{,}4,\ mut_0 = 0{,}4$). Esto es razonable, al tratarse de la configuración con menor tamaño de población y que ejecuta el cruzamiento y la mutación, ambas operaciones de alta complejidad computacional, con menos probabilidad. Sin embargo, para tomar una decisión respecto a qué configuración elegir es necesario utilizar los datos de la muestra para inferir sobre la distribución de la población de la que provienen.

De la misma forma que en la parte anterior, se realiza el test de normalidad de Anderson-Darling para determinar si los datos son normales. Esto se verifica comparando el valor estadístico $A(\vec{x})$, retornado por el test, con los valores críticos en los distintos niveles de significación. Si el valor del estadístico es mayor que el valor crítico, entonces se puede afirmar con ese nivel de significación que la variable no distribuye con normalidad.

Tabla 7: Valores críticos de Anderson-Darling

Significación	15 %	10 %	5 %	2,5 %	1 %
Valor crítico	0,521	0,593	0,712	0,83	0,988

Tabla 8: Valores de $A(\vec{x})$ para cada configuración.

Configuración			$A(\vec{x})$		
μ	cx_1	mut_0	**iris**	**wine**	**glass**
75	0,4	0,4	0,675	0,510	0,541
75	0,6	0,6	2,275	0,650	0,399
100	0,4	0,4	10,847	0,556	2,808
100	0,4	0,6	1,514	0,596	0,972
100	0,6	0,2	1,245	0,393	1,376
100	0,6	0,4	1,226	0,688	0,687
100	0,8	0,2	1,108	0,676	0,348

Tabla 9: Conclusiones del test para significación 5 %.

Configuración			**Población normal**		
μ	cx_1	mut_0	**iris**	**wine**	**glass**
75	0,4	0,4	Sí	Sí	Sí
75	0,6	0,6	No	Sí	Sí
100	0,4	0,4	No	Sí	No
100	0,4	0,6	No	Sí	No
100	0,6	0,2	No	Sí	No
100	0,6	0,4	No	Sí	Sí
100	0,8	0,2	No	Sí	Sí

En la Tabla 8 es posible observar los valores de $A(\vec{x})$ obtenidos para cada configuración en cada *dataset*. Considerando como referencia el nivel de significación de 5 % definido anteriormente, las conclusiones de la prueba de Anderson-Darling se exponen en la Tabla 9.

Al verificar los resultados de la prueba de Anderson-Darling es posible observar que los tiempos de ejecución de todas las configuraciones consideradas en el *dataset* wine distribuyen con normalidad. Asimismo, esto también se cumple para algunas configuraciones en el caso del *dataset* glass y para solamente una en el caso de iris.

En todos estos casos, es posible calcular intervalos de confianza para las medias reales. Estos estadísticos están expuestos en la Tabla 10 y permiten comparar la eficiencia computacional de algunas configuraciones en determinados *dataset*.

Razonablemente, la configuración ($\mu = 75,\ cx_1 = 0{,}4,\ mut_0 = 0{,}4$) también se detaca en estos resultados, puesto que el cálculo del intervalo de confianza involucra los valores del promedio y del desvío muestral. Además de esta configuración, se destacan ($\mu = 100,\ cx_1 = 0{,}6,\ mut_0 = 0{,}2$) en wine y ($\mu = 100,\ cx_1 = 0{,}8,\ mut_0 = 0{,}2$) tanto en wine como en glass.

Aunque los intervalos de confianza permitan identificar algunas tendencias en el comportamiento del algoritmo bajo condiciones específicas, el hecho de que no todas las muestras provengan de una distribución Normal impide la comparación equitativa y motiva la utilización de un análisis no paramétrico.

En este caso, se emplea el mismo procedimiento de comparación medianas utilizado para comparar las configuraciones según el máximo *fitness*. Debido a que al analizar la eficiencia computacional es deseable que los tiempos de ejecución sean menores, se adapta el método para el test de Mann-Whitney pruebe la relación inversa a la dominancia estocástica. Esto es, se pasa a considerar que X domina a Y si X que tiende a tomar valores menores que Y.

Nuevamente, se consideran los resultados del test de hipótesis con un nivel de significación de 5 %, por lo

Tabla 10: Intervalos de confianza al 95 % para el tiempo de ejecución.

Configuración			Intervalo de confianza al 95 %		
μ	cx_1	mut_0	**iris**	**wine**	**glass**
75	0,4	0,4	[89,56, 102,44]	[239,69, 268,31]	[197,98, 218,02]
75	0,6	0,6	-	[355,04, 382,96]	[255,61, 286,39]
100	0,4	0,4	-	[332,97, 363,03]	-
100	0,4	0,6	-	[455,18, 488,82]	-
100	0,6	0,2	-	[219,13, 234,87]	-
100	0,6	0,4	-	[338,34, 357,66]	[308,83, 333,17]
100	0,8	0,2	-	[271,80, 326,20]	[234, 62, 255, 38]

que en general podemos afirmar un nivel de confianza de $95\,\%$ sobre el resultado final del proceso. Al aplicar este algoritmo sobre las siete configuraciones dominantes según el máximo *fitness*, utilizando los datos obtenidos sobre el tiempo de ejecución, se llega a una única configuración: $(\mu = 100,\ cx_1 = 0{,}8,\ mut_0 = 0{,}2)$.

La interpretación de este resultado es que, con confianza al $95\,\%$, es posible considerar que esta configuración tiende a tomar valores menores que las otras. Debido a que este método sí toma en cuenta todas las configuraciones y todos los *dataset*, se elige esta configuración para proseguir con el análisis experimental. La elección es consistente con el resto de los resultados obtenidos, puesto que esta configuración es dominante respecto al máximo *fitness*, presentando buenos resultados muestrales. Además, a pesar de no alcanzar los mejores valores en el análisis parametrico de la eficiencia computacional, obtiene resultados destacables.

4.2. Evaluación del algoritmo

Una vez elegida una configuración definitiva para el algoritmo, se evalúa el desempeño del mismo en instancias de mayor tamaño. Las instancias utilizadas para esta parte, denominadas instancias de evaluación, son las siguientes:

1. **Mushrooms:** refiere a la clasificación de plantas hongos en comestibles o venenosos a partir de 22 atributos. Este *dataset* fue reducido de su tamaño original de 8124 ejemplos a 1500. [3]
2. **Wine Quality**: refiere a un estudio químico de vinos, clasificándolos en 10 categórias según su calidad. Cada ejemplo tiene 12 atributos. Este *dataset* está dividio en dos: Wine Quality Red, con 1599 ejemplos y Wine Quality White, con4898 ejemplos. [4]

El proceso seguido para caracterizar el desempeño del algoritmo es semejante al de la configuración paramétrica. Primero, se generaron muestras de 30 observaciones para cada *dataset*. Cada observación es el máximo *fitness* obtenido al ejecutar el algoritmo con la configuración propuesta sobre el dataset indicado. Luego, se realizó el contraste de hipótesis de Anderson-Darling para determinar si los datos relevados provienen de distribuciones normales.

Los valores del estadístico $A(\vec{x})$ para mushrooms, wine-quality-red y wine-quality-white son $0{,}464$, $0{,}420$ y $0{,}420$ y en la Tabla 7 es posible observar los valores críticos del test de Anderson-Darling por nivel de significación. De estos datos se concluye, con confianza al $95\,\%$, que las tres muestras provienen de una población Normal. Es más, ni admitiendo probabilidad de error del $15\,\%$ es conveniente rechazar esta hipótesis.

En la Tabla 11 se exponen los estadísticos calculados a partir de las muestras generadas para cada *dataset*. Nuevamente, $x_{(30)}$ es el máximo, $\overline{x}$ es el promedio, s_{30} es el desvío estándar y m es la mediana. Asimismo, en la Tabla 12 se muestran los intervalos de confianza al $95\,\%$ calculados para estimar la media del máximo *fitness* alcanzado por el algoritmo en cada *dataset*.

Tabla 11: Datos muestrales para la configuración elegida.

Dataset	$x_{(30)}$	$\overline{x}$	s_{30}	m
mushrooms	0,995	0,954	0,029	0,962
wine-quality-red	0,495	0,470	0,011	0,468
wine-quality-white	0,405	0,390	0,010	0,389

Tabla 12: Intervalos de confianza al 95 % para la configuración elegida.

Dataset	**Intervalo de confianza al 95 %**
mushrooms	[0,952, 0,956]
wine-quality-red	[0,469, 0,471]
wine-quality-white	[0,389, 0,391]

Los datos obtenidos muestran poca robustez del algoritmo diseñado al variar la instancia del problema. Mientras que los resultados son excelentes (semejantes a aquellos de iris) para el *dataset* mushrooms, en las instancias de wine-quality se obtuvieron resultados peores de lo esperado. Estos resultados podrían ser evidencia de diferencias sustanciales en el desempeño en *datasets* con características específicas. Por ejemplo, mushrooms es un dataset de clasificación binaria, mientras que las instancias de wine-quality están clasificadas entre diez posibles categorías extremadamente desbalanceadas.

De todas formas, la reducida amplitud de los intervalos de confianza para el tamaño de muestra de 30 observaciones,

Tabla 13: Parámetros para entrenamiento de redes con *backpropagation*

Parámetro	Descripción	Valores
`solver`	El solver para la optimización de pesos	`sgd`
`alpha`	Término de regularización	$1e-5$
`random_state`	Semilla usada por el generador de números pseudoaleatórios	1
`max_iter`	Número máximo de iteraciones	300
`learning_rate`	Tipo de aprendizaje para actualizaciones de pesos	`adaptive`

consecuencia de los bajos valores del desvío estándar, es evidencia de la robustez del algoritmo para una misma instancia del problema.

4.3. Análisis comparativo

Para esta etapa se definieron y entrenaron tres redes neuronales con el algoritmo de *backpropagation*. Este algoritmo implementa una técnica de optimización mediante descenso por gradiente y es ampliamente utilizado en la industria por su eficiencia. Es necesario destacar que este algoritmo solamente se utiliza para ajustar los pesos de redes neuronales cuya topologia está prefijada. Por lo tanto, *backpropagation* no resuelve exactamente el problema abordado, y sí una parte de este. Por este motivo, solamente se compara el desempeño de las redes neuronales obtenidas respecto al *fitness*, omitiéndose la comparación respecto a la eficiencia computacional.

La definición de las topologías de las redes neuronales entrenadas con *backpropagation* se realiza buscando la experimentación con casos borde respecto al tamaño de la red. Por lo tanto, se definen las siguientes capas implícitas para las redes:

1. **Pequeña**: 1 capa con 1 neurona.
2. **Grande**: 2 capas con 1000 neuronas cada una.
3. **Profunda**: 7 capas con 10 neuronas cada una.

La implementación de *backpropagation* utilizada es la de Scikit-Learn [8], que abstrae el concepto de una red neuronal en Python utilizando la clase `MLPCLassifier` [9]. Los parámetros definidos para generar las redes neuronales se muestran en la Tabla 13, para el resto de la configuración se usaron los valores por defecto de la librería.

Las redes neuronales fueron entrenadas sobre los mismos *datasets* de evaluación del algoritmo evolutivo: mushrooms, wine-quality-red y wine-quality-white. El cálculo del *fitness* (la medida F_1) para cada una de las redes se realizó sobre *datasets* de verificación diferentes a los de entrenamiento. Los *datasets* de entrenamiento y verificación se obtuvieron a partir de los originales, preservando la distribución de las instancias por clase.

Tabla 14: F_1 de las redes entrenadas por *backpropagation*.

Red	Dataset		
	mushrooms	wine-quality-red	wine-quality-white
Pequeña	0,829	0,252	0,278
Grande	1,000	0,511	0,443
Profunda	1,000	0,559	0,434

Los resultados obtenidos por estas redes están expuestos en la Tabla 14. Al comparar estos resultados con los del algoritmo evolutivo, presentes en la Tabla 12, es posible observar que el desempeño de la red grande y de la red profunda entrenadas con *backpropagation* es levemente superior que el alcanzado por el algoritmo evolutivo. Asimismo, el algoritmo evolutivo es levemente mejor que la red pequeña.

Estos resultados no resultan alentadores para el algoritmo evolutivo propuesto, puesto que redes neuronales con topología definida mediante la intuición lograron superar su desempeño. De todos modos, en esta comparación no está involucrada únicamente la capacidad del algoritmo para obtener una topología (sub)óptima. También está implícito el desempeño del mismo al entrenar una red neuronal. Por este motivo, podría considerarse la posibilidad de usar un algoritmo evolutivo para generar la topología de la red y ajustar los pesos con *backpropagation*, a modo de combinar y potenciar ambas técnicas.

5. Conclusión

En este trabajo se presentó el problema de la neuroevolución, instanciado en redes neuronales de clasificación multiclase y se definió un modelo de redes neuronales que sirvió de base para proponer una solución utilizando algoritmos evolutivos.

Una vez diseñado el algoritmo evolutivo propuesto para resolver el problema, se realizó un proceso de configuración paramétrica en el cual participaron el tamaño de la población, la probabilidad de cruzamiento y la probabilidad de mutación. Este proceso incluyó análisis no paramétricos sobre muestras del máximo *fitness* alcanzado por el algoritmo y del tiempo de ejecución bajo diversas configuraciones y sobre *datasets* pequeños denominados instancias de configuración. De este proceso se eligió una configuración definitiva para proseguir con la evaluación experimental.

En seguida, se reportaron los resultados del algoritmo para otros *datasets* de mayor tamaño, denominados instancias de evaluación. En este caso, fue posible constatar la normalidad de la distribución del máximo *fitness* alcanzado y, en consecuencia, realizar un análisis paramétrico. Como resultado de este proceso, se obtuvieron intervalos de confianza al $95\,\%$, de amplitud considerablemente baja para

el máximo *fitness* que el algoritmo alcanza en promedio para cada instancia de evaluación. De estos estadísticos se concluyó la robustez del algoritmo al ser ejecutando un mismo *datasets*, pero también se observó la variabilidad de su desempeño al cambiar de instancia del problema.

Finalmente, se entrenaron redes neuronales con *backpropagation* para realizar un análisis comparativo. La topologia de estas redes fue fijada para cubrir casos bordes: una red pequeña, una red grande (pocas capas, muchas neuronas) y una red profunda (muchas capas). Dado que el algoritmo evolutivo obtiene la topología de una red, se destacó que *backpropagation* no resuelve el mismo problema y sí una parte: el entrenamiento. Por este motivo, la comparación se realizó solamente respecto al *fitness* de las redes obtenidas, omitiendo la eficiencia computacional.

A pesar de que los algoritmos resuelvan problemas diferentes, los resultados fueron relevante y no favorecieron al algoritmo evolutivo. Esto se debe a que, al fijar una topología con un criterio relativamente arbitrario, *backpropagation* las entrenó para que obtengan mejores resultados que el algoritmo evolutivo. Es necesario destacar que esto no permite diagnosticar sobre la capacidad del algoritmo evolutivo para obtener topologías y sí sobre su desempeño al entrenarlas. Combinar ambas técnicas, utilizando un algoritmo evolutivo para obtener la red y *backpropagation* para entrenarla podría ser un caso de estudio interesante.

También se identificaron altos tiempos de ejecución para el algoritmo evolutivo. Esto es razonable debido la naturaleza del problema, dado que la evaluación de instancias presenta una alta complejidad computacional. Asimismo, al tratarse de individuos complejos, los operadores evolutivos (especialmente la mutación y el cruzamiento) presentaron elevados tiempos de ejecución. Esto dificultó la realización de pruebas iterativas que permitiesen refinar aspectos del algoritmo o, inclusive, de la evaluación experimental, para llegar a mejores resultados.

Investigar otras maneras de distribuir el algoritmo (no solamente el proceso de evaluación), utilizar estructuras de datos rápidamente clonables y transformables (al contrario de Numpy) e inclusive utilizar un lenguaje de programación más eficiente (como C o Java) para la implementación, son medidas posibles para reducir los tiempos de ejecución del algoritmo. Las políticas de selección de individuos para la mutación también inciden considerablemente en el tiempo de ejecución.

Para obtener mejores resultados, podrían refinarse los operadores al algoritmo evolutivo propuesto. Esto incluye la modificación de las políticas de selección de individuos para cruzamiento y mutación, la configuración de más parámetros y aumentar el énfasis en los procesos elitistas.

Finalmente, se considera que, a pesar de que el desempeño final y la eficiencia computacional del algoritmo fueron deficientes, se obtuvieron resultados interesantes. La complejidad del problema permitió diseñar un algoritmo con representación de individuos y operadores poco convencionales. Asimismo, se realizó un análisis estadístico lo más detallado posible dados los recursos computacionales disponibles. Si bien las barreras encontradas no permitieron realizar un refinamiento muy profundo del algoritmo, se identificaron diversos problemas y posibles mejoras.

Referencias

[1] DEAP, 2019. *DEAP 1.3.0 documentation*. Disponible en: https://deap.readthedocs.io/.

[2] Shung, 2018. *Accuracy, Precision, Recall or F1?*. Disponible en: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

[3] UCI - Machine Learning repository. *Iris Data Set*. https://archive.ics.uci.edu/ml/datasets/iris

[4] UCI - Machine Learning repository. *Glass Identification Data Set*. Disponible en: https://archive.ics.uci.edu/ml/datasets/glass+identification

[5] UCI - Machine Learning repository. *Wine Data Set*. Disponible en: https://archive.ics.uci.edu/ml/datasets/Wine

[6] UCI - Machine Learning repository. *Wine Quality Data Set*. Disponible en: https://archive.ics.uci.edu/ml/datasets/Wine+Quality

[7] UCI - Machine Learning repository. *Mushroom Data Set*. Disponible en: https://archive.ics.uci.edu/ml/datasets/Mushroom

[8] Sklearn - Machine Learning in Python. *Scikit-learn*. Disponible en: https://scikit-learn.org/

[9] Sklearn - Machine Learning in Python. *MLPClassifier*. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[10] Heidenrich, 2019. *NEAT: An Awesome Approach to NeuroEvolution*. Disponible en: https://towardsdatascience.com/neat-an-awesome-approach-to-neuroevolution-3eca5cc7930f