Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Programación genética para la generación automática de prototipos de clasificación

por

Karlo Mario Mendoza Mendoza

en opción al grado de

Maestro en Ciencias

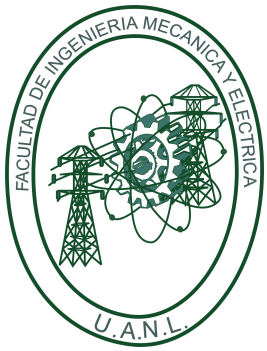
en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León febrero 2012

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado



Programación genética para la generación automática de prototipos de clasificación

por

Karlo Mario Mendoza Mendoza

en opción al grado de

Maestro en Ciencias

en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León febrero 2012

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis «Programación genética para la generación automática de prototipos de clasificación», realizada por el alumno Karlo Mario Mendoza Mendoza, con número de matrícula 1402454, sea aceptada para su defensa como opción al grado de Maestro en Ciencias en Ingeniería de Sistemas.

El Comité de Tesis

Dr. José Arturo Berrones Santos
Director

Dr. Hugo Jair Escalante Balderas
Codirector

Dr. Oscar Leonel Chacón Mondrágon
Revisor

Vo. Bo.

Dr. Moisés Hinojosa Rivera
División de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, febrero 2012

Dedicatoria

Dedico este trabajo a mis padres por toda la confianza que me brindaron.

Y a Alexandra por toda la ayuda prestada a lo largo del camino.

Agradecimientos

Agradezco a la Facultad de Ingeniería Mecánica y Eléctrica y a la Universidad Autónoma de Nuevo León por el apoyo financiero otorgado para la realización de mi maestría.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) quien me otorgo una beca a lo largo de mi maestría.

Al Posgrado de Ingeniería de Sistemas y sus profesores quienes me ayudaron a mi formación académica y me brindaron su tiempo y paciencia.

Mi más profundo agradecimiento al Dr. Hugo Jair Escalante Balderas por todo el tiempo, conocimiento y paciencia que me brindó en la realización de este trabajo.

Resumen

Karlo Mario Mendoza Mendoza.

Candidato para el grado de Maestro en Ingeniería
con especialidad en Ingeniería de Sistemas.

Universidad Autónoma de Nuevo León.
Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio:

Programación genética para la generación automática de prototipos de clasificación

Número de páginas: 112.

Objetivos y método de estudio: El presente trabajo de tesis está enfocado en el estudio de la clasificación automática por medio de la generación de prototipos de clasificación. Para abordar este problema se decidió utilizar la programación genética para generar de forma automática los prototipos de clasificación. Éstos prototipos junto a un enfoque de un vecino más cercano (*1-NN*) forman el modelo de clasificación propuesto en esta tesis.

De esta forma se propone un enfoque que pueda generar modelos de clasificación que permita clasificar adecuadamente nuevas instancias y al mismo tiempo reducir el numero de observaciones que se requieren para clasificar nuevos datos.

Con base en lo anterior, los objetivos en esta tesis son el desarrollar e implementar una metodología de solución que utilice la programación genética (PG) para la generación automática de prototipos, y realizar experimentos computacionales para validar la metodología desarrollada.

CONTRIBUCIONES Y CONCLUSIONES: La contribución fundamental de este trabajo es el desarrollo e implementación de un enfoque de programación genética para la tarea de clasificación automática que permita clasificar adecuadamente nuevas instancias y a la vez reducir el número de observaciones que se requieren para clasificar nuevas instancias. Además de generar nuevos prototipos con altos índices de efectividad de clasificación, el método propuesto permite incrementar la cantidad de instancias para conjuntos en los que se tengan pocos ejemplos, añadiendo los prototipos generados con las instancias del conjunto de datos. Además de reducir de forma considerable el tiempo de cómputo bajo el esquema de clasificación (*1-NN*). Lo cual hace que la metodología adoptada en esta tesis, sirva además de una forma de desarrollar modelos de clasificación como un generador de prototipos.

Firma del director: _____

Dr. José Arturo Berrones Santos

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Descripción del problema

En este trabajo se estudia el problema de clasificación supervisada usando programación genética. Se busca mediante la evolución de instancias realizar una generación de prototipos que permita clasificar adecuadamente nuevas instancias y al mismo tiempo reducir el número de observaciones que se requieren para clasificar nuevos datos.

En el problema de clasificación supervisada se tiene un conjunto de observaciones o instancias que están asociadas a una de k-clases posibles. Las instancias son pares ordenados en los cuales la primera entrada es un vector de atributos y la segunda entrada una etiqueta asignada a cada vector, esta etiqueta nos dice a que clase pertenece cada uno de éstos vectores. Los pares de *(observación, clase)* se dividen en conjuntos de entrenamiento y prueba. La tarea en clasificación supervisada consiste en crear un modelo de clasificación, usando el conjunto de entrenamiento, que debe ser capaz de determinar a qué clase pertenece un vector que no haya sido visto con anterioridad (i.e. no incluido en el conjunto de entrenamiento).
La programación genética es una rama de computación evolutiva en la que las estructuras que se evolucionan generalmente están representadas por árboles, que son grafos acíclicos como el ilustrado en la figura 1.1, donde la representación del árbol nos indica las operaciones que deben seguirse para evaluar a cada árbol.

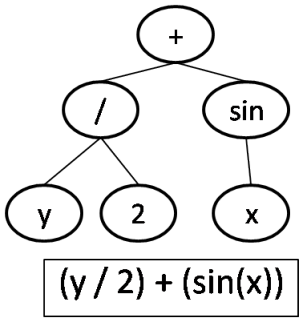


Figura 1.1: Un árbol en PG así como su representación.

La programación genética ha sido usada con anterioridad para trabajar problemas de clasificación (Cordella *et al.*, 2006b), (Eggermont *et al.*, 2004), (Muni *et al.*, 2004), entre otros.

En este trabajo se usan instancias en los nodos de los árboles para abordar el problema de clasificación. Al utilizar instancias en los nodos hace que nuestros árboles representen un punto en el espacio de atributos los cuales pueden usarse como prototipos. Un prototipo es un ejemplo representativo de un conjunto de datos que pertenece a la misma clase, el cual puede ser usado para distinguir a otras instancias de su misma clase bajo la premisa de que "instancias parecidas entre sí deben pertenecer a la misma clase". En generación de prototipos se busca obtener un número pequeños de prototipos para realizar la clasificación.

Una de las ventajas de obtener prototipos de clasificación es que estos se pueden usar para la interpretación de los datos. Por ejemplo, se puede decir que clase se suele confundir con otra (cuando los prototipos son muy parecidos), que tantas subclases puede contener una clase (analizando en número de prototipos generados por cada clase, entre más prototipos más subclases podrían estar presentes).

Cabe mencionar que este enfoque tiene como característica adicional que al variar la dimensión de los conjuntos el tamaño de los árboles generados por el método desarrollado en esta tesis no cambiara significativamente.

1.2 Hipótesis

La programación genética puede usarse para la generación de prototipos de clasificación que permita clasificar adecuadamente nuevas instancias y al mismo tiempo reducir el número de observaciones que se requieren para clasificar nuevos datos.

La programación genética (PG) ha sido usada con efectividad en el problema de clasificación supervisada, a su vez el uso de prototipos en la tarea de clasificación ha resultado ser efectiva. La hipótesis de esta tesis es que el uso de PG para generar prototipos es benéfico.

1.3 Justificación

En la actualidad se generan grandes cantidades de datos, los cuales contienen información que puede de ser de gran utilidad para las empresas, gobierno, ciencia, o la población en general. El problema radica en que la cantidad de información generada rebasa la capacidad de asimilación que tiene el ser humano, esto es, no es capaz de asimilar los conocimientos que se encuentran en estas bases de datos. En estos casos la clasificación automatizada juega un papel muy importante en la extracción de estos conocimientos. Tareas como determinar si es probable que cierta audiencia se interese en nuestro producto cuando se les ofrece una promoción determinada; determinar la forma de manejar un coche por cierta persona para conocer el riesgo de que éste tenga un accidente y poder cobrarle una cantidad adecuada con base en eso; poder determinar si una nota suicida fue en realidad escrita por la persona en cuestión o si fue escrita por alguien más, etc. En todas estas tareas el uso de la clasificación automática ayuda al ser humano a realizar la tarea de forma más rápida, eficiente y acertada.

Existe un gran número de variantes para abordar el problema de clasificación automática. La clasificación basada en prototipos es un enfoque en el que un conjunto de datos se representa por un conjunto pequeño de instancias prototípicas. Nuevos

datos se clasifican considerando la distancia de dichos datos a los prototipos. Entre las ventajas de esta metodología se encuentra que una vez terminada la fase de entrenamiento, se requiere una menor cantidad de memoria, a la vez, al ser menor la cantidad de prototipos contra las instancias la velocidad de estos es más rápida que aquellos que usan el conjunto de instancias inicial. Bajo esta metodología cuando se tienen clases desbalanceadas la generación de prototipos de clasificación se puede usar para balancear los conjuntos, mas aun en conjuntos con pocos ejemplos permite aumentar el número de éstos de forma eficiente y sin gran costo. Otra ventaja es que estos prototipos se pueden usar para la interpretación de los datos. Por ejemplo, se puede decir que si dos clases son muy parecidas entre si o si se suele confundir una con otra (cuando los prototipos son muy parecidos), que tantas subclases puede contener una clase (analizando en número de prototipos generados por cada clase, entre más prototipos más subclases podrían estar presentes).

Por tal razón, en este trabajo abordamos el problema de generación de prototipos de clasificación mediante el uso de técnicas de PG.

De entre los métodos actuales para la generación de prototipos se distinguen dos desventajas principales.
1. El número de prototipos generados es un parámetro, esto puede traer dos problemas, el primero es que el número de prototipos generados sea muy bajo y no permita al clasificador tener un buen desempeño, mientras que usar un alto número de prototipos reduce las ventajas que tienen los prototipos de clasificación sobre otros métodos de clasificación.
2. Ante un bajo número de instancias se incurre en el sobreajuste.

La PG ha sido usada para generar clasificadores; las metodologías encontradas en la literatura usan PG para evolucionar árboles de decisión, reglas de clasificación y funciones discriminantes. La principal hipótesis de este trabajo es que la programación genética puede ser aplicada para generar prototipos. Lo anterior con base en resultados obtenidos por otros autores usando programación genética para clasifica-

ción bajo distintos esquemas (Espejo et al. 2010). Además la programación genética permite un modelado flexible de problemas complicados y permite el uso de estructuras de datos complejas como las que se requieren para la generación automática de prototipos.

1.4 Objetivos

Diseñar, desarrollar e implementar un programa genético para la generación automática de prototipos de clasificación que obtenga resultados comparables o superiores al estado del arte (en términos de reducción y clasificación).

Para ello se desglosan los siguientes objetivos particulares:

• Desarrollar una metodología de solución que utilice la PG para la generación automática de prototipos, definiendo la estructura de los individuos así como las herramientas que el programa genético tendrá a su disposición para evolucionar dichas estructuras.

• Implementar de forma computacional la metodología de PG para la generación automática de prototipos desarrollada en el punto anterior.

• Realizar experimentos computacionales para validar la metodología desarrollada.

1.5 Estructura de la tesis

Este trabajo se encuentra dividido en seis capítulos que describiremos brevemente a continuación.

El capítulo uno ofrece una descripción general del problema, así como la motivación, justificación y objetivos del trabajo presentado. En el capítulo dos se presenta el marco teórico. En el capítulo tres se presenta la revisión de la literatura relacionada con el problema. En el capítulo cuatro se describe detalladamente la metodología utilizada. En el capítulo cinco se describe la experimentación computacional realizada y los resultados obtenidos. Por último en el capítulo seis se presentan las

conclusiones, así como aportaciones y líneas de trabajo a futuro.

Capítulo 2

Antecedentes

2.1 Clasificación

El problema de clasificación supervisada es indudablemente uno de los tópicos de investigación más estudiados dentro de las áreas de aprendizaje automático y reconocimiento de patrones (Guyon *et al.*, 2010) (Bishop, 2006). Esto es debido a que una gran cantidad de problemas de la vida diaria pueden verse como problemas de clasificación; entre algunos de ellos se encuentran el decidir si un mensaje de correo electrónico es *spam* o no, conociendo las palabras contenidas en el mensaje (Carreras *et al.*, 2001); determinar el tipo de galaxia (de un conjunto de N tipos posibles) contenida en una imagen fotométrica (Calleja y Fuentes, 2004); o bien reconocer caracteres escritos a mano agrupados en palabras a partir de imágenes digitalizadas (Taskar *et al.*, 2003).

El objetivo en el problema de clasificación en aprendizaje computacional consiste en tomar un vector de entrada $\mathbf{x} \in \Re^d$ y asignarlo a una de N clases discretas c_n donde $n = 1 \ldots, N$ (Bishop, 2006).

A los elementos del vector $\mathbf{x}$ se les conoce como atributos, y éstos representan valores o características de un objeto o ente. Por ejemplo si tenemos un tornillo, quisiéramos poder determinar si el tornillo es defectuoso o no, esto es clasificar el tornillo con la clase defectuoso o con la clase funcional. Entre las características de interés para poder determinar esto, se encuentran la longitud del tornillo, su material, la porosidad de la cabeza, la cantidad de grados que esta desviado de la vertical, entre

otros, estos datos formarán el vector de entrada de nuestro problema de clasificación y les llamamos atributos. Es importante resaltar que dada la magnitud de tornillos que se fabrican cada día es imposible tratar de abordar este problema de forma manual; aquí es donde entra la clasificación automática, donde se busca lograr que un programa computacional realice la clasificación automáticamente.

Para abordar un problema de clasificación se debe construir un clasificador; esto es, una función que asocia elementos del conjunto de datos (vectores) con el conjunto de clases ($C = \{c_n, n = 1, \ldots, N\}$).

$$f : \mathbf{x} \in \Re^d \rightarrow c_n \in C$$

Dentro del aprendizaje supervisado se usa un conjunto de datos previamente clasificados correctamente (etiquetados), para inducir un modelo (i.e. aprender) capaz de clasificar instancias asociadas al problema de clasificación (Espejo *et al.*, 2010).

Formalmente el problema de clasificación de N clases consiste en encontrar una función *f*, partiendo de un conjunto (denominado de entrenamiento) de M pares ordenados $(\mathbf{x}_i, y_i)$ con $\mathbf{x}_i \in \Re^d$ y $y_i \in \{c_1, c_2, \ldots, c_N\}$, de forma que la función pueda usarse para hacer predicciones para datos (denominados de prueba) para los cuales no se conoce su clase, es decir se tiene el conjunto $\mathbf{x}^T_{1,\ldots,L}$ y se debe hallar $f(\mathbf{x}^T_j) = y^T_j, j = 1, \ldots, L$. Donde M es el número de instancias del conjunto de entrenamiento y L el número de instancias del conjunto de prueba.

Existen muchos métodos para aprender la función f, por ejemplo, en la figura 2.1 se muestra un clasificador lineal para 2 clases ($N = 2$), donde cada instancia $\mathbf{x}_i \in \Re^2$. En este ejemplo instancias por arriba de la línea ($z = m\mathbf{x} + b$) serán clasificados como positivos y negativos de otra forma, esto es, f es de la forma:

$$\mathrm{f}(\mathbf{x}) = \begin{cases} \textit{positivo} & \text{si } m\mathbf{x} + b > 0 \\ \textit{negativo} & \text{si } m\mathbf{x} + b \leq 0 \end{cases}$$

Además de los clasificadores lineales existen muchos enfoques para abordar el problema de clasificación (Hastie *et al.*, 2009); el enfoque que se aborda en esta tesis es el de generación de prototipos de clasificación presentado a continuación.

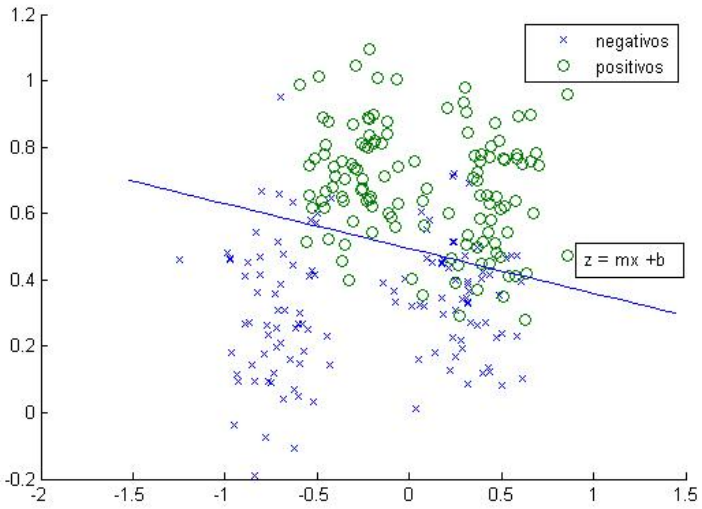


Figura 2.1: Ejemplo de un clasificador lineal. Observaciones que se encuentran del lado superior de la línea se clasificarán como positivos y negativos de otra forma

2.2 Métodos basados en prototipos

La idea de la clasificación basada en prototipos surge por los trabajos de Eleanor Rosch (Rosch, 1975) y otros, en donde se maneja la idea de que algunos miembros de una categoría son más centrales o prototípicos que otros. Por ejemplo un *canario* es más prototípico de la categoría *ave* que un *pingüino*.

En clasificación por prototipos, para un problema de N clases, se buscan por lo menos N prototipos, uno para cada una de las clases. Intuitivamente se quiere representar a todo un conjunto de datos por un subconjunto de observaciones que sean representativas del conjunto de datos completo. Si vemos a las instancias de entrenamiento como puntos en el espacio d-dimensional de atributos, el problema se reduce a hallar un subconjunto de esos puntos.

Cada prototipo está asociado a una clase, y a una nueva instancia se le clasifica con la clase del prototipo más cercano a éste. En la figura 2.2(a) se muestra un conjunto de datos de entrenamiento para un problema de 2 clases en un espacio de dimensión 2. La figura 2.2(b) muestra el uso de un prototipo por clase para clasificar una nueva observación $\mathbf{x}$ (no vista en los datos de entrenamiento).

Mediante la clasificación por prototipos el espacio de atributos es particionado en N regiones disjuntas, donde N es el número de prototipos, y cada región es asignada a cada prototipo. La frontera de estas regiones se determina como los puntos que son equidistantes a los 2 prototipos más cercanos a éstas. De esta manera la clase de un nuevo dato es asignada por el prototipo en cuya región del espacio el dato este posicionado.

Éstos métodos pueden ser muy efectivos si los prototipos están bien posicionados para capturar la distribución de cada clase (Hastie *et al.*, 2009). Además estos métodos requieren poco espacio en memoria y son extremadamente rápidos para clasificación. A continuación se describen los métodos basados en prototipos más comunes.

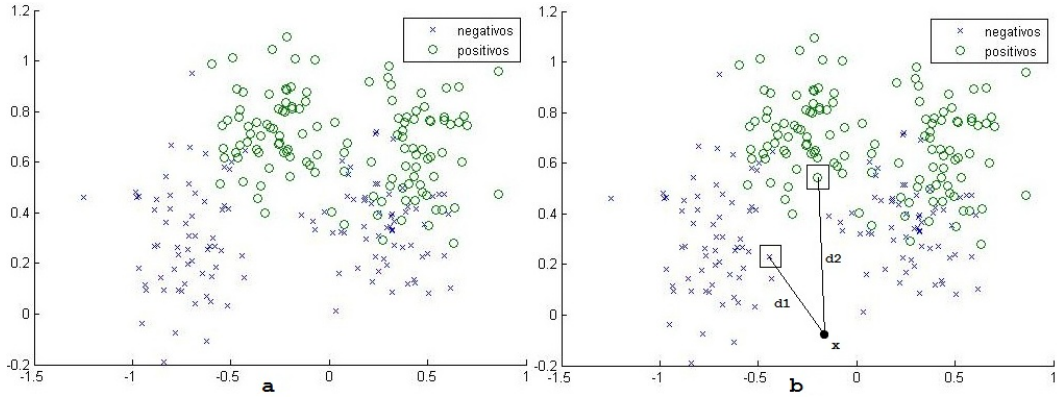


Figura 2.2: Figura (a) muestra un conjunto de datos de entrenamiento para un problema de clasificación de 2 clases. Figura (b) se elige un prototipo para cada clase y se calcula la distancia a una nueva observación $\mathbf{x}$, al ser $d_1 < d_2$ se asigna la clase negativa a la nueva observación

2.2.1 Algoritmo K-medias (*K-means*)

Este método está diseñado para encontrar agrupamientos de datos (*clustering*), así como los centros (prototipos) de los agrupamientos. Como parámetro se elige el número de centros que el algoritmo obtendrá (Lloyd, 1982). La entrada a *K-means* es un conjunto de datos y la salida es una asignación de cada observación a uno de K-grupos.

K-means se puede aplicar tanto a clasificación no supervisada como a clasificación supervisada. Para su aplicación en clasificación no supervisada, dado el conjunto

inicial de centros, el algoritmo busca para cada centro todas aquellas instancias que sean más cercanas a este centro y construye un nuevo centro usando la media de estas instancias. Este proceso se repite hasta satisfacer un criterio de paro. La idea de este algoritmo se basa en que instancias que sean parecidas entre sí (estén cerca unas de otras o agrupadas) tenderán a pertenecer al mismo grupo. En el algoritmo 1 se presenta el algoritmo de *K-means*.

Algorithm 1 *K-means*

Inicialización de centros

while Criterio de paro **do**

for $\forall centro_i$ **do**

\ * cen_i es el conjunto de instancias más cercanas a $centro_i$ que a cualquier otro centro. *\

$cen_i = \{\mathbf{x} | distancia(centro_i, \mathbf{x}) < distancia(centro_j, \mathbf{x}), \forall j\}$

* Se calcula el nuevo centro. *\

$centro_i = \sum_{\forall \mathbf{x} \in cen_i}(\mathbf{x})/|cen_i|$

end for

end while

El uso de *K-means* en aprendizaje supervisado consiste en separar las instancias de cada clase y posteriormente aplicar *K-means* a cada subconjunto de instancias para obtener R centros (prototipos) por cada clase. Una vez obtenidos éstos, se clasifica siguiendo algún esquema parecido al ilustrado en la figura 2.2(b). En el algoritmo 2 se presenta el algoritmo base de *K-means* para aprendizaje supervisado.

Algorithm 2 *K-means* para aprendizaje supervisado

for i=1 hasta N **do**

Usar *K-means* sobre todos los datos de entrenamiento con clase c_i usando R centros por clase.

* Éstos centros son los prototipos de la clase c_i. *\

Los centros obtenidos son etiquetados con la clase c_i.

end for

Entre los problemas de *K-means* para la construcción de prototipos se encuentra que no considera a las instancias de otras clases para construir los prototipos. Esto puede llevar a construir prototipos de distintas clases muy cercanos entre sí. También el parámetro R (número de centros) tiene que ser especificado, una mala elección de R puede llevar a malos resultados. Además, *K-means* no escala bien con el número de instancias y se vuelve prohibitivo cuando hay que analizar grandes bases de datos (Faber, 1994).

2.2.2 LVQ (*Learning Vector Quantization*)

Esta técnica fue propuesta por (Kohonen, 1989) y en ella los prototipos son desplazados a lugares estratégicos del espacio de atributos de manera que éstos tiendan a alejarse de las regiones de decisión y de otros prototipos de clases diferentes a la suya. Esto se logra seleccionando de forma aleatoria una instancia del conjunto de entrenamiento, se encuentra el prototipo más cercano a esta instancia y si la clase de ambos es la misma el prototipo se acerca a la instancia, si por el contrario pertenecen a clases diferentes, el prototipo se aleja de la observación. Por la forma en que LVQ actualiza sus prototipos (usando un dato a la vez) se dice que es un algoritmo en línea. En el algoritmo 3 se muestra la forma más elemental del algoritmo LVQ, el parámetro ϵ regula el tamaño de los desplazamientos en LVQ y es conocido como la taza de aprendizaje.

Una de las desventajas de los métodos basados en LVQ es que los prototipos obtenidos son generados por un algoritmo en lugar de obtenerse mediante la optimización de un criterio dado; esto hace difícil entender sus propiedades (Hastie *et al.*, 2009).

2.2.3 K-NN: K-vecinos más cercanos

Este método no genera ningún prototipo, sino que usa todos los datos en el conjunto de entrenamiento como prototipos. La diferencia radica en que no se usan

Algorithm 3 *LVQ*

Escoger R prototipos iníciales para cada clase $m_1(n), m_2(n), \ldots, m_R(n), n = 1, 2, \ldots, N$.

while Criterio de paro **do**

Seleccionar un dato de entrenamiento $\mathbf{x}_i$ usando muestreo con remplazo, sea $m_j(n)$ el prototipo más cercano a $\mathbf{x}_i$.

if $y_i = n$ **then**

$m_j(n) = m_j(n) + \epsilon(x_i - m_j(n))$

else

$m_j(n) = m_j(n) - \epsilon(x_i - m_j(n))$

end if

Reducir el valor de ϵ

end while

todos los prototipos al momento de clasificar, si no que solo se toman en cuenta los K más cercanos. Se realiza una votación donde se cuenta el número de veces que aparece cada clase entre los K prototipos más cercanos y a la instancia a clasificar se le asigna la clase mayoritaria. Existen variantes en las cuales la votación es ponderada respecto a la distancia; esto es, el voto de prototipos más cercanos es más influyente que el de prototipos más alejados. Este algoritmo es fácil de implementar, pero es costoso computacionalmente calcular las distancias, especialmente cuando el tamaño del conjunto de entrenamiento crece. Además que todos los datos deben de permanecer en la memoria.

2.3 Computación Evolutiva y Programación Genética

La computación evolutiva es un área de la ciencia computacional que se inspira de los procesos de evolución natural para resolver ciertos problemas (generalmente de optimización) (Eiben y Smith, 2008). Intuitivamente, se quieren desarrollar al-

goritmos de optimización que emulen la evolución natural. Haciendo una analogía entre las soluciones al problema de optimización y los individuos que evolucionan.
Si nos fijamos en una especie en particular, por ejemplo, *la mariposa negra de Manchester*, esta especie de mariposa vive en los bosques cercanos a Manchester, y el color predominante en las alas de sus individuos era originalmente de color blanquecino con motas negras, especial para confundirse durante el día en la corteza de los árboles. Con la llegada de la revolución industrial los bosques donde vive esta mariposa se vieron cubiertos de hollín negro, esto ocasionó que las mariposas de alas blanquecinas resaltaran al descansar en los árboles, haciéndolas más vulnerables ante depredadores. Esto a su vez dio como resultado que en 50 años el 95 % de las mariposas de esta especie tuvieran alas de coloración negruzca. Hoy en día se considera uno de los mejores ejemplos de cambio por selección natural. Antes del cambio de coloración de los bosques de Manchester la mariposa de alas blanquecinas era la que tenía más posibilidades de sobrevivir y multiplicarse. Esto es, era la más apta a su medio ambiente, cuando este se vio modificado, la mariposa de alas negruzcas se convirtió en la más apta. Luego con el paso de las generaciones, dado que esta mariposa podía sobrevivir y reproducirse, sus genes (que ocasionan coloración negruzca en las alas) se pasaban de generación en generación.
En computación evolutiva las soluciones a un problema de optimización toman el lugar de los individuos de la especie (i.e. mariposas). Mientras que la aptitud de las soluciones (individuos) se basa en un estimado de que tan buenas son dichas soluciones para nuestro problema. Así, el medio ambiente lo conforma el problema. Luego, en computación evolutiva tenemos un conjunto de soluciones, las cuales son evaluadas en el problema bajo estudio para obtener su aptitud, y esta determinará la posibilidad de conservarlas en nuestro conjunto de soluciones (i.e. sobrevivir) y de ser usadas como semillas para construir nuevas soluciones que tendrán características (rasgos) parecidas a los de sus padres (i.e. reproducirse). En la figura 2.3 se muestra un diagrama de flujo de un algoritmo de computación evolutiva estándar.
Existen diversas variantes de computación evolutiva, como algoritmos genéticos (Deb y Agrawal, 1998), estrategias evolutivas (Beyer, 2001) y programación genética (Ko-

za, 1992). En la siguiente sección describiremos este último enfoque debido a que es la técnica en que se basa este trabajo.

2.3.1 Programación Genética

La idea original de la programación genética fue concebida por John Koza y sus asociados (Koza, 1992). La programación genética (PG) es un campo de la computación evolutiva, por lo que se basa en la aptitud de las soluciones e involucra la transformación de conocimiento y la eliminación de soluciones no aptas mediante un proceso de selección que preserva soluciones apropiadas en las siguientes generaciones. La PG se aplica usualmente para encontrar soluciones que optimicen algún criterio, con la particularidad de que las soluciones tienen una estructura. Así, en PG se busca encontrar la estructura que optimice cierto criterio.

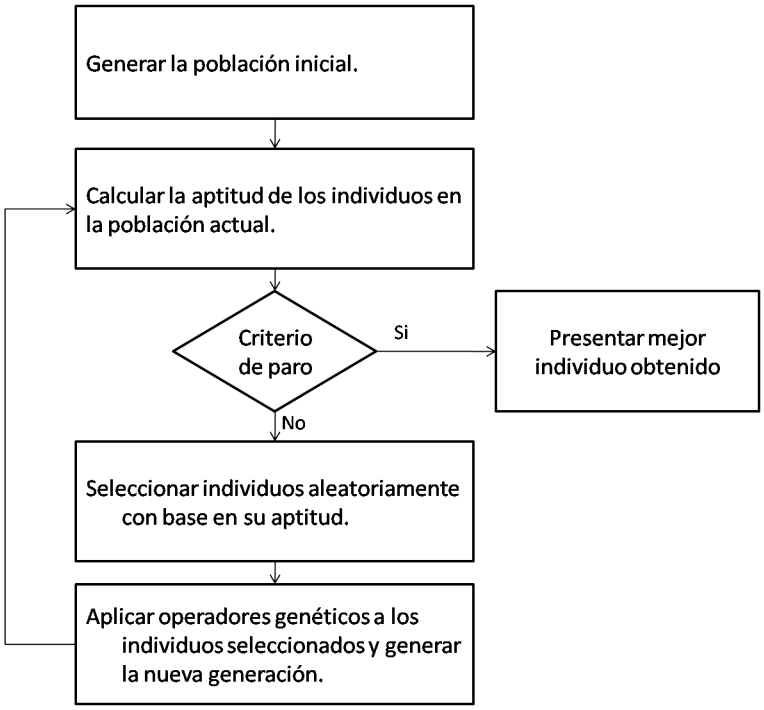


Figura 2.3: Diagrama de flujo de un algoritmo de computación evolutiva. La población inicial se genera aleatoriamente. La aptitud, de los individuos se determina por el problema bajo estudio. El criterio de paro más usado es ejecutar el algoritmo un número fijo de generaciones. En cada generación se actualiza la población y al final se presenta al mejor individuo. Véase (De Jong, 2008) para más detalles.

En PG se evolucionan programas, generalmente representados por árboles que son grafos acíclicos con una estructura como la presentada en la figura 2.4, esto es en PG nuestros individuos son árboles. En un árbol se tienen nodos, entre ellos se distinguen el nodo raíz desde el cual inicia el árbol, los nodos hoja o nodos de terminales y los nodos no-terminales. Un árbol siempre comienza con un nodo raíz y los nodos al final del árbol siempre son nodos terminales. En la figura 2.4 el nodo con la leyenda raíz es el nodo raíz, los nodos con los símbolos /, +, * representan los nodos no-terminales, los nodos con símbolos *y, 0, x, 2* representan los nodos hojas o terminales. Los hijos de un nodo son todos aquellos nodos que salen del nodo, así los hijos del nodo + son el nodo *y, 0.*

Los nodos no-terminales representan las operaciones que se realizarán con sus nodos hijos, y los nodos terminales representan las variables o constantes que se usarán en estas operaciones. Por ejemplo en la figura 2.4 la operación representada es $(y + 0)/(x * 2)$. Aquí el problema podría ser encontrar una ecuación algebraica (como la obtenida) que optimice cierto criterio.

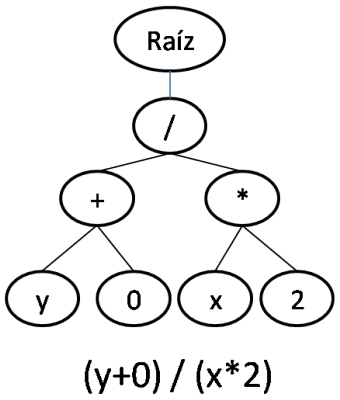


Figura 2.4: Ejemplo de un árbol y su representación algebraica

Para inicializar un árbol primero se necesitan definir dos conjuntos uno es el conjunto de terminales y el otro es el conjunto de operadores, el primero generalmente consiste en variables asociadas al problema bajo estudio, así como constantes (e.g. 0, 1 y -1) y números aleatorios. El conjunto de operadores está formado por todos aquellos operadores que pueden ser usados en los nodos de los árboles. Asociado

a cada operador se encuentra la aridad y esta representa el número de subárboles sobre el que se aplica dicho operador. Por ejemplo el operador *suma* requiere dos subárboles, mientras que el operador *coseno* requiere uno.

En PG los árboles generalmente son inicializados de forma aleatoria, entre los métodos más simples están el completo (*full*), el de crecimiento (*grow*) y el más popular es una mezcla de ambos conocido como *Ramped half-and-half* (Luke y Panait, 2001). Tanto en el método completo como en el de crecimiento los individuos son generados de tal manera que no excedan una profundidad máxima especificada por el usuario. La profundidad de un nodo es el número de aristas por las cuales se debe viajar para alcanzar éste nodo empezando desde el nodo raíz (el cual se considera tiene una profundidad de 0) (Poli *et al.*, 2008). La profundidad de un árbol es el máximo de entre todas las profundidades de los nodos que lo componen.

En el método completo de inicialización todos los árboles generados tienen la misma profundidad y el número de nodos dependerá de la aridad de los operadores usados. En la figura 2.5 se presenta un ejemplo paso a paso de el método de inicialización completo para una profundidad máxima de 2.

El método de crecimiento genera árboles de profundidad variable pues en cada paso del proceso se decide aleatoriamente si el nodo actual es un elemento del conjunto terminal o del conjunto de funciones. En la figura 2.6 se presenta paso a paso un ejemplo del método de inicialización crecimiento utilizando una profundidad máxima de 2.

Como ninguno de los dos métodos anteriores proveen con una variedad adecuada de tamaños o formas, Koza propuso una combinación llamada *ramped half-and-half* (Koza, 1992), este método construye la mitad de la población inicial con el método de crecimiento y la otra mitad con el método completo. Esto se hace usando una variedad de profundidades máximas para asegurar la generación de árboles variados tanto en tamaño como en forma.

En PG al igual que en cualquier algoritmo evolutivo se tienen operadores genéticos los cuales son los encargados de crear nuevos individuos que sean más aptos que sus antecesores, los 2 operadores básicos en PG son la mutación y el cruzamiento.

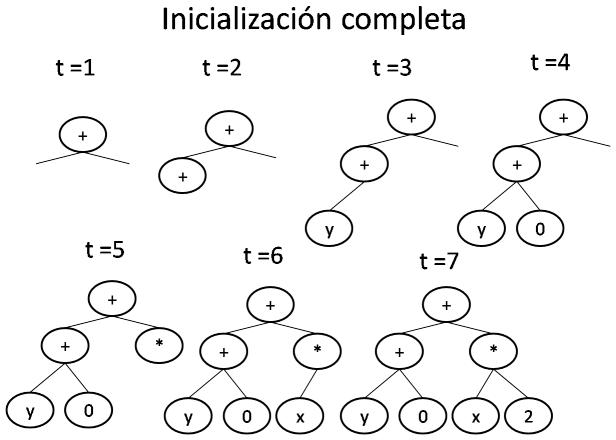


Figura 2.5: Inicialización de un árbol por medio del método completo con profundidad 2.

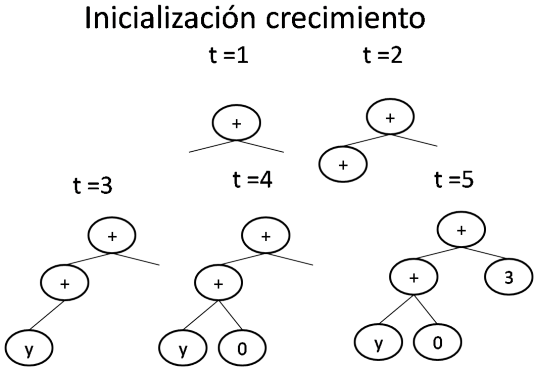


Figura 2.6: Inicialización de un árbol por medio del método de crecimiento con profundidad 2.

Las dos formas más comunes de mutación usadas en PG son:

•**La mutación subárbol**: en esta mutación se selecciona de forma aleatoria un nodo en un árbol y reemplaza todos los nodos que descienden de ese nodo por un subárbol generado de forma aleatoria.

•**La mutación por punto**: en este tipo de mutación un nodo es elegido de forma aleatoria, si el nodo es una función, ésta se reemplaza por otra función con la misma aridad que la reemplazada. Si es un nodo terminal, este se reemplaza por otro elemento del conjunto de terminales.

El otro operador genético es el de cruzamiento que fue propuesto por Koza (Koza, 1992). Dos individuos de la población son seleccionados y combinados para generar

un nuevo par de individuos. En este método se usan dos árboles, en la figura 2.7 se muestra el funcionamiento de este operador. Después de seleccionar los nodos y el subárbol que se desprende de ellos, éstos son intercambiados, así el primer individuo recibe el subárbol del segundo y respectivamente el segundo lo recibe del primero.

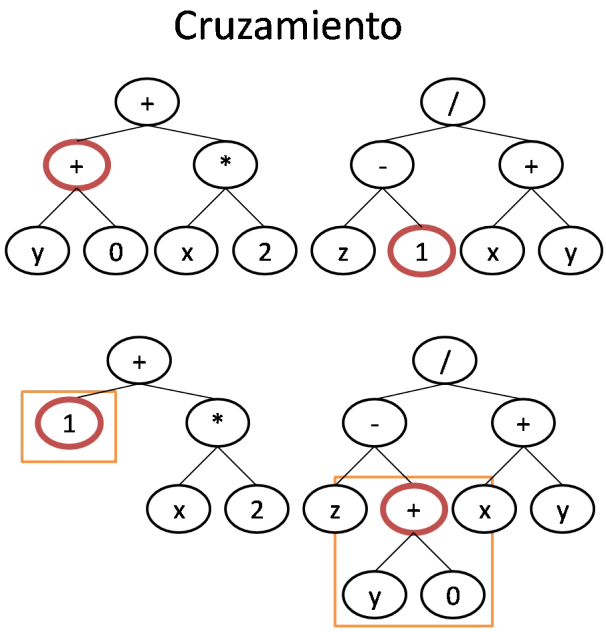


Figura 2.7: Método de cruzamiento propuesto por (Koza, 1992).

Entonces los operadores genéticos son los encargados de modificar los árboles lo que nos permite movernos en el espacio de soluciones. La encargada de guiar la búsqueda es la función de aptitud, ésta define que tan bueno es un individuo (i.e. un árbol) y por tanto la que guía que individuos serán usados en la creación de la nueva generación. En clasificación, la función de aptitud usualmente está basada en la exactitud, y generalmente se mide como el cociente del número de ejemplos clasificados correctamente y el número total de ejemplos (Espejo *et al.*, 2010).

En la PG se necesita definir un criterio de paro y designar la salida del algoritmo. Entre los criterios de paro más usuales se encuentra ejecutar el programa genético hasta un máximo número de generaciones así como alcanzar algún valor de aptitud. La salida del programa genético es el individuo que obtuvo el mejor valor de aptitud cuando se cumpla con el criterio de paro.

Los métodos de PG pueden considerarse como algoritmos de aprendizaje; para estos algoritmos se provee al programa con 2 conjuntos de datos conocidos como el conjunto de entrenamiento y el conjunto de prueba. El conjunto de entrenamiento contiene las instancias que el programa usará para aprender las reglas o relaciones que existen entre las variables de las instancias. Mientras que el conjunto de prueba se usa para determinar que tan bueno es el programa.
Generalmente, en un programa genético para clasificación, se inicializan los árboles y la aptitud de éstos es obtenida mediante los datos de entrenamiento. Se aplican los operadores genéticos de cruzamiento y mutación con base en la aptitud de los árboles y se sigue iterando hasta satisfacer el criterio de parada. Al final, se guarda el mejor árbol obtenido por el programa y éste se evalúa contra los datos de prueba. Este programa genético se puede describir con la figura 2.3 poniendo especial atención al inicializar nuestros individuos y en cómo los operadores genéticos se aplicarán. Profundizaremos en la siguiente sección un poco más sobre las diferentes representaciones que se les da a los árboles para atacar el problema de clasificación usando PG.

2.3.2 Representación de los árboles

Esta sección describe a los tres métodos de clasificación basados en PG más comunes. Como se podrá apreciar, la principal diferencia entre los enfoques es si los individuos conforman un árbol o un conjunto de árboles y la forma de representar estos árboles.

Árboles de decisión

Un árbol de decisión es un árbol en el cual cada nodo no-terminal representa una decisión y los arcos que salen de este nodo representan las posibles acciones a tomar; los nodos terminales representan la consecuencia de todas las decisiones que se tomaron para llegar a este. En el popular juego de mesa *Adivina Quien* en el cual dos jugadores compiten para tratar de adivinar el personaje del otro, se ilustra

el uso de árboles de decisión para clasificación; en este juego se toman turnos con un rival para preguntarse por una característica (atributo) de sus personajes (e.g. color de cabello, sexo, aretes, barba) dependiendo la respuesta que el rival dé a las preguntas de uno, la tarea es descartar a todos los personajes que no cumplan con la característica del cual el rival informó, y así con una serie de preguntas se trata de encontrar el personaje de el rival.

En PG para generar árboles de decisión, cada individuo es un árbol de decisión, los nodos no-terminales en un árbol de decisión representan un atributo de una instancia. Considérese el problema de clasificar a una persona como niño, adulto o adulto mayor, o clases A, B y C respectivamente. Para llevar a cabo esta tarea debemos obtener características de esta persona, entre ellas supóngase que tenemos su altura (cm), peso (kg) y número de horas de sueño promedio por día. Luego, los nodos no-terminales en nuestro árbol serán preguntas sobre estas características, por ejemplo ¿es la altura de la persona mayor de 50cm? ¿la persona duerme en promedio más de 12 horas por día? Viajando a través de los nodos, contestando estas preguntas llegaremos a un nodo terminal el cual nos indicara la clase de la instancia. En la imagen 2.8 se presenta un ejemplo de árbol de decisión para el problema antes mencionado junto con 4 instancias diferentes y la clase predicha por el árbol.

Altura	Peso	Horas sueño	Clase
92	40	10	A
60	20	12	A
175	70	7	B
166	85	4.5	C

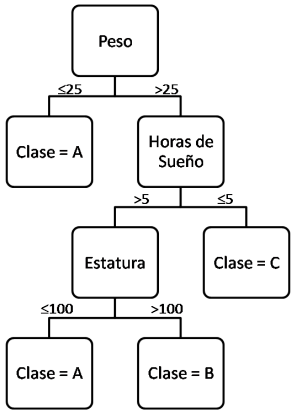


Figura 2.8: Un árbol de decisión para el problema de clasificación de 3 clases A, B y C. Se ejemplifican 4 instancias con la clase que le asigna dicho árbol.

En PG para generar árboles de decisión se diseña un programa genético que permita encontrar un árbol de decisión que clasifique correctamente las instancias provistas en un conjunto de datos de entrenamiento. Los árboles son generados aleatoriamente siguiendo las técnicas de inicialización completa, crecimiento, *ramped half-and-half* entre muchas otras, vea (Luke y Panait, 2001) para una lista más completa. Una vez inicializada la población se evalúa a cada individuo usando la exactitud en datos de entrenamiento como función objetivo. Bajo un esquema de torneo se seleccionan los individuos a ser usados en los operadores genéticos de cruzamiento y mutación subárbol, para crear la siguiente generación y así se itera hasta satisfacer un criterio de paro como el número de generaciones máximo, porcentaje de exactitud en datos de entrenamiento logrado y/o tiempo de cómputo máximo.

Reglas de clasificación

Las reglas de clasificación tienen dos partes, el antecedente y el consecuente. El antecedente es en estos casos el árbol; éste contiene una serie de reglas sobre los atributos de los datos y el consecuente es la clase predicha por el árbol. Los nodos no-terminales están formados por operadores lógicos como *AND, OR, NOT*, entre otros, los cuales conectan condiciones elementales de la forma *Atributo* $>$ *valor*, donde también se emplean $<$, $\leq$, $\geq$, entre otros. Este tipo de representación genera reglas de la forma:

$$\text{Clase} = 1 \begin{cases} Si & ((x_i > a_o) \cap \ldots \cap (x_j \leq a_p)) \\ O & ((x_i < a_q) \cap \ldots \cap (x_j > a_w)) \end{cases}$$

Donde x_i es un atributo de los datos y a_o es una constante.

En la figura 2.9 se muestra un árbol bajo este enfoque, su representación como reglas y la forma de clasificar algunas instancias de ejemplo.

Este enfoque se divide en dos categorías, aquellos en los que cada individuo representa una regla y aquellos en el que cada individuo consiste en un conjunto de reglas. Cuando cada individuo es una regla y se tiene un problema de clasificación

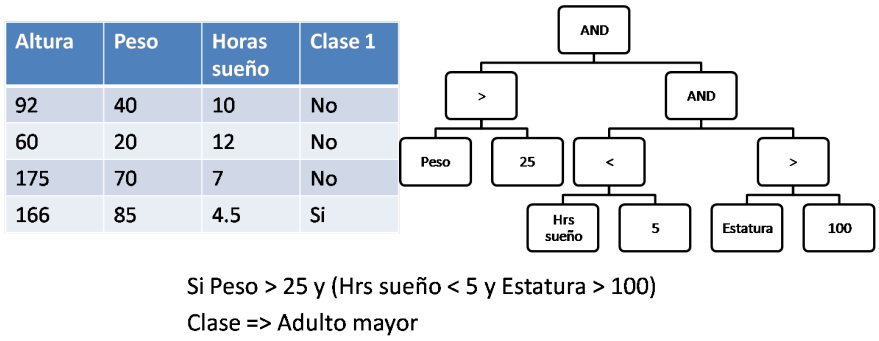


Figura 2.9: Un árbol como regla de clasificación, su representación en forma de regla y ejemplos de cómo clasifica a diferentes instancias.

binaria la salida del programa genético es la mejor solución durante las generaciones y ésta se usa para distinguir entre ambas clases; si cumple las reglas del árbol, se clasifica como de esta clase, si no las cumple se clasifica como la otra clase, véase por ejemplo (Eggermont *et al.*, 1999). Si se trata de un problema de N clases se evoluciona una regla por cada corrida del sistema, y se realizan N corridas (Bojarczuk *et al.*, 2000). El problema con este enfoque para N clases es que no se garantiza que cada instancia sea clasificada por una de las reglas, esto permite que existan instancias a las cuales se les asignen múltiples clases o instancias a las que no se les asigne clase al no pasar ninguna de las reglas. Cuando se trata el problema de N clases usando múltiples reglas por cada individuo se asigna una regla a cada clase y el clasificador es el mejor individuo obtenido.

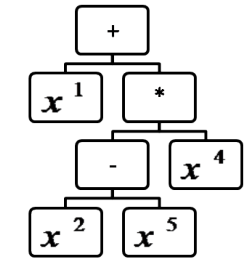
Funciones discriminantes

Bajo este esquema los nodos terminales son atributos del problema de clasificación y los no terminales son operadores.

En este enfoque cada individuo está conformado por un conjunto de árboles donde cada árbol tiene asociada una etiqueta de clase, una instancia se evalúa en

un árbol y este árbol regresa un número real el cual es usado para distinguir si el árbol clasificará la instancia como perteneciente a su clase, o no (cada árbol es una función), usualmente el valor usado para hacer esta clasificación es cero. En este enfoque se puede tener más de un árbol que reclama como de su clase a una instancia y para esto se deben crear reglas de desempate véase por ejemplo (Muni *et al.*, 2004). En la figura 2.10 se ilustra un árbol bajo este enfoque, el resultado dado por el árbol es analizado; si este es > 0, el árbol clasifica a la instancia evaluada como perteneciente a su clase; si es ≤ 0, no lo es. Para un problema de clasificación binaria no es necesario tener múltiples árboles por individuo, se puede usar un árbol el cual clasifica como perteneciente a su clase o positivo si la salida del árbol es positiva y de la otra clase o negativo si la salida es negativa o 0.

Atributos					Salida
1	2	3	4	5	-11
0	-3	3	-1	2	5
0	0	3	0	0	0



$$x^1 + ((x^2 - x^5) * x^4)$$

Figura 2.10: Un árbol con su representación algebraica y ejemplos de instancias y el resultado que regresa el árbol.

Para N clases se necesitan al menos N-1 árboles (N-1 árboles clasifican N-1 clases y la clase no clasificada por estos árboles decimos que pertenece a la clase N) pero generalmente se usan al menos N árboles; al evaluar un individuo una instancia, cada árbol de este individuo nos regresara un valor que luego traduciremos a positivo o negativo, o 1 y 0, esto es, para d árboles tendremos un vector $\mathbf{d}$, de 0's y 1's, de tamaño d.

Para este enfoque, al terminar el algoritmo y obtener el mejor individuo, se desea que al evaluar una instancia se cumpla $\sum_d(D(i)) = 1$; esto es, para cada instancia se desea que un árbol y solo uno clasifique a esta instancia, de no ser así se deben crear formas de desempatar si más de un árbol clasifica a una instancia, o de elegir

uno si ninguno árbol clasifica a una instancia. Es de notar que si se tienen clases con más de un árbol esta suma podría ser mayor, digamos todos los árboles de una clase clasifican a una instancia, sin incurrir en ninguna dificultad mientras solo árboles de una misma clase sean quienes clasifiquen a dicha instancia.

Capítulo 3

Trabajo Relacionado

Espejo *et al.* realiza un compendió de la aplicación de PG en el problema de clasificación (Espejo *et al.*, 2010). En esa referencia se puede ver que la mayor cantidad de trabajos publicados usan árboles que representan árboles de decisión, reglas de clasificación y funciones discriminantes. El resto de este capítulo revisa trabajos relacionados en el uso de PG para clasificación así como algunos métodos de generación de prototipos, ademas de clasificar a nuestro programa genético dentro de este tipo de método. Para finalizar se analizan otras áreas de interés para esta tesis.

3.1 Árboles de decisión

Una de las principales formas en que se usa PG para clasificación es para la generación de arboles de decisión (véase el capítulo 2); en esta sección se revisan aquellos trabajos relacionados con este enfoque.

Eggermont et al., proponen el uso de PG para generar árboles de decisión con representación atómica completa (*full atomic representation*) y se usa una función objetivo de múltiples capas para reducir el tamaño de los árboles (Eggermont *et al.*, 2004). Una representación atómica completa usa átomos en los nodos terminales y no terminales, donde un átomo es un predicado de la forma (*atributo-operador-valor*), los operadores usados aquí son funciones que regresen valores boleanos e.g. ($<$ o $=$). Los nodos terminales tienen asignaciones a clases ($clase = A$), en la figura 3.1 se

muestra un árbol generado por este enfoque.

Con base en el conjunto de datos de entrenamiento se crea una lista de los átomos a usar en el árbol, esto es, para cada atributo no numérico se crea un átomo de la forma ($atributo_i$ = valor) para cada valor diferente en los datos de entrenamiento. Para atributos numéricos se crean átomos de la forma ($atributo_i <$ valor) para cada valor diferente en los datos de entrenamiento. Para evitar la gran cantidad de átomos que este enfoque simple obtiene, se usa *K-means* sobre todos los atributos uno a la vez; los grupos encontrados son usados para reducir la cantidad de átomos. Si *K-means* determina 2 grupos este enfoque creará 2 átomos para este atributo de la forma (atributo $\in [min_1, max_1]$, atributo $\in [min_2, max_2]$ donde min_i, max_i son los valores máximos y mínimos para el grupo i respectivamente. Las medidas de evaluación usadas son la exactitud y el tamaño del árbol, si dos individuos tienen la misma exactitud, se usa su tamaño como criterio de desempate.

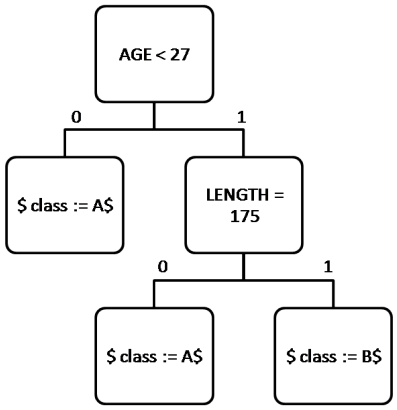


Figura 3.1: Árbol de decisión con representación atómica total (*full atomic tree*) (Eggermont *et al.*, 2004).

El uso de *K-means* permite que bases de datos con un gran número de instancias en general no tendrán gran cantidad de grupos, pues al usar *K-means* el enfoque se basa en la forma en que están distribuidos los datos y no su número lo que permite su aplicación a conjuntos con un gran número de instancias.

P. Turney utiliza árboles de decisión (Turney, 1995) para trabajar problemas de clasificación binarios en el área médica (clase: enfermo o sano) donde se implementa una función de costo que considera los costos de realizar tratamientos, así como el tiempo que tomó realizarlos. En este trabajo, Turney construye árboles en el que cada nodo no-terminal realiza un estudio sobre el paciente y se pregunta sobre el resultado del estudio. Esto podría ser el tomar una muestra de sangre del paciente y preguntar sobre el nivel de glucosa. La medida de desempeño considera los costos de realizar un diagnostico incorrecto, además de costos por cada tratamiento que se realiza; para esto, cada tratamiento tiene asociado un costo, así como el diagnosticar de forma errónea. Los resultados reportados muestran mejores resultados que los obtenidos con otros algoritmos de clasificación como *IDX* (Norton, 1989), *EG2* (Núñez, 1991), *CS-ID3* (Tan, 1993). La desventaja de este enfoque es que los costos de tratamientos y diagnósticos erróneos deben ser introducidos manualmente, y aunque es sencillo cuando se ponderan los costos de tratamientos, la relación de costo entre un diagnóstico erróneo y tratamientos no es algo sencillo y es además dependiente de la enfermedad con la que se trata.

Rouwhorst et. al. usa árboles de decisión (Rouwhorst *et al.*, 2000) para el problema de clasificación de N clases, donde cada individuo es un árbol y los nodos en el árbol pueden hacer comparaciones entre atributos. No existe una diferencia significativa al usar árboles de decisión para el problema de clasificación binario y el de N clases, pues la estructura es la misma y la clasificación de las N clases se realiza con un solo árbol. Solo se debe tener cuidado en generar árboles que puedan clasificar instancias de todas las clases. En este trabajo además de las comparaciones de atributos con constantes (e.g. $atributo_1 \leq 5$) también permite la comparación de atributos con atributos (e.g. $atributo_1 \leq atributo_2$). Por esta razón cae en la categoría de árboles de decisión oblicuos (Espejo *et al.*, 2010), pues haciendo un simple despeje este tipo de comparación es una combinación lineal de atributos en los nodos. Esto es equivalente a generar hiperplanos oblicuos a los ejes del espacio de atributos (Espejo *et al.*, 2010); lo que permite tener un clasificador con mayor flexibilidad. Se utilizan dos operadores de mutación, uno cambia el operador de relación ($=$, $<$, $\leq$,

etc.) y el otro cambia el lado derecho de una condición por otra escogida al azar. Se usa un operador de poda (*Prune*) el cual selecciona un nodo no-terminal, elimina todas las ramas que salen de este nodo y lo convierte en un nodo terminal. Antes de usar los operadores se analizan los árboles de la generación actual y se obtiene el tamaño y profundidad promedio de éstos, sí están por debajo de un parámetro especificado por el usuario, se agregan nuevas condiciones a los árboles. De esta forma se evita que los árboles vayan reduciendo su tamaño debido al operador poda.
Este enfoque obtiene resultados similares o mejores que algoritmos como *C4.5* (Quinlan, 1993) y *CN2* (Clark y Niblett, 1989), usando de forma consistente menores reglas. Una de las principales problemáticas de usar combinaciones lineales de atributos en los nodos es que puede llevar a un mayor sobreajuste que aquellos árboles de decisión que solo utilizan comparaciones de atributos con constantes. En este trabajo no se reporta un método para prevenir el sobreajuste, no se reporta la diferencia entre la exactitud en pruebas y entrenamiento; sería interesante conocer si el solo uso del operador poda, que evita el generar árboles de gran complejidad, ayuda a evitar el problema de sobreajuste.

En todos los trabajos previos se usan árboles de decisión, Jabeen y Baig nos dicen que es importante recalcar que la eficiencia de los árboles de decisión se ve perturbada si el conjunto de datos de entrenamiento es muy pequeño o muy grande, además los árboles de decisión se pueden volver muy grandes (Jabeen y Baig, 2010). Lo cual afectara el tiempo de ejecución de estos métodos.

3.2 Reglas de clasificación

Otro de los enfoques más comunes de PG para clasificación es la generación de reglas de clasificación (véase el capítulo 2), en esta sección se revisan aquellos trabajos relacionados con este enfoque.

El trabajo realizado en (Eggermont *et al.*, 1999) propone un clasificador lineal usando reglas de clasificación para el problema de clasificación binaria, donde cada

individuo es una regla. Al final de una corrida de un programa genético se obtiene la mejor regla de clasificación evolucionada, y esta conforma al clasificador, pues instancias que cumplan con la regla obtenida son clasificadas como positivas o deseadas, y aquellas que no cumplan con la regla son clasificadas como negativas o no deseadas. En este artículo se usa una función objetivo auto adaptativa. Cada instancia tiene un peso el cual es usado para evaluar la función objetivo, normalmente el acertar la clase de un instancia aumenta la función en una unidad. En este trabajo al acertar la clase de la instancia $\mathbf{x}_i$ la función aumenta en w_i que es el peso asociado a la i-esima instancia, y decimos que es adaptativa pues el programa genético tiene la capacidad de modificar éstos pesos; esto se logra incrementando el peso de las instancias que son más difíciles de clasificar. La idea general es que siendo una instancia más fácil de clasificar, al clasificar aquellas instancias difíciles, las fáciles serán clasificadas correctamente por el mismo clasificador. Esta función objetivo corre el riesgo de sesgarse debido a instancias mal clasificadas en el conjunto de entrenamiento, pues generalmente estas instancias son difíciles de clasificar, el clasificador que se evoluciona les daría más peso sacrificando el clasificar incorrectamente a otras instancias por darle prioridad a instancias mal clasificadas, mientras que una función usual como la exactitud, sacrificará esta instancia clasificada de forma incorrecta.

Un enfoque usando bosques es el propuesto por (Cordella *et al.*, 2006b), en este trabajo cada árbol representa una regla de clasificación, los árboles no tienen asignada una clase por defecto sino que esta asignación se deja al programa. Para cada instancia en el conjunto de entrenamiento se analiza si dicha instancia cumple con la regla de clasificación inducida por cada árbol de cada individuo. Sí la instancia solo cumple con una única regla, esa instancia se asigna al respectivo árbol. Sí cumple más de una regla y una tiene menos variables que las demás, se le asigna a ese árbol. Si por el contrario todas tienen el mismo número de variables la instancia es rechazada. Si no cumple ninguna regla también es rechazada. Después de realizar este proceso para todas las instancias en el conjunto de entrenamiento a cada árbol se le asigna la clase con mayor número de instancias asignadas a ese árbol. Si el árbol no tiene ningúna instancia asignada ese árbol es rechazado, para clasificar solo se consideran

los árboles con clase asignada. De esta manera la cantidad de árboles asignados a cada clase varía. El operador de cruza usado permite el intercambio de un número variable de árboles entre individuos, lo que a su vez le permite al programa aprender la cantidad de árboles apropiados para clasificar cada clase. En este trabajo no se crean árboles que tengan asignada una clase por omisión, esto puede llevar a tener individuos en los que no exista al menos un árbol para cada clase, y esto podría llevar la evolución de los árboles a clasificadores donde las clases faltantes siempre serán clasificadas de forma incorrecta, conforme el número de clases crezca este problema se hará más evidente.

El trabajo de Bojarczuk et. al. se basa en dividir el problema de clasificación de N clases en N problemas de clasificación binaria, y usa una función objetivo basada en la sensibilidad y especificidad (Bojarczuk *et al.*, 1999). En este trabajo se trató con un problema de 12 clases donde cada clase corresponde con una enfermedad diferente, en este trabajo se resolvían 12 problemas de clasificación binaria. Para crear la regla de clasificación de la clase 1 se corría un programa genético en donde la todas las instancias que no fueran de la clase 1 se combinan en una clase "negativa", el mejor individuo de esta corrida será la regla de clasificación de la clase 1, este proceso se repite para las 11 clases restantes obteniendo una regla para clasificar cada clase. La función objetivo es el producto de la sensibilidad Se y la especificidad Sp.

$$F(x) = Se * Sp$$

La sensibilidad y la especificidad se definen de la siguiente manera.

$$Se = tp/(tp + fn)$$

$$Sp = tn/(tn + fp)$$

Donde tp son los verdaderos positivos (*true positives*), fp falsos positivos (*false positives*), fn los falsos negativos (*false negatives*) y tn los verdaderos negativos (*true negatives*). Cuando se generan reglas de manera independiente no se puede asegurar que el conjunto de instancias clasificadas por cada clase sea mutuamente excluyente

(i.e. que una instancia pase más de una regla) o que alguna instancia no sea clasificada por ninguna regla. Para resolver este tipo de problemas debe crearse un algoritmo que desempate instancias que pasen múltiples reglas, y trate con instancias que no pasen ninguna, de otra forma todas estas instancias causaran conflictos y afectarán la efectividad del algoritmo.

3.3 Funciones discriminantes

El último de los tres enfoques más ampliamente usados en PG para el problema de clasificación es el de funciones discriminantes (véase capítulo 2), en esta sección se hace una revisión de la literatura con este enfoque.

Rauss et. al. enfocan el problema de clasificación binario usando funciones discriminantes en (Rauss *et al.*, 2000). En este trabajo se emplea un programa genético básico usando el software *lilgp* (Zongker y Punch, 1995), en el que cada individuo corresponde a una función discriminante, el umbral de clasificación es el cero, donde instancias evaluadas en la función discriminante que sean > 0 serán clasificadas como la clase positiva, y negativa de otro modo.

Silva y Tseng tratan el problema de clasificación de N clases como N problemas de clasificación binaria (Silva y Tseng, 2008). La base de datos usada consiste en gráficas de sonar tomadas del suelo marino. Se realizó un pre-proceso de los datos donde se obtuvieron 7 atributos para cada muestra. Se uso un programa genético base, el cual se corría una vez para cada clase diferente; la clase elegida se considera la clase positiva, mientras que todas las otras clases se juntan en una clase negativa. Como se trabajó con una base de datos desbalanceada (clases con considerablemente más o menos instancias que otras) la función objetivo usada fue la exactitud ponderada en relación al número de instancias de cada clase. Así, clasificar correctamente una instancia de una clase con pocas instancias da mayor beneficio que clasificar una instancia de una clase con mayor número de instancias. Para clasificar una instancia nueva ésta se evalúa de manera secuencial en todas las funciones discriminantes

creadas; si supera el umbral encontrado, es clasificada como la clase asociada a esa función, y sí no lo supera, se pasa a la siguiente función. En la figura 3.2 se muestra el clasificador y el proceso antes mencionado.

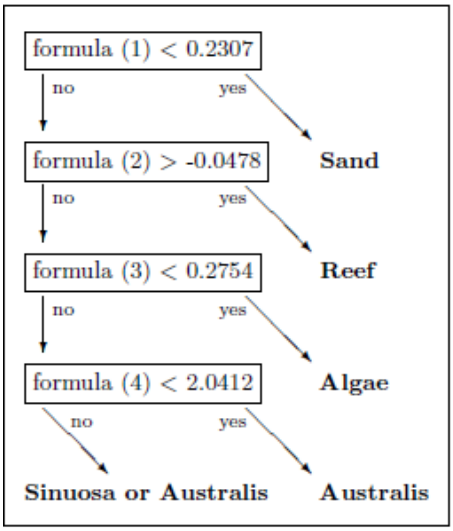


Figura 3.2: Clasificador propuesto para el problema de N clases, figura tomada de (Silva y Tseng, 2008)

Otro enfoque se basa en la creación de los N clasificadores con una sola corrida de un programa genético. En estos enfoques los individuos toman la forma de bosques (conjunto de árboles o multi-árboles) donde cada individuo tiene por lo menos N árboles y a cada árbol se le asigna la clasificación de una clase. A continuación se presentan trabajos que siguen este enfoque.

Muni et. al. en (Muni *et al.*, 2004) tratan el problema de N clases usando individuos con N árboles cada uno, cada árbol representa una función discriminante y éstos establecen si una instancia pertenece a su clase o no. Además, del concepto de aptitud usual para medir el desempeño de cada individuo, se introduce el concepto de incompetencia, el cual mide a nivel árbol, que tan apto es cada árbol; cada vez que un árbol predice que una instancia pertenece a su clase y no es de esta manera, se incrementa la incompetencia de ese árbol en una unidad. La incompetencia es usada para elegir los árboles sobre los que se aplicarán los operadores genéticos, de esta manera árboles con mayor incompetencia tienen más probabilidad de ser modificados.

Para resolver conflictos (i.e. una instancias es clasificada por más de un árbol) se usa la *extracción de reglas*. Esta heurística recuerda la clase de aquellas instancias en el conjunto de entrenamiento con problemas, además del vector v_i de tamaño N, en donde $v_{ij} = 1$ si el árbol de clase j clasifica a la instancia i y 0 de otro modo. Cuando se clasifican las instancias de prueba y se encuentra un conflicto, se busca si el vector de los árboles que la clasifican es igual a algún v_i, de ser así, se obtiene la clase mayoritaria en la que se da ese vector y esa es la clase asignada. Este trabajo propone una variación interesante al enfoque de funciones discriminantes usual, donde aquellas instancias que no son clasificadas por ninguna función o lo son por más de una se clasifican siguiendo un algoritmo y no son olvidadas o clasificadas arbitrariamente. La problemática de la heurística propuesta radica en el tamaño de la base de datos (número de instancias), si este es muy pequeño, entonces la cantidad de vectores v_i será menor y esto hará más difícil encontrar uno igual a alguna instancia con problemas.

Faraoun y Boukelif trabajan el problema de clasificación de N clases usando funciones discriminantes para mapear instancias al conjunto de números reales y usar un clasificador simple sobre estos números (Faraoun y Boukelif, 2000). En este trabajo cada individuo representa un conjunto de N funciones discriminantes no lineales, una por clase. Cada función discriminante regresa un valor real, el resultado es usado como un prototipo, y posteriormente se clasifica utilizando un enfoque *1-NN* (presentado en la sección 2.2.3 para K = 1) usando estos prototipos. Este trabajo puede verse como un clasificador *1-nn* en el que la distancia usada evoluciona, pues al usar funciones discriminantes que mapeen las instancias a $\Re$, este mapeo toma la función de definir la distancia. Este enfoque es altamente flexible al usar diferentes "distancias"para cada clase. Al usar como función objetivo la exactitud se puede caer en el sobreajuste, mientras que al usar operadores no lineales se pueden crear árboles cuya función discriminante será muy compleja, lo cual nuevamente puede caer en sobreajuste.

3.4 Generación de prototipos

Debido a que cada día el tamaño de los conjuntos reales es mayor, ha nacido la necesidad de crear métodos que puedan disminuir la cantidad de información que es necesaria procesar. La generación de prototipos busca que los prototipos generados aporten una buena representación de la distribución de las clases en el espacio de atributos. Además la cardinalidad de este conjunto, debe ser lo suficientemente pequeña para reducir la cantidad de memoria y el tiempo de evaluación que se utilice un clasificador basado en *K-NN* usando el conjunto de prototipos en lugar del conjunto de entrenamiento.

La generación de prototipos se define de manera formal en (Triguero *et al.*, 2012) de la siguiente manera:

Sea x_p una instancia, donde $\mathbf{x}_p = (x_{p1}, x_{p2}, ..., x_{pm}, x_{pw})$, con $\mathbf{x}_p$ perteneciente a la clase w, expresado como x_{pw}, y $\mathbf{x}_p$ en un espacio m-dimensional, donde x_{pi} es el valor del i-esimo atributo de la instancia p-esima. Dado un conjunto de entrenamiento con n instancias $\mathbf{X}_p$, y un conjunto de prueba compuesto de s instancias $\mathbf{X}_t$, con x_{tw} desconocido. El propósito de la generación de prototipos es obtener un conjunto de prototipos el cual consiste de r, r $<$ n, prototipos que son seleccionados o generados a partir de las instancias en el conjunto de entrenamiento. Tratando que el conjunto generado tenga niveles de clasificación cercanos al conjunto original, o en el mejor de los casos, que el nivel sea mayor.

3.4.1 Principales enfoques

En esta sección se describen algunos de los mejores métodos para generación de prototipos, otros métodos de generación de prototipos fueron comentados en la sección 2.2 y no son tratados nuevamente aquí.

Generalized editing using Nearest Neighbor (GENN)

Este método se basa en utilizar un enfoque basado en *K-NN* para obtener aquellas instancias que son clasificadas de forma incorrecta en la fase de entrenamiento. Si se encuentra una instancia con estas características se realiza un análisis que determina si dicha instancia sera eliminada del conjunto de entrenamiento, o si la clase de esta instancia sera cambiada por la que el método predijo en primera instancia. Este método se caracteriza por obtener buenos resultados para una gran variedad de conjuntos. Su principal problema se encuentra en el tamaño del conjunto de prototipos generados, pues la reducción obtenida por este metodo es muy pequeña. Para una lectura mas amplia del tema lea (Chang, 1974)

Particle Swarm Optimization (PSO)

Este es un método evolutivo, en donde se tiene una población (enjambre) de soluciones candidatas (llamadas partículas), estas partículas son movidas en el espacio de búsqueda siguiendo una formula matemática simple. El movimiento de cada partícula es influenciado por la posición de la mejor solución conocida por esta particula, ademas de la mejor solución conocida en todo el espacio de búsqueda.

Sea S el numero de partículas en el enjambre, con posición $\mathbf{x}_i \in \Re^n$. Sea p_i la mejor posición conocida de la partícula i y $\mathbf{g}$ la mejor solución de todo el enjambre. Un algoritmo base de PSO se describe en el algoritmo 4

El PSO al usarse en generación de prototipos es considerado un método de filtrado. Esto es cuando no se utiliza una regla de clasificación basado en *K-NN*, el uso de otras heuristicas como regla de clasificación les permite ser mas rápidos, pero la exactitud sobre el conjuro de prueba puede empeorar. Para una lectura mas profunda del tema se recomienda leer (Nanni y Lumini, 2009)

Algorithm 4 *PSO*

for i=1 hasta S **do**
  Inicializar la posición de las partículas de forma uniforme.
  Inicializar la mejor posición de la partícula a su posición inicial.
  si $f(p_i < f(g)$ actualizar la mejor posición del enjambre $\mathbf{g} = p_i$
  Inicializar la velocidad de la partícula de forma uniforme.
end for
while Criterio de paro **do**
  for i=1 hasta S **do**
    for d =1 hasta n (para cada dimensión) **do**
      Obtener números aleatorios uniformes en (0,1)
      Actualizar la velocidad de la partícula en esa dimensión en base a su mejor posición conocida y la de todo el enjambre
    end for
    Actualizar la posición de la partícula x_i dada su velocidad.
    if $f(\mathbf{x}_i) < f(p_i)$ Actualizar la mejor posición conocida de la partícula **then**
      si $f(p_i) < f(\mathbf{g})$ actualizar la mejor posición de todo el enjambre
    end if
  end for
end while
g representa la mejor posición obtenida por el algoritmo

Evolutionary Nearest Prototype Classifier (ENPC)

Este método es un algoritmo evolutivo que trata de mantener o incrementar la exactitud en el conjunto de prueba. En este método se empieza inicialmente con un individuo que representa a un prototipo (un punto en el espacio de soluciones), cada prototipo tiene la capacidad de reproducirse, creando otro prototipo parecido pero que logre clasificar aquellas instancias con las que el prototipo original tenía problemas. De esta manera solo se crean aquellos prototipos que sean necesarios para clasificar de mejor forma el conjunto. Se implementa ademas un operador de

mutación en el que se permite que un prototipo de clase i que clasifique mas instancias de clase j que de clase i pueda cambiar su clase de la i a la j; incrementando la exactitud en el conjunto de entrenamiento de forma directa. Por sus características se considera a este método como un método de ajuste de posición, en donde se busca corregir la posición de los prototipos utilizando un procedimiento de optimización, las nuevas posiciones se pueden obtener sumando o restando cantidades a los valores de los atributos de los prototipos. Vease (Fernández y Isasi, 2004) para mas información sobre este método.

3.4.2 Taxonomía del programa genético

En (Triguero *et al.*, 2012) se crea una taxonomía para clasificar a los método de generación de prototipos, dadas las características del programa genético desarrollado en esta tesis, este se cataloga como un *Mixed, Hybrid, Position adjustment, wrapper*.

La clasificación *Mixed* hace referencia a aquellos algoritmos que inician con un número preseleccionado de prototipos que pueden ser obtenidos de forma aleatoria; después se agregan, eliminan o modifican a los prototipos. Las ventajas de estos métodos es que tienen a su disposición todo el conjunto de entrenamiento para tomar decisiones sobre la modificación del conjunto de prototipos, lo que les permite realizar diferentes rectificaciones a lo largo de toda la ejecución del algoritmo.

Los métodos catalogados como *Hybrid* se caracterizan por intentar encontrar el conjunto de prototipos mas pequeño que mantenga o aumente la exactitud obtenida en el conjunto de prueba. Para permitir esto se permite la modificación de todos los prototipos, basado en algún criterio seguido por el algoritmo. El clasificador *K-NN* es altamente adaptable a este tipo de métodos, obteniendo grandes mejorías incluso en conjuntos con un número pequeño de prototipos.

Position adjustment hace referencia a aquellos métodos que buscan corregir la posición del conjunto de prototipos por medio de un procedimiento de optimizacion.

Las nuevas posiciones de los prototipos pueden obtenerse realizando sumas o restas de valores a los atributos de los prototipos.

Se considera al programa genético desarrollado en esta tesis como un método *wrapper* debido que se usa un clasificador *1-NN* para determinar la clase de cada instancia, usando como sus vecinos a los prototipos generados por el algoritmo.

En la figura 3.3 se presenta una imagen con 24 métodos de generación de prototipos catalogados por medio de sus características. Vemos que junto al programa genético desarrollado en esta tesis se encuentra algoritmos como ENPC y Adaptative Michigan PSO (AMPSO).

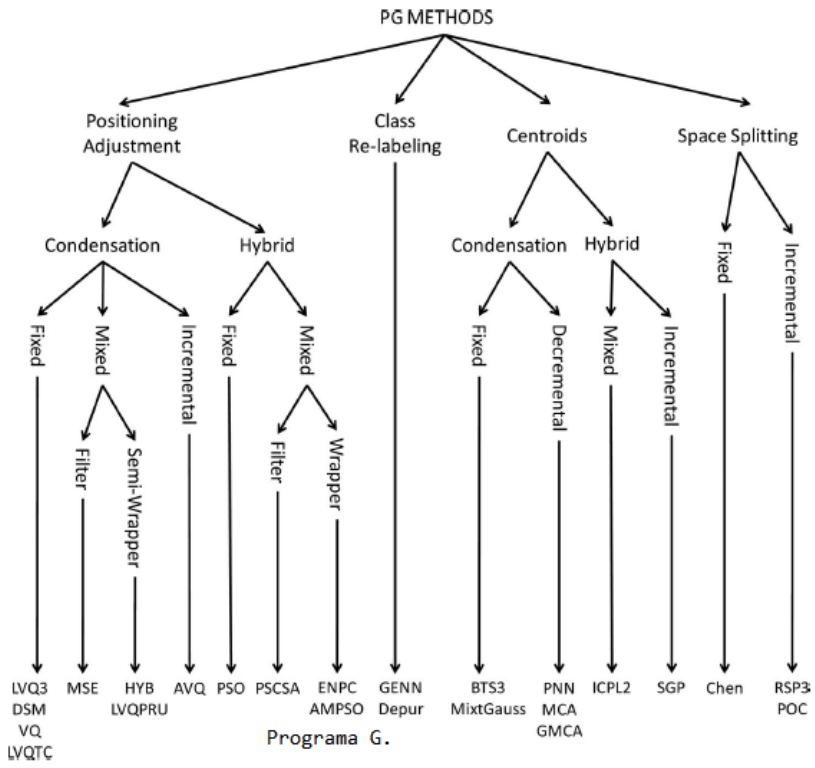


Figura 3.3: Taxonomía de clasificadores de prototipos

En la sección experimental se realizaron experimentos para comparar al programa genético desarrollado contra los otros 25 métodos (24 + *1-NN*) utilizando los resultados presentados en (Triguero *et al.*, 2012).

3.5 Otros enfoques

En esta sección se revisarán aquellos trabajos que por su relación al trabajo propuesto en esta tesis son de interés, aunque dichos trabajos no pertenecen a ninguna de las variantes descritas en secciones anteriores.

Cordella et al. proponen un clasificador basado en prototipos usando algoritmos genéticos (*Genetic Algorithms*) (Cordella *et al.*, 2006a). En este trabajo cada individuo es un conjunto de vectores con entradas reales, donde el tamaño de estos vectores es el mismo que el número de atributos de las instancias en el problema, esto es, cada vector es un punto en el espacio de atributos (un prototipo). Cada individuo forma, al aplicar *1-nn* usando los prototipos, un clasificador por su cuenta. Los prototipos son etiquetados como se describió anteriormente en la sección 3.2 (Cordella *et al.*, 2006b). En este trabajo se aplica una cruza ligeramente modificada, que es aplicar una cruza normal, pero los puntos de corte solo pueden ser antes o después de un prototipo (i.e. individuos intercambian prototipos enteros entre sí). Esta cruza permite tener una cantidad variable de prototipos para cada individuo, lo que le da la flexibilidad al PG de aprender la cantidad de prototipos necesaria para clasificar cada clase. Al igual que su análogo en reglas de clasificación en este trabajo no se crean prototipos que tengan asignada una clase por omisión, lo que puede traer como consecuencia el tener individuos en los que no exista por lo menos un prototipo para cada clase, lo cual podría llevar a la evolución de individuos donde estas clases faltantes no tengan prototipos, lo que a su vez hará que instancias de estas clases siempre serán clasificadas de forma incorrecta.

Tan et al. retoma la idea de un clasificador basado en el centroide (Tan *et al.*, 2011). Para la clase i se calcula el centroide de las instancias en el conjunto de entrenamiento con clase i, este paso se realiza para cada clase. Estos centroides son usados como prototipos iniciales en un enfoque *LVQ*, donde para mover éstos a posiciones más convenientes, se toman en cuenta el *margen* (Crammer *et al.*, 2002) y la exactitud. El *margen* es una medida a nivel instancia, donde para cada instan-

cia se calcula la distancia del prototipo de su clase más cercano y la distancia del prototipo más cercano de clase diferente, el margen es la resta de estas dos medidas (distancia diferente clase - distancia misma clase). Así si el *margen* > 0 equivale a que el prototipo más cercano a esta instancia es de su misma clase, y por ende se clasificará correctamente y si es ≤ 0 implica que un prototipo de clase diferente es el más cercano, lo que equivale a una clasificación incorrecta, en la figura 3.4 se ilustra el uso de esta medida para tres instancias diferentes, es de notar que aquí se uso la distancia euclidiana.

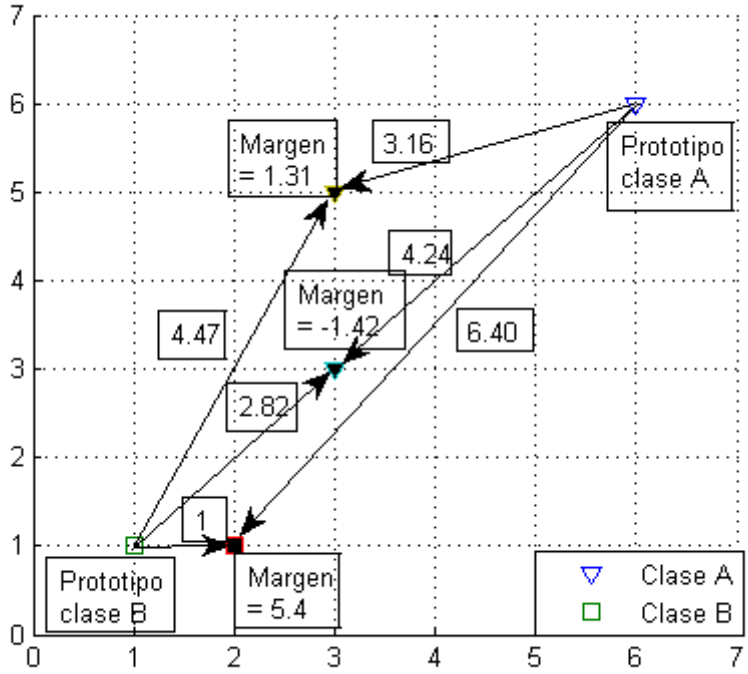


Figura 3.4: Se presenta el funcionamiento de la medida *margen*, en la imagen hay dos prototipos y tres instancias, para calcular el *margen* se calculan las distancias (en este caso euclidianas como ilustración) de ambos prototipos (el más cercano de otra clase y el más cercano de su clase) y estas distancias se restan, distancia diferente clase - distancia misma clase. El *margen* es el resultado de esta diferencia. Para la instancia en (3,5) el margen es > 0 lo que implicaría una clasificación correcta, la instancia localizada en (3,3) tiene un margen negativo pues está más cercana a un prototipo de clase diferente, esto correspondería a clasificarlo de forma incorrecta.

En esta propuesta se usa la distancia coseno como medida de similitud, lo que lleva a que el *margen* $\in$ (-1,1). Para mover los prototipos de lugar se utiliza una

combinación del *margen* y la exactitud, de esta manera se usan ambas medidas. Para el *margen* solo se utilizan aquellas instancias que estén en el rango $(0,\theta)$, donde $\theta < 1$, pues se desea concentrar la búsqueda a aquellas instancias con *margen* pequeño. Nuestro principal interés en este trabajo es el uso de la medida *margen* para evitar el sobreajuste.

3.6 Medidas de Desempeño

La función de aptitud más usada en clasificación es la exactitud (*accuracy*), esto es, la proporción de instancias clasificadas de forma correcta contra el total de instancias (Jabeen y Baig, 2010). El problema con este tipo de medida de evaluación es que no mide la relación entre el número de instancias por cada clase. Supóngase que se trabajó con el problema de clasificación de pacientes para detectar alguna enfermedad rara, en donde es común tener un gran número de instancias con clase *sana* y pocas de clase *enfermo*, digamos que se tienen 99 % de las instancias como *sanos*. Un clasificador que siempre diga *sano* tendrá una exactitud del 99 % aunque no logre distinguir en realidad una clase de la otra. En este tipo de casos es más útil sacrificar exactitud para lograr una mejor clasificación de la clase con menos datos. Entre las medidas de desempeño que buscan evitar este tipo de problemas se encuentra la cobertura (*recall*) la cual es una medida a nivel clase, esto es, para cada clase se obtiene una valor de *recall*, y se calcula de la siguiente manera:

$$Recall(c_n) = \frac{número\ de\ instancias\ de\ clase\ c_n\ clasificadas\ correctamente}{número\ de\ instancias:\ \ de\ clase\ c_n}$$

La Precisión (*precision*) también una función a nivel clase la cual se calcula como:

$$Precision(c_n) = \frac{número\ de\ instancias\ de\ clase\ c_n\ clasificadas\ correctamente}{número\ de\ instancias\ predichas\ como\ c_n}$$

Y la Medida-F (*F-Measure*) combina *recall* y *precision*

$$F - Measure = \frac{1}{N} \sum_{n \in C} \left[\frac{2 \times Recall(c_n) \times Precision(c_n)}{Recall(c_n) + Precision(c_n)} \right]$$

Recall es la proporción de instancias de la clase n clasificados correctamente contra total de instancias de clase n, *Precision* es la proporción de instancias clasificadas como clase n correctamente contra el total de instancias clasificadas como clase n. La Medida-F es una combinación entre el *Recall* y *Precision* la cual es sumamente útil pues permite tener una sola medida de evaluación que a la vez conserva propiedades de estas medidas, para una lista más elaborada de posibles funciones véase (Borgelt, 2005).

Capítulo 4

Programación Genética de prototipos

En este capitulo se describe el algoritmo desarrollado en esta tesis mencionando las características que lo componen.

Como ya se menciono previamente el programa genético desarrollado en esta tesis es un método para generar prototipos de clasificación, buscando obtener buenos valores tanto de clasificación como de reducción de la cardinalidad comparando el conjunto de prototipos con el conjunto de instancias de entrenamiento. Se busca que al usar la metodología de la PG, que nos permite un modelado flexible, se logre encontrar un punto intermedio en el cual se pueda maximizar la exactitud sobre el conjunto de prueba a la vez que se reduce el tamaño del conjunto de prototipos generado. Para realizar esto se emplea una función de aptitud basada en maximizar el indice de clasificaciónes correctas, mientras que la reducción del número de prototipos a ser usados se deja a libertad del programa genético desarrollado.

En el resto del capítulo se explica el enfoque propuesto para abordar el problema de clasificación basada en prototipos usando PG. Además, se describe el operador para PG propuesto en esta tesis.

4.1 Programación Genética de prototipos para clasificación

El enfoque propuesto en esta tesis combina la clasificación basada en prototipos con la PG. En el enfoque propuesto cada individuo del programa genético consta de un conjunto de árboles y cada árbol define un prototipo. Como se vio en la sección 2.2, un prototipo es un punto en el espacio de atributos de los datos del problema de clasificación. Para el problema de clasificación de N clases generamos p árboles por individuo donde $p \geq N$. Así cada individuo consiste de p puntos en el espacio de atributos, donde cada punto tiene una etiqueta de clase, luego podemos clasificar nuevas instancias asignándoselas a la clase cuyo punto sea el más cercano (como se ilustró en la figura 2.2(b)).
La clase del árbol s_i la notamos como s_i^c y al prototipo inducido por este lo notaremos como p_{s_i}, la clase de la instancia $\mathbf{x}_k$ la escribimos como $\mathbf{y}_k$.

Decimos que un árbol s_i clasifica una instancia $\mathbf{x}$ si el prototipo inducido por este árbol es el más cercano a dicha instancia y lo escribimos de la siguiente manera:

$$\rho(s_i, \mathbf{x}) = 1 \Leftrightarrow |\mathbf{x} - p_{s_i}| < |\mathbf{x} - p_{s_j}| \, \forall j \neq i$$

Cabe notar que dada la construcción de este clasificador $\sum_{\forall i} \rho(s_i, \mathbf{x}) = 1; \forall \mathbf{x} \in X$, i.e., para todos los árboles en un individuo solo un árbol clasifica a una instancia (solo un prototipo es el más cercano) y cada instancia es clasificada por un único árbol (siempre existe un prototipo que será el más cercano). Esto es importante, pues cualquier instancia que se le presente al clasificador, será clasificada siguiendo el mismo procedimiento que todas las demás; es decir, no tendremos instancias problemáticas que no puedan ser clasificadas de la misma forma que las demás.

La forma de clasificar en este trabajo se basa en la idea del prototipo más cercano, luego la forma de definir distancia cobra especial interés en este trabajo.

Esto implica que la noción de cercanía es más fundamental que la noción de clase (Duin y Paclík, 2006). En este trabajo se usa la distancia del *coseno* como función de similitud, definida como:

$$cos(\mathbf{d}_i, \mathbf{d}_j) = \frac{\mathbf{d}_i \cdot \mathbf{d}_j}{\|\mathbf{d}_i\|_2 \ * \ \|\mathbf{d}_j\|_2}$$

Donde $\mathbf{d}_i$ es una instancia de nuestro conjunto y d_j es un prototipo. La distancia del coseno entre dos vectores mide la similaridad de éstos, midiendo el coseno del ángulo que se forma entre ellos, el coseno de un ángulo de dos vectores determina si dichos vectores apuntan en aproximadamente la misma dirección.

En este trabajo el conjunto de terminales varía dependiendo la clase del árbol en cuestión y se define de la siguiente manera. Para un árbol de clase c_n el conjunto de terminales T_n es la unión del conjunto $Q_n = \{\mathbf{x}_k \in X | \, y_k = c_n\}$, además de la media, varianza, desviación estándar, mediana y la suma de los elementos del conjunto Q_n. Q_n es el conjunto de todas las instancias de clase n, y la idea de usar este conjunto para cada clase es que, al inicializar los árboles con instancias pertenecientes a su misma clase, se tendrá un mejor punto de inicio para la búsqueda de mejores prototipos. La media también es agregada bajo estos preceptos por su gran capacidad de clasificación (Han y Karypis, 2000). La varianza, desviación estándar, mediana y la suma, son agregados por su utilidad para describir una clase. Además, éstas medidas ayudan al programa genético al poder incorporarlas de forma directa a su estructura sin la necesidad de tener que evolucionar ramas de árboles que realicen esta misma función; de esta manera se incrementa la velocidad de evolución hacia estas medidas si son de utilidad, además de ayudar a disminuir el tamaño de los árboles.

En la figura 4.1 se ilustra la estructura de los árboles en el programa genético propuesto, su representación algebraica así como el prototipo que inducen.

Definimos el conjunto de operadores como F = {+, -, *, /, coseno, seno, tangente, $power^2$, $power^3$, minT, abs}. En la tabla 4.1 se muestra la aridad de cada operador y la operación que realiza. Este conjunto de operadores es el usual (Faraoun y Boukelif, 2000) (Iba *et al.*, 2009).

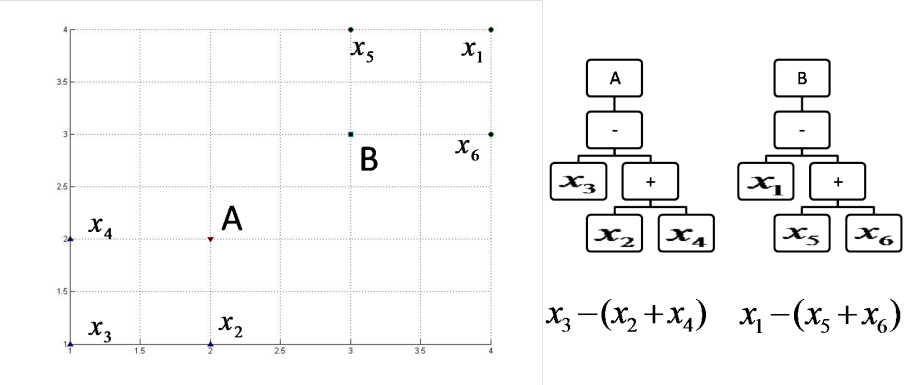


Figura 4.1: Dos árboles (prototipos) bajo el PG propuesto, su estructura (derecha), representación algebraica (abajo derecha) y los prototipos (A, B) inducidos por los árboles.

Nuestro conjunto de terminales son instancias y los operadores usados cumplen que al operar sobre instancias regresan instancias, a esta propiedad se le conoce como cerradura (Poli *et al.*, 2008); esto implica que realizamos operaciones cerradas sobre puntos en el espacio de atributos, por consiguiente, los prototipos inducidos por los árboles en nuestra propuesta son puntos en el espacio de atributos.

Operador	**Aridad**	**Función**
+	2 instancias	Suma atributo a atributo
-	2 instancias	Resta atributo a atributo
*	2 instancias	Multiplicación atributo a atributo
/	2 instancias	División atributo a atributo
Coseno	1 instancia	Coseno atributo a atributo
Seno	1 instancia	Seno atributo a atributo
Tangente	1 instancia	Tangente atributo a atributo
$Power^2$	1 instancia	Segunda potencia atributo a atributo
$Power^3$	1 instancia	Tercera potencia atributo a atributo
minT	2 instancias	Menor valor atributo a atributo
Abs	1 instancia	Valor absoluto atributo a atributo

Tabla 4.1: Operadores considerados en el PG propuesto

Como se mencionó anteriormente cada individuo consiste de p árboles ($p \geq N$),

durante la inicialización de los individuos se crean $\lfloor p/N \rfloor$ árboles por cada clase, a los restantes $p\,\%N$ árboles a ser creados se les asigna una clase de forma aleatoria. Esto garantiza que cada clase tenga por lo menos un árbol asignado y al aumentar el valor de p el número de árboles por clase se mantenga equilibrado. Una vez que se decide cuantos árboles tendrá cada clase, los árboles son inicializados usando el método *ramped half-and-half*, véase la sección 2.3.1.

Después de inicializar todos los individuos se procede a evaluar la aptitud de los individuos y la incompetencia de los árboles. La aptitud de los individuos es una combinación entre la exactitud y el margen. La combinación de ambas medidas permite utilizar lo que deseamos maximizar (el número de instancias clasificadas de forma correcta) que es medido de forma proporcional por la exactitud, sin caer en los problemas mencionados en la sección 3.6 gracias al uso del margen. La idea del margen fue usada por Tan, Wang y Wu en (Tan *et al.*, 2011). Esta medida de desempeño guía la búsqueda tratando de evolucionar individuos capaces de clasificar de forma correcta las instancias en el conjunto de entrenamiento y al mismo tiempo obtener individuos cuyos prototipos de clases diferentes estén posicionados de tal manera que la frontera de la teselación de Voronoi (Aurenhammer, 1991) inducida por éstos se encuentre lo más alejada posible de las instancias cercanas a la frontera. En la figura 4.2(a) se presenta un problema de clasificación de una dimensión, se presentan los prototipos A y B, así como la distribución de las instancias de entrenamiento de la clase A y B (en azul), y la frontera f que definen los prototipos A y B. Cualquier instancia que caiga a la derecha de f será clasificado como clase B y de clase A si ocurre lo contrario. Para esta imagen vemos que éstos prototipos clasifican de manera correcta todas las instancias en el conjunto de entrenamiento. En la figura 4.2(b) se presentan los mismos prototipos, pero ahora se muestra la distribución de las instancias de prueba de la clase A (en gris). Vemos que ahora la elección de los prototipos A y B no parece ser buena pues incurrimos en muchos errores para clasificar las instancias de prueba de la clase A. Esta elección de los prototipos A y B hacen que el margen de las instancias de entrenamiento de A cercanas a f sea muy pequeño. En la figura 4.2(c) se usan los prototipos A' y B y la frontera f', dada

esta elección de prototipos el margen para las instancias de entrenamiento de A se ve incrementado. Esta idea es usada ampliamente en maquinas de soporte vectorial, para obtener mayor información sobre este algoritmo véase (Hastie *et al.*, 2009); y véase (Crammer *et al.*, 2002) para una discusión más detallada de la medida margen en métodos de clasificación basados en distancia.

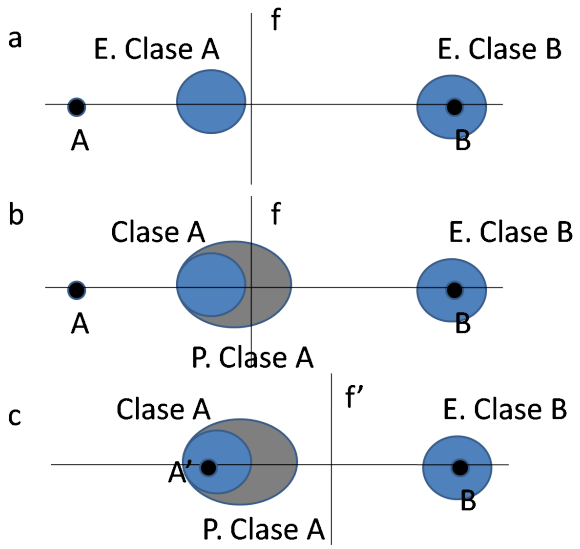


Figura 4.2: En la figura (a) se muestran los prototipos A y B, la frontera f y la distribución de las instancias de entrenamiento de cada clase. La figura (b) muestra la distribución de las instancias de prueba de la clase A. En la figura (c) se ilustra cómo se mueve la frontera al cambiar el prototipo A por A', y como esto incrementa el margen de las instancias de A, así como clasifica mejor las instancias del conjunto de prueba.

El margen se calcula de la siguiente manera:

$$Margen = 1/M * \sum_{\forall m} (cos(p_i, x_m) - cos(p_j, x_m))$$

Donde M es el número de instancias en el conjunto de entrenamiento, p_i es el prototipo más cercano a x_m que no tiene la misma clase que x_m, p_j es el prototipo más cercano a x_m de su misma clase.

La función de aptitud usada en este trabajo:

$$Aptitud = Exactitud + Margen$$

Notese que esta función no toma en cuenta el número de prototipos que tiene cada individuo, luego bajo este precepto un individuo con 100 prototipos es bajo esta función igual que un individuo con 10 prototipos siempre y cuando ambos tenga el mismo valor de aptitud. Dejando el trabajo de reducir el número de prototipos a los operadores genéticos implementados.

En computación evolutiva se busca evolucionar a nuestros individuos hasta obtener el mejor individuo; esta evolución de los individuos se logra modificando parte del individuo y tratando de encontrar mejores individuos con cada modificación. El problema es que se desconoce que parte del individuo conviene modificar, ya que el cambio que se realiza puede borrar genes del individuo que sean los responsables del buen comportamiento de ese individuo. Si se supiera que partes del individuo son responsables de esto, se podría instruir a nuestro algoritmo que modifique estas partes en menor proporción, evitando así destruir genes importantes de nuestros individuos. En este trabajo nuestros individuos se conforman de un conjunto de árboles, esto nos da la oportunidad de obtener una medida de desempeño para los árboles, lo que nos da una forma de conocer qué partes de nuestros individuos son las encargadas de su buen comportamiento, y aquellas que no lo son. Esta medida de desempeño, llamada incompetencia, es una medida a nivel árbol, y mide la cantidad de clasificaciones incorrectas que realiza cada árbol, i.e., la cantidad de veces que un árbol clasifica una instancia y esta no es de la misma clase que el árbol. Con la ayuda de esta medida podemos enfocar el uso de nuestros operadores para modificar aquellos árboles dentro de nuestros individuos que son responsables por clasificar de manera incorrecta a las instancias en el conjunto de entrenamiento.

Una vez evaluados todos los individuos se ordenan de acuerdo a su aptitud y los mejores se consideran los individuos élite, este es el conjunto de individuos que pasaran intactos (sobrevivirán) a la siguiente generación. Para los individuos a crear (nacer) restantes, se genera un número aleatorio $\alpha \in (0, 1)$ y se aplica uno

de 4 operadores genéticos, mutación por punto (*point mutation*), cruza, gran cruza (*large crossover*) y mitosis. Cada uno de éstos operadores se describe a continuación.

• **Gran Cruza**: Realiza una cruza entre 2 individuos seleccionados usando un proceso de torneo. Dados los 2 individuos se elige una clase aleatoriamente, una vez obtenida ésta, se selecciona uno de los árboles para cada individuo que tenga como etiqueta la clase seleccionada, todos los árboles a la derecha del corte son intercambiados con el otro individuo, en la figura 4.3 se muestra un ejemplo.

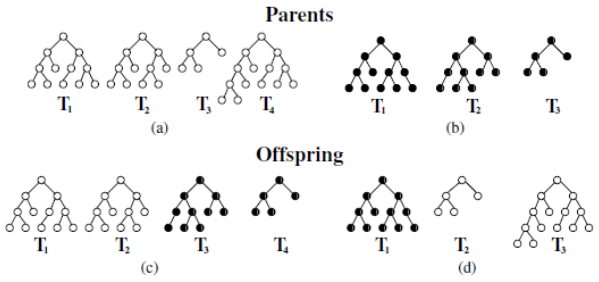


Figura 4.3: Un ejemplo de la aplicación del operador gran cruza. Las figuras (a) y (b) muestran dos individuos a los que se les aplicara el operador. Las figuras (c) y (d) muestras los individuos resultantes de la operación, en este caso se cortó a partir de T_{3a} y T_{2b}. Figura tomada de (Cordella *et al.*, 2006b).

• **Cruza**: Este tipo de cruza permite el intercambio de múltiples árboles con la misma clase entre individuos. Se seleccionan 2 individuos usando un proceso de torneo y una clase de forma aleatoria, para ambos individuos se escoge al menos un árbol de esta clase y se intercambian entre si. En la figura 4.4 se muestra un ejemplo de este operador.

• **Mutación por punto**: Se selecciona de forma aleatoria un individuo, el árbol a mutar se selecciona usando la incompetencia de los árboles. Seleccionado el árbol a mutar, se selecciona al azar un nodo, si es un operador se cambia por otro operador de la misma aridad, si es un nodo terminal se cambia por otro elemento del conjunto de terminales. Este paso se realiza H veces, donde H es un porcentaje del número de nodos del árbol a mutar.

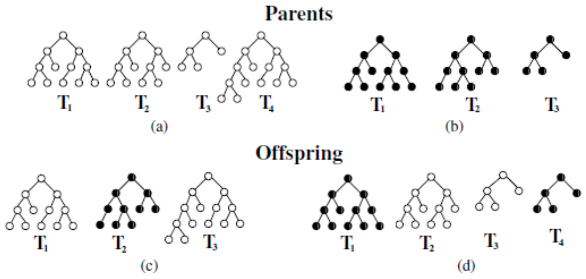


Figura 4.4: Un ejemplo de la aplicación del operador Cruza. Las figuras (a) y (b) muestran dos individuos a los que se les aplicara el operador. Las figuras (c) y (d) muestran los individuos resultantes de la operación, en este caso se cortaron $T_2^a y T_3^a$ y T_2^b generando individuos de longitud diferente a la de sus padres.

El operador mitosis es otra contribución de este trabajo y por tanto se describe en una sección individual, véase la sección 4.2.

La principal ventaja de usar operadores como cruza, gran cruza y mitosis (descrito en la siguiente sección) es que permiten a los individuos variar el número de prototipos para cada clase. Esto a su vez le permite al programa genético poder clasificar de forma correcta conjuntos de datos con estructuras complicadas que no puedan ser clasificadas correctamente con pocos prototipos. Suponga un problema de clasificación de 2 clases como el presentado en la figura 4.5. Para este problema cada clase tiene dos subclases y se puede apreciar que utilizar solo 2 prototipos no lograra obtener más del 50 % de clasificaciones correctas. Un método de generación de prototipos que genere un número predefinido de prototipos tendrá problemas para clasificar este conjunto si no se conoce desde antes que las clases en este conjunto tienen esta particularidad. Como en general no se conoce la distribución de las clases con anterioridad cualquier conjunto con estas características causaría problemas a este tipo de métodos. Mientras nuestro método se distingue por poder variar el número de prototipos generados, lo que le permite poder encontrar de forma automática el número de prototipos necesarios para clasificar a cada subclase.

También con ayuda del la medida de incompetencia, serán los prototipos que tengan un mal desempeño en los individuos los que tiendan a ser operados, permi-

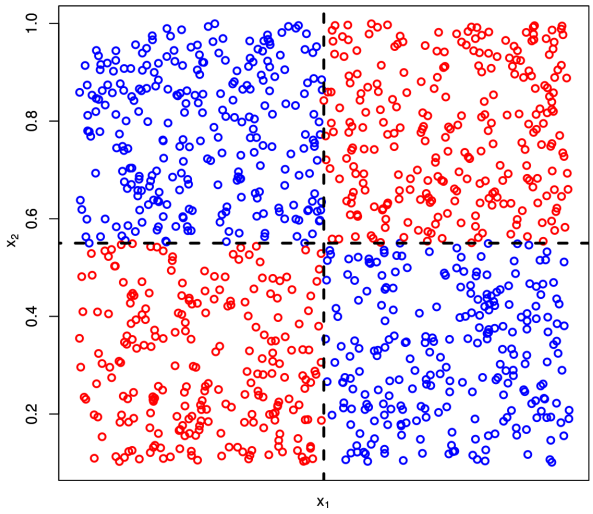


Figura 4.5: Distribución de instancias en un problema de 2 clases, cada clase presenta 2 subclases. Clase 1 puntos en azul, clase 2 puntos en rojo.

tiendo reducir o incrementar el número de prototipos en un individuo al deshacerse de prototipos que no realizan un aporte al clasificador o introduciendo prototipos clave con los que antes no se contaba.

4.2 Mitosis

Este operador está basado en el proceso de la mitosis. La mitosis es el proceso en el cual las células de un individuo se replican, en este proceso una célula genera dos células hijas que tienen su mismo material genético, aunque también suceden mutaciones durante este proceso, lo que lleva a células hijas con diferencias de su madre. En la figura 4.6 se muestra el proceso que se lleva a cabo durante la mitosis.

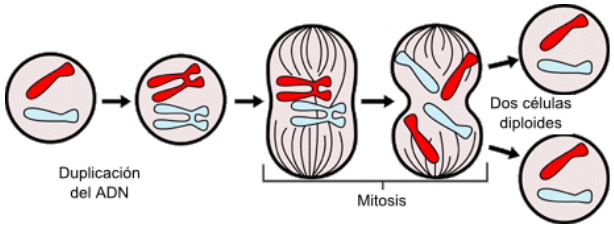


Figura 4.6: Esquema que muestra de manera resumida lo que ocurre durante la mitosis.

El operador propuesto, *mitosis*, al igual que el proceso del cual se toma la idea

actúa no sobre los individuos, sino sobre las células que lo componen, en nuestro caso los individuos son conjuntos de árboles y nuestras células son árboles. Este operador selecciona un individuo a operar por medio de torneo, para cada árbol de este individuo se calcula el número de instancias que dicho árbol clasifica como pertenecientes a su clase y no lo son (falsos positivos) i.e. para el árbol s_i se calcula la tasa de clasificación errónea obtenida de la siguiente manera:

$$TCE_i = |\{\mathbf{x}|\rho(\mathbf{x}_k, s_i) = 1;\ y_k \neq s_i^c\})|$$

Donde recordemos que

$$\rho(s_i, \mathbf{x}_k) = 1 \Leftrightarrow |\mathbf{x}_k - p_{s_i}| < |\mathbf{x}_k - p_{s_j}|\ \forall j \neq i$$

y_k es la clase de la instancia $\mathbf{x}_k$ y s_i^c es la clase del árbol s_i.

El árbol elegido (s_o) para aplicar el operador es aquel con la mayor tasa de clasificación errónea. Seleccionado el árbol s_o, se obtiene el conjunto para cada clase n: $R_n = \{\mathbf{x}|\rho(\mathbf{x}_j, s_o) = 1;\ y_j = c_n\}$ esto es, el conjunto de las instancias clasificadas por el árbol s_o de la clase n. En la figura 4.7(a) se muestra el árbol s_i y las tres clases diferentes que predice, $|R_{rombo}| = 4$, $|R_{circulo}| = 4$, $|R_{cuadrado}| = 3$. Las dos clases cuyo conjunto R_i tiene mayor cardinalidad se llamarán n_1, n_2, para cada una de ellas se genera un nuevo árbol con clase n_1 y n_2 de la siguiente manera. Se calcula el centroide (Ce_n) de R_n, esto es el punto en el espacio correspondiente a la media, atributo a atributo, de las instancias en el conjunto R_n. Además se calcula el vector (D_n) que separa al centroide (Ce_n) del prototipo p_{s_i}, el árbol generado será la suma del árbol s_i y el vector D_n esto se hace para n_1 y n_2. En la figura 4.7(b) se ilustra el centroide y el vector de separación; en la figura 4.7(c) se muestran los dos prototipos generados. Por último se decide si los árboles generados se insertarán en el individuo o si éstos remplazarán a árboles ya existentes. Esto se decide calculando para cada árbol en el individuo con la misma clase que el árbol generado el número de clasificaciones $|\{\mathbf{x}|\rho(s_n, \mathbf{x}) = 1\}|$ que realiza cada árbol y se elige el que menor número de clasificaciones realice. Si este valor es menor que un parámetro el árbol

es remplazado por el árbol generado, si no, el árbol generado es insertado en el individuo.

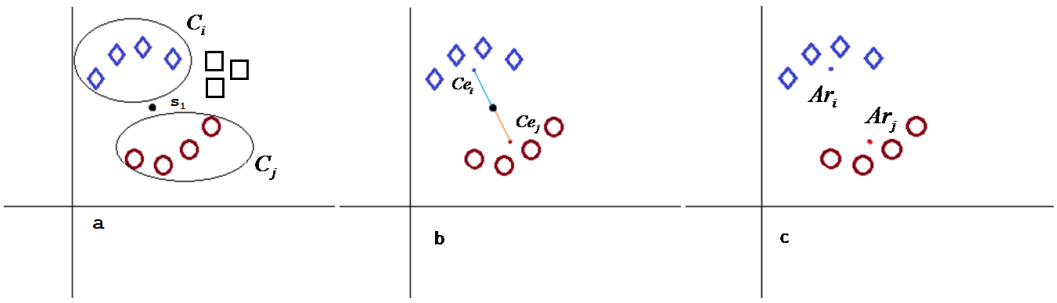


Figura 4.7: 4.7(a) muestra los ejemplos predichos por el prototipo inducido por el árbol S_1. 4.7(b) muestra los centroides Ce_i, Ce_j y el vector que separa al prototipo con el centroide. 4.7(c) Muestra los prototipos generados (Ar_i, Ar_j).

En la figura 4.7 se puede apreciar el funcionamiento del operador propuesto, mientras que en la figura 4.8 se muestra como el árbol es modificado.

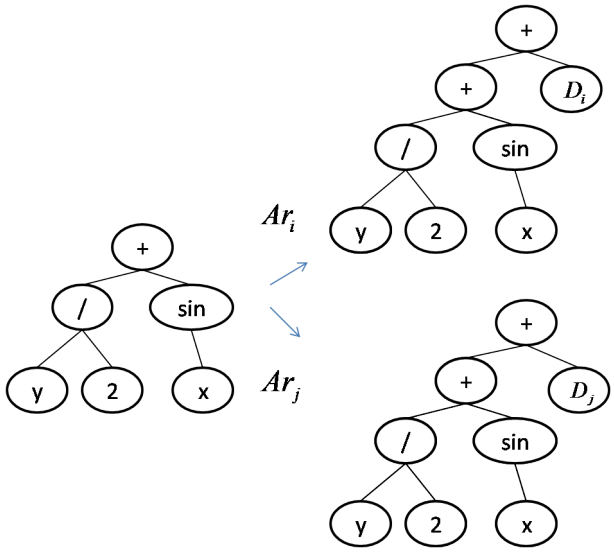


Figura 4.8: Árbol original y como es modificado para inducir el prototipo deseado.

El operador es presentado en el algoritmo 5, este operador funciona como un operador de intensificación, intentando mejorar la solución seleccionada de forma local al crear dos árboles que tenderán a clasificar correctamente la predicción que, el árbol operado antes hacía de forma incorrecta. Esto es fácil de ver pues el prototipo inducido por al menos uno de los dos nuevos árboles es el centroide de las instancias

(de una misma clase) clasificadas incorrectamente por este árbol. Por ende este nuevo prototipo ahora clasificará de forma correcta algunas de estas instancias.

Otra ventaja importante de este clasificador es la de ayudar al PG cuando se tienen pocos prototipos que clasifican la mayor cantidad de instancias (como podría pasar durante las primeras generaciones en que la inicialización de forma aleatoria de los individuos creó muchos prototipos muy alejados de los datos y unos pocos cerca de éstos); este operador puede *poner en buen camino* individuos de esta naturaleza creando prototipos más cercanos a los datos, permitiendo que los otros operadores usados sean significantes y no se convierta solamente en una búsqueda aleatoria.

En el siguiente capítulo se reportan resultados experimentales del uso del programa genético propuesto en diversas tareas de clasificación.

Algorithm 5 *Mitosis*

Seleccionar individuo I_1 usando torneo.

for $s_i \in I_1$ **do**

$TCE_i = |\{\mathbf{x}|\rho(\mathbf{x}_j, s_i) = 1;\ y_j \neq s_i^c\})|$

end for

$s_o = argmax_{\forall i}(TCE_i)$

for n=1 a N **do**

$R_n = \{\mathbf{x}|\rho(\mathbf{x}_j, s_o) = 1;\ y_j = c_n\}$

end for

A las 2 clases tal que $|R_n|$ sea mayor las llamaremos c^1, c^2.

$a(1) = c^1,\ a(2) = c^2$.

for i=1 a 2 **do**

$R(i) = R_{a(i)}$

$Ce_i = Centroide(R(i))$.

$D_i = Ce_i$ - p_{s_o}.

Crear un nodo con el operador *suma* de aridad 2.

A un nodo hijo de este operador se le asigna el vector D_i.

Al otro nodo se le asigna el árbol original s_o.

Al árbol generado lo llamaremos Ar_i y su clase será a(i).

$Tmp_i = \{s|\ a(i) = s_h^c;\ \forall s_h \in I_1\}$

for $\forall s \in Tmpi$ **do**

$Num\_clas_h = |\{\mathbf{x}|\ \rho(s_h, \mathbf{x} \in X) = 1\}|$

end for

if $min(Num\_clas_{\forall h}) < \epsilon$ **then**

Ar_i reemplaza al árbol con el menor número de clasificaciones.

else

Ar_i es insertado dentro del individuo.

end if

end for

Capítulo 5

Experimentación

En este capítulo se presentan los experimentos realizados para validar el trabajo implementado en esta tesis, se describen los conjuntos de datos usados, y se presenta una discusión sobre los resultados obtenidos. El capítulo está organizado de la siguiente forma, en la sección 5.1 se describen las características de los conjuntos de datos usados en esta experimentación. En la sección 5.2 se presentan los experimentos realizados para lograr ajustar los parámetros del método propuesto, así como los resultados obtenidos. La sección 5.3 presenta los resultados obtenidos en un gran número de problemas de clasificación, así como una comparación con otros métodos para la generación de prototipos. En la sección 5.4 se compara el método propuesto con algoritmos de clasificación estándar. Por último la sección 5.5 presenta los resultados obtenidos por el clasificador propuesto en esta tesis en la tarea de atribución de autoría.

5.1 Conjuntos de datos

Se realizaron experimentos con cuatro tipos de conjuntos de datos que se usaron para evaluar diferentes aspectos del método propuesto, los primeros se usaron para evaluar el desempeño del método propuesto ante distintas configuraciones de parámetros, mientras que los restantes se usaron para evaluar su desempeño en problemas de clasificación y para comparar el programa genético con otras alternativas. Las pruebas para la evaluación de parámetros y para la comparación del método pro-

puesto con técnicas de clasificación estándar (tradicionales y ampliamente usadas en aprendizaje computacional), se realizaron usando 10 conjuntos de datos estándar de aprendizaje computacional (*iris, cancer, balanza, dermatology, glass, haberman survival, ionosphere, blood transfusion, breast, diabetes*) tomados del repositorio de aprendizaje computacional de la *University of California at Irvine (UCI)* (Frank y Asuncion, 2010). En la tabla 5.1 se resumen las características de estos conjuntos de datos, es de notar que estos conjuntos no se encuentran separados en conjuntos de entrenamiento y prueba, por lo que se realizó un proceso de validación cruzada de 5 pliegues (*5-fold cross validation*) para evaluar el desempeño del método. La validación cruzada de k pliegues divide el conjunto de datos de forma aleatoria en k partes de igual tamaño, se realiza el proceso de aprendizaje (entrenamiento) usando *k-1* de las k partes, y el proceso de prueba del método se realiza con la k-esima que no fue utilizada en el paso anterior. Este proceso se realiza para cada una de las k partes en que se dividió el conjunto de datos. Los k resultados obtenidos son promediados para obtener un estimado del desempeño del método en este conjunto de datos.

Para la segunda etapa de evaluación del método propuesto se evaluó el desempeño del programa genético contra 25 métodos de generación de prototipos entre los que se encuentran métodos representativos del estado del arte así como métodos clásicos. Para realizar esta experimentación se hicieron pruebas con 59 conjuntos de datos tomados del repositorio de la universidad de california (Frank y Asuncion, 2010) y del de repositorio de datos KEEL (Alcalá-Fdez *et al.*, 2011). Los conjuntos de datos consideraros para esta etapa de evaluación fueron recopilados por Alcalá-Fdez et al., quienes además propusieron una taxonomía de métodos para generación de prototipos. Dichos autores proveen estos conjuntos de datos así como un marco de evaluación para métodos de generación de prototipos. En este trabajo consideramos dicho marco de evaluación para determinar cómo se compara el método propuesto con técnicas de naturaleza similar. Estos conjuntos se dividen en 2 grupos, los considerados como pequeños con menos de 2,000 instancias y aquellos como grandes, con más de 2,000 instancias véase (Triguero *et al.*, 2012) para mayores detalles. Las

ID	Características / Conjunto	Número de clases	Número de atributos	Número de instancias
1	Iris	3	4	150
2	Cancer	2	9	683
3	Balanza	3	4	625
4	Dermatology	6	34	358
5	Glass	6	9	214
6	Haberman survival	2	3	306
7	Ionosphere	2	34	351
8	Blood transfusion	2	4	748
9	Breast	6	9	106
10	Diabetes	2	8	768

Tabla 5.1: Características principales de los conjuntos de aprendizaje computacional. En el resto del capítulo usaremos los identificadores de la columna 1(ID) para hacer referencia a los distintos conjuntos de datos.

tabla 5.2 y 5.3 muestran las características de estos conjuntos.

Como se menciono previamente, una de las ventajas que se tienen al utilizar instancias en los nodos terminales de los árboles es el hecho de que al aumentar la dimensión de las instancias (el número de atributos) el tamaño de los árboles no crece. Lo que le permite aplicar el método propuesto en conjuntos de datos sin restricciones de dimencionalidad de estos. Aunque el método desarrollado no esta diseñado especialmente para este tipo de datos se desea analizar los resultados obtenidos por nuestro programa genético contra otros algoritmos especializados en este tipo de tareas para conocer el comportamiento de nuestro método en dicho escenario. Para realizar esta experimentación se decidió trabajar en el problema de atribución de autoria (AA). En este problema se tiene un conjunto de autores (clases) y un conjunto de textos de entrenamiento, de los cuales conocemos al autor de cada texto (no se consideran textos de múltiples autores). El problema consiste

Conjuntos	Instancias	Atributos	Clases	Conjuntos	Instancias	Atributos	Clases
appendicitis	106	7	2	hepatitis	155	19	2
australian	690	14	2	housevotes	435	16	2
automobile	205	25	6	iris	150	4	3
balance	625	4	3	led7digit	500	7	10
bands	539	19	2	lymphography	148	18	4
breast	286	9	2	mammographic	961	5	2
bupa	345	6	2	monks	432	6	2
car	1728	6	4	movement_libras	360	90	15
cleveland	297	13	5	newthyroid	215	5	3
zoo	101	16	7	pima	768	8	2
contraceptive	1473	9	3	saheart	462	9	2
crx	690	15	6	sonar	208	60	2
dermatology	366	33	6	spectheart	267	44	2
ecoli	336	7	8	tae	151	5	3
flare-solar	1066	11	2	tic-tac-toe	958	9	2
german	1000	20	2	vehicle	846	18	4
glass	214	9	7	vowel	990	13	11
haberman	306	3	2	wine	178	13	3
hayes-roth	160	4	3	wisconsin	699	9	2
heart	270	13	2	yeast	1484	8	10

Tabla 5.2: Características de los conjuntos pequeños en generación de prototipos.

en, dado un nuevo texto del cual no conocemos a su autor, asignarle uno de los autores considerados en la fase de entrenamiento. En la figura 5.1 se muestra este proceso. Nótese que en nuestro método cada texto se considera una instancia del conjunto de datos, a este tipo de métodos se les conoce como métodos basados en instancias. Existe otro enfoque que es también muy utilizado en el cual todos los textos de un autor son combinados en un gran texto, el cual se considera una única instancia, para más información sobre este otro enfoque véase (Stamatatos, 2009).

Esta tarea fue elegida debido a la alta dimensionalidad de los datos (8016 atributos en el conjunto de datos de menor tamaño considerado), además de ser un área practica de interés, pues en los últimos años el internet ha facilitado generación masiva de documentos digitales y por ende tareas como detección de plagio, filtrado de documentos y otras tareas relacionadas con seguridad han cobrado una gran importancia. Los métodos de atribución de autoria proveen soporte para las tareas

Conjuntos	Instancias	Atributos	Clases	Conjuntos	Instancias	Atributos	Clases
abalone	4174	8	28	ring	7400	20	2
banana	5300	2	2	satimage	6435	36	7
chess	3196	36	2	segment	2310	19	7
coil2000	9822	85	2	spambase	4597	57	2
magic	19020	10	2	splice	3190	60	3
marketing	8993	13	9	texture	5500	40	11
nursery	12960	8	5	thyroid	7200	21	3
pageblocks	5472	10	5	titanic	2201	3	2
penbased	10992	16	10	twonorm	7400	20	2
phoneme	5404	5	2				

Tabla 5.3: Características de los conjuntos grandes en generación de prototipos.

anteriores y muchas otras, de ahí la importancia de la tarea.

Para la experimentación se consideraron diversos conjuntos de datos del dominio de AA (Stamatatos, 2009). En la tabla 5.4 se resumen las características de estos conjuntos, nótese que el número de atributos para estos conjuntos es mucho mayor que para los otros conjuntos utilizados de forma usual como los presentados en las tablas 5.1, 5.2 y 5.3. Estos conjuntos de datos son procesados para obtener un vector de valores numéricos que representan a los documentos y con los cuales podemos trabajar. Una forma de representar textos mediante vectores numéricos es usando n-gramas (n-grams) a nivel caracter.

En esta técnica se tiene un vector de tamaño m, donde m es el número de caracteres en el texto. Además el i-esimo elemento de este vector corresponde con el i-esimo caracter del texto. Inicialmente se consideran los primeros n elementos de este vector, y esta cadena de caracteres es ingresada a nuestra lista de n-gramas. Posteriormente se avanza un elemento en este vector, esto es, se consideran los elementos del 2 al n+1, y la cadena formada por estos n-caracteres es agregada a nuestra lista. Este proceso se itera hasta que ya no se puedan formar cadenas de n caracteres. Por ejemplo, si queremos usar 3-gramas de caracteres para el inicio de este párrafo los 3-gramas obtenidos serían —En —, —n e—, — es—, —est—, —sta—, —ta —, etcétera. Al finalizar este proceso tendremos una lista de n-gramas. Posteriormente se aborda un enfoque basado en la bolsa de palabras binaria explicado a continua-

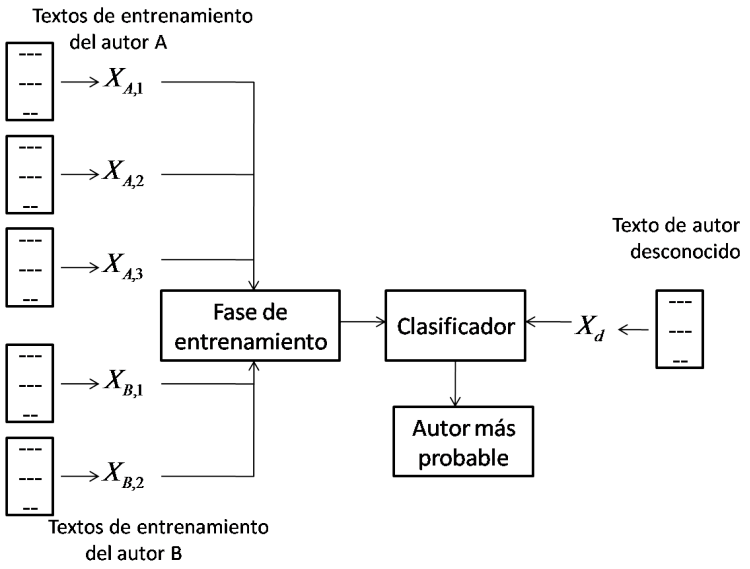


Figura 5.1: Arquitectura típica de los métodos basados en instancias

ción. Nuestro vector para cada texto será un vector binario de dimensionalidad $|V|$, donde V es el conjunto de todos los términos diferentes que ocurren en el conjunto de documentos bajo estudio (i.e. el vocabulario). Un 1 representa que ese texto tiene el n-grama correspondiente, y un 0 indica lo contrario, suponga que se tienen 3 textos diferentes los cuales son "*arma*", "*man*" y "*firma*" en la tabla 5.5 se describe el uso de n-gramas para obtener el vector que representa a los documentos de caracteres. La representación basada en n-gramas de caracteres es una de las más usadas para abordar el problema de atribución de autoria (Stamatatos, 2009).

5.2 Ajuste de parámetros

En computación evolutiva existen un conjunto de parámetros que siempre es necesario definir, por ejemplo: los porcentajes de aplicación de cada operador genético; el número de individuos; y el número de generaciones. Generalmente se toman valores que se consideran por omisión (Koza, 1990), pero cuando se introducen operadores nuevos es necesario estudiar cómo afectan estos operadores el desempeño del programa. Luego, los parámetros más importantes a definir en esta tesis son:

Características / Conjunto	Número de clases	Número de atributos	Instancias de entrenamiento	Instancias de prueba
Poetry	6	8016	145	55
Travel	4	11581	112	60
Cricket	4	10044	98	60
CCAT	10	15587	500	500
Twitter	50	26156	4500	500
NFL	3	8620	52	45
Business	6	10550	85	90

Tabla 5.4: Características principales de los conjuntos de clasificación de autoría.

n-grams / textos	ar	rm	ma	an	fi	ir
arma	1	1	1	0	0	0
man	0	0	1	1	0	0
firma	0	1	1	0	1	1

Tabla 5.5: Vectores de datos obtenidos para 3 textos diferentes.

• las probabilidades de usar cada operador genético;

• el número de individuos;

• el número de generaciones;

• el porcentaje de prototipos, con respecto al número de instancias en el conjunto de entrenamiento, que serían usados inicialmente.

Este último es de interés para este trabajo pues como los operadores usados permiten variar el número de prototipos, el uso de pocos prototipos iníciales puede incurrir a que el programa genético solo evolucione individuos con un número pequeño de prototipos, lo que hará difícil clasificar conjuntos de datos con subclases o estructuras complicadas, mientras que se desea mantener el número de prototipos utilizados a la menor cantidad posible.

El ajuste de parámetros se realizó ajustando éstos, uno a la vez, de forma secuencial; esto es, primero se ajustaron todas las probabilidades de los operadores y los valores obtenidos fueron fijados en el programa. Posteriormente se realizaron pruebas para encontrar los mejores valores para el número de individuos y generaciones, los mejores valores obtenidos de individuos y generaciones fueron fijados en el programa. Por último se realizó la experimentación sobre el número de prototipos inícials, nuevamente fijando el mejor valor obtenido para este parámetro.

Se realizó de esta manera porque tratar de ajustar todos los parámetros a la vez hubiera requerido de una gran cantidad de ejecuciones. En las siguientes subsecciones se procede a explicar los experimentos realizados y las conclusiones obtenidas de éstos.

5.2.1 Probabilidades de operadores

La elección de las probabilidades para cada operador es de gran importancia para el desempeño de este trabajo, es por esto que se realizaron pruebas con 12 configuraciones de probabilidades diferentes para cada operador. Los valores para cada operador se variaron entre valores altos, medios, bajos y nulos, para analizar como afectaban éstos al desempeño del programa. En la tabla 5.6 se muestra la probabilidad de elegir cada operador de acuerdo a cada configuración evaluada. Para estos experimentos se usaron 200 individuos y 50 generaciones usando un prototipo inicial para cada clase.

En la tabla 5.7 se muestra el resultado promedio de 5 corridas para cada conjunto usando las 12 configuraciones, éstos resultados son los obtenidos por el mejor individuo que el programa genético evoluciona evaluados en el conjunto de prueba.

De estos resultados las dos mejores configuraciones son la 5 y 8. La configuración 5 corresponde a un alto uso de mutación por punto y bajo uso de los otros operadores. La configuración 8 en cambio presenta un alto uso del operador mitosis y niveles bajos para los operadores de cruza y un nivel medio de mutación por punto.

Operadores / Configuración	Mutación	Cruza	G. Cruza	Mitosis
1	10 %	60 %	30 %	0 %
2	5 %	30 %	60 %	5 %
3	0 %	30 %	60 %	10 %
4	10 %	0 %	60 %	30 %
5	70 %	10 %	10 %	10 %
6	70 %	15 %	0 %	15 %
7	25 %	25 %	25 %	25 %
8	20 %	10 %	10 %	60 %
9	20 %	0 %	20 %	60 %
10	20 %	60 %	0 %	20 %
11	0 %	20 %	10 %	70 %
12	10 %	70 %	20 %	0 %

Tabla 5.6: Configuraciones de porcentajes usados para cada operador genético.

Se realizó una prueba de rangos con signos de Wilcoxon para determinar si los resultados de estas configuraciones tenían la misma distribución. El resultado de esta prueba determinó que no se puede rechazar la hipótesis nula de que las configuraciones tienen la misma media con un nivel de significancia del 95 %. A pesar de que no existiera una diferencia significativa, la configuración 8 fue la que obtuvo mejores resultados en la mayor cantidad de conjuntos de datos siendo la mejor configuración en 3 de éstos, además de ser la mejor configuración considerando el promedio, por lo cual se decidió elegir esta configuración para la experimentación subsecuente.

En la figura 5.2 se muestra una gráfica tipo *boxplot* para los resultados de cada configuración, donde la línea dentro de las cajas es la mediana, los bordes de la caja son los 25 y 75 percentiles y la línea punteada se extiende hasta los puntos más extremos que no se consideran errores. Éstos valores fueron obtenidos estandarizando los resultados para cada conjunto de datos.

ID \ Conf	1	2	3	4	5	6	7	8	9	10	11	12
1	92.5	90.9	89.1	93.1	94.4	93.7	94.4	**95.9**	94.3	89.5	93.6	91.2
2	70.9	76.3	74.6	72.5	69.8	72.8	**77.1**	71.5	65.8	73.4	71.0	75.4
3	70.4	73.6	**80.2**	78.2	79.3	77.7	71.6	77.8	72.6	79.7	70.8	78.4
4	80.2	90.5	90.4	89.9	93.2	91.4	92.0	91.9	93.5	**94.0**	86.2	82.6
5	46.9	52.4	52.6	53.8	53.9	**58.1**	55.5	56.4	53.9	53.9	51.3	39.8
6	71.7	66.7	71.5	73.5	**73.6**	73.5	73.5	71.5	73.3	73.5	73.5	70.8
7	64.8	64.8	65.1	64.8	65.6	65.9	68.3	**70.0**	69.2	63.8	69.6	64.1
8	73.9	76.2	76.2	74.0	74.5	76.2	76.2	76.2	76.2	76.2	76.2	**76.2**
9	38.0	43.6	42.1	43.4	**46.3**	40.8	41.6	41.8	45.4	40.8	37.2	37.7
10	65.1	65.1	65.1	65.1	65.1	65.1	65.1	65.1	65.1	65.1	**65.1**	65.1
Promedio	67.5	70.0	70.7	70.8	71.6	71.5	71.5	**71.8**	70.9	71.0	69.5	68.1

Tabla 5.7: Porcentaje de exactitud de las 12 configuraciones para todas las bases de datos, se muestra en negritas el mejor resultado de cada configuración.

El uso de la exactitud estandarizada es debido a que se trabaja con diferentes conjuntos de datos, tome por ejemplo los resultados obtenidos para el conjunto de datos 1 y los del conjunto 5, los resultados de las 12 configuraciones para el conjunto de datos 1 rondan entre los 89.1 % y los 95.9 %, mientras que los resultados de las 12 configuraciones para el conjunto de datos 6 ronda entre los 39.8 % y los 56.4 %. Vemos que existe un varianza significativa entre los resultados obtenidos para diferentes conjuntos, y para evitar conflictos debido a esta varianza entre conjuntos se usa la exactitud estandarizada.

Es de interés notar que aquellas configuraciones en las que no se uso el operador propuesto (mitosis) en esta tesis fueron las que tuvieron peores resultados, estas son las configuraciones 1 y 12 que además se caracterizan por tener valores bajos de mutación. Véase también como la configuración 2 utiliza valores bajos de mutación y mitosis, mientras la configuración 3 aumenta el valor de la mitosis al eliminar el de la mutación, en este caso la configuración 3 obtiene mejores resultados que los obtenidos por la configuración 2. Es interesante ver que los resultados obtenidos con

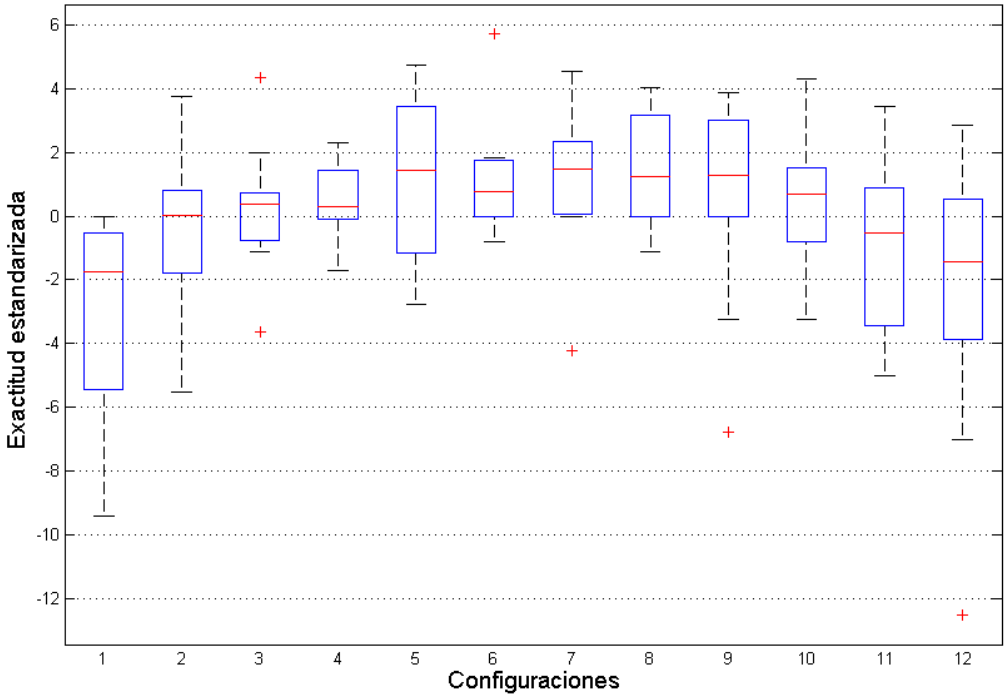


Figura 5.2: Gráfica de boxplot para las 12 configuraciones contra la exactitud ponderada.

las configuraciones 5, 6, 7, 8, 9 y 10 son las seis mejores configuraciones, en dos de las cuales (5 y 6) se presenta el alto uso de la mutación, en las configuraciones 8 y 9 se tiene un alto uso del operador mitosis, mientras que la configuración 7 presenta un uso medio para todos sus operadores, y por último la configuración 10 presenta un elevado uso del operador cruza. Analizando las otras 6 configuraciones (1, 2, 3, 4, 11 y 12) vemos que su principal característica es el alto uso de los operadores de cruza o gran cruza, donde solo la configuración 11 utiliza niveles elevados de mitosis. Esto nos lleva a concluir que aunque el uso del operador mitosis ayuda a obtener mejores resultados, es necesario el utilizar el operador mutación en conjunto con éste para obtener los mejores resultados. Este resultado no es del todo sorprendente si se analiza la naturaleza de los operadores empleados. El operador mitosis desempeña en cierta medida el de una búsqueda de especificación al modificar los árboles buscando mejorar inmediatamente la solución actual. En cambio el operador mutación realiza

una búsqueda de diversificación modificando los árboles de tal manera que permite explorar el uso de nuevos prototipos. Mientras que ambos operadores de cruza empleados no modifican los árboles si no que solo los intercambian entre individuos.

5.2.2 Individuos y generaciones

Después de fijar las probabilidades para cada operador, el determinar el número de individuos y generaciones se vuelve lo más esencial, en la literatura se sugiere usar la mayor cantidad posible de individuos, mientras que se recomienda un número pequeño de generaciones (Koza, 1990).
Para fijar éstos valores se realizaron pruebas con los 10 conjuntos de datos antes mencionados, se realizaron pruebas variando el número de individuos con valores de 10, 25, 50, 100 y 200, así como variando el número de generaciones con valores de 10, 25, 50, 100 y 200, se hicieron 5 corridas para cada configuración posible de individuos y generaciones; los resultados presentados son el promedio de estas 5 corridas. La exactitud estandarizada es calculada igual que en la sección anterior.

En la figura 5.3 se observa un gráfico del número de generaciones contra la exactitud estandarizada en el conjunto de datos de prueba y entrenamiento promediada para 5 corridas sobre los conjuntos de datos 1 al 10.

En la figura podemos observar que el algoritmo en verdad evoluciona las soluciones inícials a mejores soluciones, pero no existe evidencia sobre la cantidad de generaciones que es más conveniente utilizar, mientras que en el conjunto de prueba se obtuvieron los mejores resultados para 50 y 200 generaciones, en el conjunto de entrenamiento las mejores soluciones se obtuvieron con 200 generaciones y pareciera que al incrementar el número de generaciones mejoraría el valor de la exactitud. Los resultados en el conjunto de entrenamiento no son de extrañar y forman parte de la naturaleza del aprendizaje computacional; una solución lo suficientemente compleja es capaz de memorizar todo el conjunto de entrenamiento, pero puede fallar totalmente en el conjunto de prueba, piense en un estudiante que se memoriza todos

las respuestas de la tarea, pero en el examen al ser diferentes ejercicios no puede contestar ninguno, a este fenómeno se le conoce como sobreajuste véase la sección 3.6.

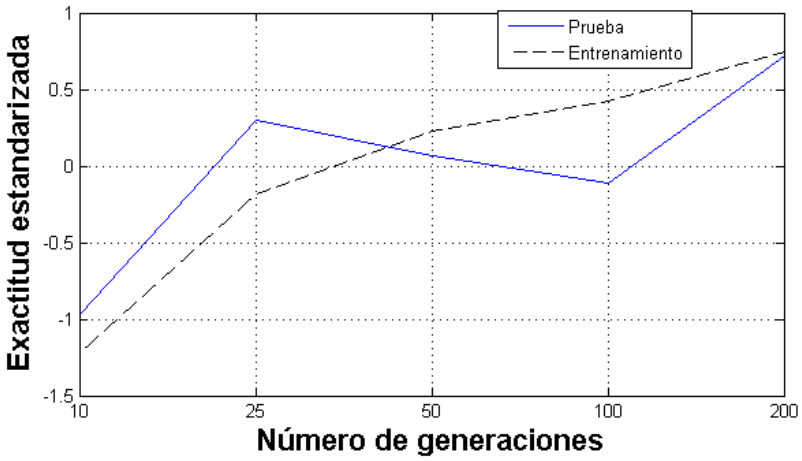


Figura 5.3: Como varía la exactitud estandarizada en el conjunto de entrenamiento y prueba conforme aumenta el número de generaciones.

En la figura 5.4 se muestra una gráfica del número de individuos contra la exactitud estandarizada en el conjunto de prueba y entrenamiento promediada para 5 corridas sobre los conjuntos de datos 1 al 10. Se ve como al aumentar el número de individuos el clasificador obtenido por el programa genético tiende a ser mejor para ambos conjuntos.

Esto es bastante intuitivo, pues al tener una mayor cantidad de individuos permite explorar mayor parte del espacio de soluciones lo que repercute directamente en tener más posibilidades de obtener mejores soluciones. Una de las características de las soluciones iniciales es que son menos complejas que las soluciones que han sido generadas a través de los operadores genéticos. Al aumentar el número de soluciones iniciales se evita enfocar todo el proceso de búsqueda en unas pocas soluciones, esto último llevaría a utilizar los operadores genéticos constantemente en estas pocas soluciones por ende haciéndolas muy complejas.

Combinando los resultados de ambos experimentos podemos concluir que los operadores genéticos utilizados en este trabajo son capaces de mejorar las soluciones

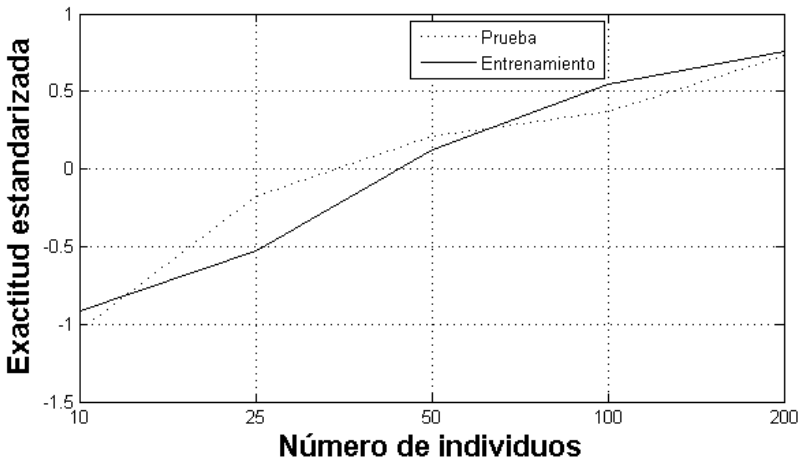


Figura 5.4: Como varía la exactitud estandarizada en el conjunto de prueba y entrenamiento conforme aumenta el número de individuos.

iniciales, pero también son sensibles al sobreajuste. Mientras que el uso de una mayor cantidad de individuos ayuda a disminuir el impacto que se tiene en el sobreajuste.

La figura 5.5 presenta un gráfico de intensidad de color, en el cual se grafican todas las configuraciones de número de individuos contra número de generaciones, en la gráfica el color representa la exactitud estandarizada, y entre más oscuro sea, mayor es la exactitud. En esta figura vemos que la mejor configuración *número de individuos / número de generaciones* es usando 200 individuos y 50 generaciones. Esta será la configuración de individuos y generaciones que se usarán para el resto de los experimentos. Los valores obtenidos se muestran en la tabla 5.8.

Individuos / Generaciones	10	25	50	100	200
10	-0.88	-0.29	0.19	0.32	0.65
25	-1.34	-0.27	0.14	0.63	0.84
50	-1.07	-0.09	-0.12	0.38	0.89
100	-1.36	-0.33	0.44	0.37	0.88
200	-1.04	0.09	0.38	0.17	0.40

Tabla 5.8: Exactitud estandarizada de individuos contra generaciones.

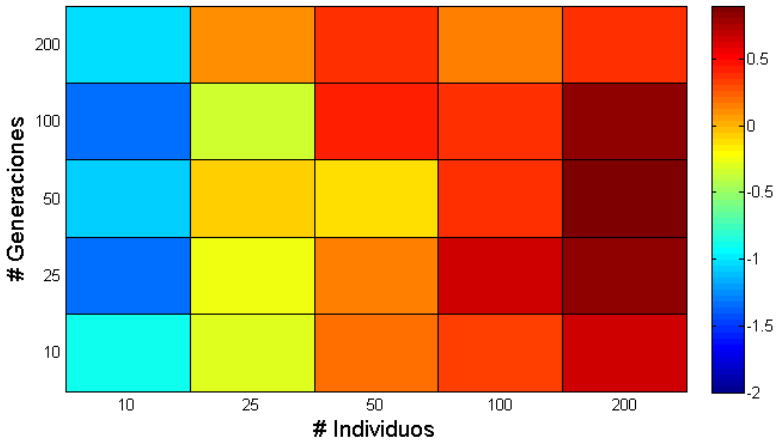


Figura 5.5: Gráfico de colores de individuos vs generaciones.

5.2.3 Porcentaje de prototipos iniciales

Determinar el porcentaje de prototipos iníciales es de vital importancia para este trabajo, pues en general no se conoce si en los conjuntos de datos existen subclases de forma apriori; esto puede llevar a tener individuos que utilizan solo un prototipo para clasificar una clase con subclases, lo cual tendrá como consecuencia que el programa genético sacrifique el clasificar correctamente algunas de estas subclases. Por esta razón es importante tener un número considerable de prototipos iníciales. La otra cara de la moneda es utilizar demasiados prototipos iniciales que existan prototipos que no realicen ningún aporte a la clasificación de las instancias, y que sean mas de los que los operadores genéticos del algoritmo desarrollado puedan reducir a buenos niveles. Esto ademas hace que el proceso de evaluación de la aptitud de los individuos sea más lenta. En esta sección solo se analiza como mejora la exactitud al aumentar el número de prototipos iniciales.

Se busca que el número de prototipos sea menor al número de instancias en la base de datos, lo que permite usar menor cantidad de memoria y tiempo de cómputo para realizar los cálculos pertinentes para clasificar una instancia nueva. Por esta razón en lugar de determinar un número arbitrario de prototipos, se decidió probar con porcentajes del número de instancias en el conjunto de entrenamiento, i.e., si el

tamaño del conjunto de entrenamiento es de 500 y se usa un porcentaje inicial del 10 %, se procederá a inicializar los individuos con 50 prototipos cada uno, distribuidos de la forma presentada en la sección 4.1.

Se realizaron pruebas para los siguientes porcentajes: 0 %, 2.5 %, 5 % y 10 %, haciendo 5 corridas para las 10 bases de datos ya mencionadas, con 0 % utilizamos un prototipo por clase. Los resultados se resumen en la tabla 5.9.

Porcentajes / **Conjunto**	**0 %**	**2.5 %**	**5 %**	**10 %**
1	93.73 %	**95.87** %	89.73 %	**95.87** %
2	51.50 %	86.63 %	87.06 %	**89.23** %
3	80.99 %	**89.34** %	88.22 %	86.82 %
4	91.67 %	92.73 %	**93.12** %	92.95 %
5	**61.84** %	55.20 %	59.95 %	60.61 %
6	72.87 %	73.00 %	**73.59** %	73.00 %
7	68.62 %	74.47 %	77.32 %	**79.36** %
8	73.70 %	76.20 %	76.20 %	**76.25** %
9	43.95 %	44.53 %	**47.17** %	44.52 %
10	62.38 %	65.05 %	**65.10** %	64.92 %
promedio	70.12 %	75.30 %	75.74 %	**76.35** %

Tabla 5.9: Porcentaje de exactitud para las 4 configuraciones de porcentajes usados, en negritas se presenta el mejor resultado para cada conjunto.

Se realizó una prueba de Wilcoxon para pares de porcentajes, esto es, se realizó una prueba para los resultados obtenidos para los porcentajes de 0 % y 2.5 %, 0 % y 5 %, 0 % y 10 % y así para todas las combinaciones posibles. La hipótesis nula para estas pruebas es que ambos resultados vienen de una distribución con medias iguales, contra la alternativa que no tienen la misma media (i.e. una es más grande que la otra). En la tabla 5.10 se presentan los resultados de éstos experimentos para todas las combinaciones 2 a 2 posibles.

	0 vs 2.5	**0 vs 5**	**0 vs 10**	**2.5 vs 5**	**2.5 vs 10**	**5 vs 10**
Valor de P	0.0014	0.00066	0.000063	0.7834	0.3948	0.5734
Conclusión	rechazar	rechazar	rechazar	no rechazar	no rechazar	no rechazar

Tabla 5.10: Pruebas de Wilcoxon para todas las combinaciones 2 a 2 de posibles porcentajes de prototipos iniciales.

De éstos resultados podemos ver que para todos los experimentos en que se prueba la configuración de 0 % siempre se rechaza la hipótesis nula de que ambas medias son iguales, con éstos resultados y los de la tabla 5.9 concluimos que el uso de más de un prototipo inicial representa un incremento estadísticamente significativo. Mientras que no existen diferencias estadísticamente significativas entre los resultados cuando se usan diferentes porcentajes de prototipos extra. Aunque no es estadísticamente significativo, la configuración de 10 % fue la que obtuvo los mejores resultados por eso será el número de prototipos iniciales que usaremos en lo sucesivo.

En PG tradicional existe el problema conocido como *bloat* (inflación) en el que los árboles crecen tanto que se vuelven computacionalmente intratables; por la naturaleza de nuestros operadores en el trabajo propuesto no se tiene este problema. Sin embargo, el número de árboles (prototipos) en nuestros individuos puede crecer sin control, llevando a problemas similares a aquellos que se presentan con el *bloat*. Durante los experimentos realizados para definir estos parámetros se presentaron individuos cuyo número de árboles crecía demasiado, haciendo muy lenta la optimización en el programa genético. Para evitar esos problemas en la PG tradicional existen técnicas como la poda (*prune*) en la cual se reduce el tamaño de los árboles al cambiar nodos no-terminales por nodos terminales. En este trabajo se propone una idea similar "*tala*", en la que se eliminan árboles que no son útiles para el bosque (individuo). En la siguiente subsección se presentan esta estrategia y los resultados obtenidos con su uso.

5.2.4 Tala

Como se ha venido trabajando, el programa genético desarrollado en esta tesis es un método de generación de prototipos, hasta ahora fuera de la elección del número de prototipos iniciales, la reducción del conjunto de prototipos se ha dejado como tarea para los operadores genéticos implementados en el programa genético. Es de especial interés recordar que la función de aptitud empleada en esta tesis se basa en la exactitud y el *margen*, esto implica que el programa genético no tendría ninguna preferencia por un individuo con 100 prototipos y otro con 10, siempre y cuando ambos tuvieran el mismo valor de aptitud. Los resultados en reducción del número de prototipos hasta ahora, aunque no han sido reportados han demostrado ser buenos. Sin embargo, considerando los problemas que se comentaron al final de la sección 5.2.3 se decidió implementar una estrategia que ayudara a reducir el número de prototipos en cada individuo a la vez que evitara una disminución en la exactitud obtenida por el programa genético.

Esta estrategia la denominamos como *tala*. En esta estrategia cada cierto número de generaciones se verifica el número de predicciones que realiza cada árbol, si este árbol tiene menos de 2 predicciones (ya sean correctas o incorrectas) el árbol es eliminado a menos que sea el único árbol restante de su clase, pues de otra forma nuestro programa genético no tendría ninguna manera de generar un nuevo prototipo para esta clase. De esta forma se eliminan árboles que no aporten utilidad a los individuos y ayuda a evitar los problemas de sobrepoblación.

Se realizó experimentación para analizar las consecuencias del uso de esta estrategia sobre la exactitud alcanzada. Para esto se decidió usar la estrategia de *tala* usando los mismos porcentajes de prototipos iniciales que los de la sección 5.2.3. Esto es, la cantidad de prototipos que se generaran por individuo dependerá de un porcentaje con respecto al número de instancias en el conjunto de entrenamiento, estos porcentajes son del 2.5 %, 5 % y 10.

Se realizaron pruebas de Wilcoxon comparando mismos porcentajes de prototi-

pos iniciales usando el paso de remover árboles contra los de no hacerlo, obtenidos en la sección 5.2.3 Esto es, se compararon los resultados del 2.5 % obtenidos empleando la *tala* contra los resultados obtenidos usando el 2.5 % sin usar esta estrategia, y las mismas pruebas respectivamente para 5 % y 10 %.

En la tabla 5.11 se resumen los resultados de estas pruebas, de estos resultados concluimos que no existe una diferencia significativa entre las medias de los resultados obtenidos empleando la estrategia de tala, contra no usarla.

	2.5 vs 2.5	**5 vs 5**	**10 vs 10**
Valor de P	0.9361	0.7699	0.8152
Conclusión	no rechazar	no rechazar	no rechazar

Tabla 5.11: Pruebas de Wilcoxon comparando mismos porcentajes de prototipos iníciales usando el paso de remover árboles contra no hacerlo.

En la tabla 5.12 se muestra la comparación de los resultados obtenidos para los diferentes porcentajes usando y sin usar la tala. Aunque la diferencia no sea significativa los resultados obtenidos usando la *tala* son un ligeramente mejores que los obtenidos sin usarse. Aun más importante el número promedio de prototipos disminuye de forma considerable y el tiempo de correr el programa genético mejora significativamente usando esta estrategia y se evitan los problemas de sobrepoblación.

Como se probo anteriormente el uso de la estrategia *tala* ayuda en una pequeña medida a incrementar la exactitud en el conjunto de prueba obtenida por nuestro algoritmo. Luego, es de interés analizar como cambia el número de prototipos al ir pasando las generaciones, poder ver el efecto que causa esta estrategia en el número de prototipos, ademas de como afectan los operadores genéticos. Para esto se incluye en la figura 5.6 un gráfico que presenta el número de prototipos promedio para cada generación, esta figura presenta los resultados obtenidos para una corrida del conjunto de datos *iris*.

En esta figura se observa como en la generación 5 (cuando entra en efecto el

	2.5 %	**2.5 % Tala**	**5 %**	**5 % Tala**	**10 %**	**10 % Tala**
1	95.87 %	94.40 %	89.73 %	93.87 %	95.87 %	**97.07** %
2	86.63 %	86.99 %	87.06 %	87.64 %	89.23 %	**89.32** %
3	89.34 %	**89.66** %	88.22 %	88.58 %	86.82 %	87.01 %
4	92.73 %	89.18 %	93.12 %	91.28 %	**92.95** %	92.16 %
5	55.20 %	61.50 %	59.95 %	63.00 %	60.61 %	**64.05** %
6	73.00 %	**73.65** %	73.59 %	73.39 %	73.00 %	73.45 %
7	74.47 %	75.72 %	77.32 %	75.71 %	**79.36** %	78.23 %
8	76.20 %	76.20 %	76.20 %	76.34 %	76.25 %	**76.47** %
9	44.53 %	42.53 %	**47.17** %	44.54 %	44.52 %	43.80 %
10	65.05 %	65.12 %	65.10 %	65.07 %	64.92 %	**65.15** %

Tabla 5.12: Resultados comparando mismos porcentajes de prototipos iníciales usando el paso de remover árboles contra no hacerlo. En negritas se presenta el mejor resultado para cada conjunto de datos.

uso de la estrategia *tala*) el número de prototipos promedio decrece dramáticamente, mientras que las siguientes generaciones múltiplos de 5 la disminución ya no es tan drástica.

Además se observa el efecto de los operadores genéticos que durante las generaciones 5 a 10 disminuyen aun mas el número de prototipos promedio, mientras que en las generaciones 25 a 35 buscan incrementar este número. También se puede observar como durante la generación 30, la estrategia *tala* no elimina ningún prototipo, pues todos eran útiles para el clasificador en ese momento.

En la tabla 5.13 se resumen los parámetros obtenidos con las pruebas descritas en esta sección. Esta configuración de parámetros se usará en el resto de la experimentación.

Mutación	**Cruza**	**Gran Cruza**	**Mitosis**	**Ind.**	**Generaciones**	**P. iníciales**
20 %	10 %	10 %	60 %	200	50	10 %

Tabla 5.13: Configuración final de los parámetros del PG

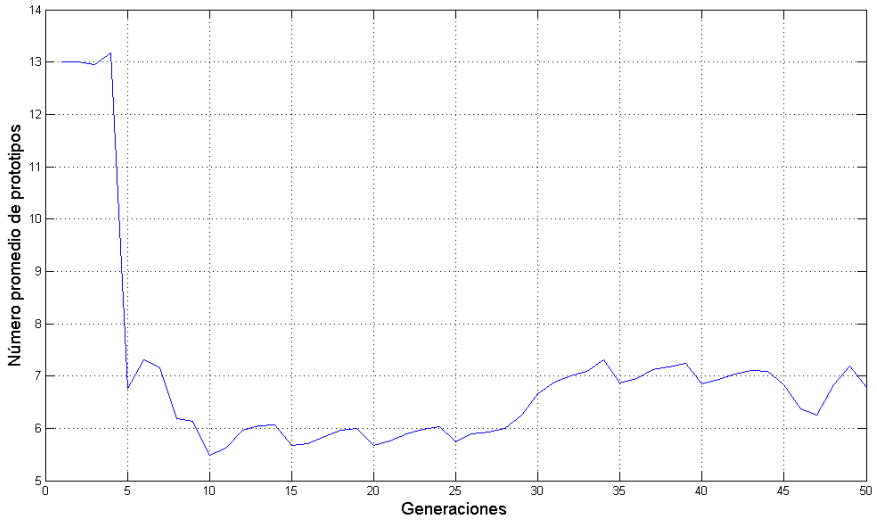


Figura 5.6: Se presenta el efecto de la estrategia tala, así como los cambios que

5.3 Experimentación en generación de prototipos

Esta sección describe experimentos y resultados del uso del programa genético en la tarea de generación de prototipos, utilizando para esto el marco de evaluación propuesto por (Triguero *et al.*, 2012). El objetivo de este experimento es comparar el desempeño del método propuesto permitiendo poder clasificarlo tanto en términos de reducción del conjunto de prototipos generados, así como en poder de clasificación contra un gran número de tÃ©cnicas representativas del estado del arte, así como métodos clásicos que abordan esta misma tarea desde una perspectiva diferente. Esta experimentación se realizó sobre los conjuntos de datos definidos en la tabla 5.2 y 5.3, para ambos conjuntos se realizo un proceso de validación cruzada de 10 pliegues, los 10 resultados son promediados para obtener el desempeño del programa genético. En esta fase de experimentación es de nuestro interés analizar como se desempeña el programa genético desarrollado ante métodos del estado del arte en generación de prototipos. En esta área de aplicación las medidas de desempeño estándar son la exactitud en el conjunto de prueba, y el porcentaje de reducción. Esta ultima se

mide como 1 menos el cociente del número de prototipos generados y el número de instancias en el conjunto de entrenamiento.
Cabe mencionar que no se presentan los resultados de obtenidos con la medida *Kappa* y los tiempos de ejecución que se presentan en (Triguero *et al.*, 2012), pues en este mismo articulo se concluye que la medida *Kappa* y la exactitud son muy similares. El tiempo de computo no se incluye pues las diferentes velocidad de procesamiento en las maquinas en que se realizaron las pruebas hacen que estos valores no sean comparables. Siguiendo el esquema de evaluación propuesto por (Triguero *et al.*, 2012), primero de evalúa y compara el método propuesto en conjuntos de datos pequeños y posteriormente se presenta el análisis en conjuntos de datos grandes, véase la sección 5.1.

5.3.1 Conjuntos pequeños

En la tabla 5.14 se presentan los resultados obtenidos para los conjuntos denominados como pequeños (número de instancias menor a 2,000), en esta tabla se presenta el promedio de los resultados obtenidos sobre todos los conjuntos de datos. Se reportan resultados obtenidos con el método propuesto, así como los resultados obtenidos por los 25 métodos para generación de prototipos consideraros en el estudio de (Triguero *et al.*, 2012). Se reporta la exactitud promedio de entrenamiento, de prueba y el porcentaje de reducción del conjunto de datos original. Nótese que los resultados de cada columna están ordenados de acuerdo al desempeño de los métodos.

Como puede observarse en la reducción el programa genético solo logra catalogarse en la posición número 15 con una reducción del 94.87 % pero hay que notar que solo métodos como PSCSA y AVQ tienen una reducción considerablemente superior a este valor, mientras que los demás métodos que se posicionan de manera superior al programa genético obtienen valores de reducción que se encuentran muy cercanos a este. Más aun, de aquellos métodos que se posicionan por arriba del PG en la categoría de reducción, solo PSO logra posicionarse por arriba en la categoría

	Reducción		Exactitud entrenamiento		Exactitud prueba
PSCSA	0.9858	MCA	0.8772	GENN	0.7564
AVQ	0.9759	GMCA	0.8405	ICPL2	0.756
LVQTC	0.9551	HYB	0.8309	PSO	0.7501
MixtGauss	0.9552	ICPL2	0.8254	PG	**0.7384**
MSE	0.952	ENPC	0.8247	GMCA	0.7351
Chen	0.9519	PSO	0.8238	1NN	0.7326
BTS3	0.9519	GENN	0.8002	RSP3	0.7325
SGP	0.9512	RSP3	0.7924	Depur	0.7296
LVQPRU	0.9503	Depur	0.7801	MSE	0.7237
PSO	0.9491	MSE	0.7566	MCA	0.7219
VQ	0.9491	PG	**0.7478**	ENPC	0.7167
DSM	0.9491	1NN	0.7369	HYB	0.7153
LVQ3	0.9488	LVQTC	0.7327	LVQPRU	0.6997
PG	**0.9487**	LVQPRU	0.7304	LVQTC	0.6981
PNN	0.9447	SGP	0.7256	SGP	0.6949
AMPSO	0.9436	AMPSO	0.7227	MixtGauss	0.6932
MCA	0.8568	MixtGauss	0.7138	AMPSO	0.6903
ICPL2	0.8371	DSM	0.7036	DSM	0.681
RSP3	0.7329	PNN	0.7015	PNN	0.6786
ENPC	0.722	Chen	0.6964	Chen	0.677
GMCA	0.6984	LVQ3	0.6931	LVQ3	0.6763
POC	0.6071	AVQ	0.6869	PSCSA	0.6682
HYB	0.4278	PSCSA	0.6787	AVQ	0.6672
Depur	0.3531	BTS3	0.6713	BTS3	0.6626
GENN	0.1862	VQ	0.6614	VQ	0.6549
1NN	0	POC	0.6487	POC	0.6493

Tabla 5.14: Resultados para reducción de la cardinalidad del conjunto de prototipos promedio, la exactitud promedio en el conjunto de entrenamiento y prueba para los conjuntos pequeños.

de exactitud en la etapa de prueba.

El resultado más importante de la tabla 5.14 es que nuestro programa genético se clasifica entre los primeros en exactitud en los datos de prueba, solo por debajo de GENN, ICPL2 y PSO. Dado que el fin de cualquier método de clasificación es maximizar la efectividad de clasificación en datos no vistos (de prueba), este resultado muestra que el método propuesto es altamente efectivo y se compara favorablemente con la mayoría de los otros métodos basados en prototipos. Nótese que cada uno de los métodos con los que nos estamos comparando es un método que resuelve el mismo problema que el programa genético propuesto y cada método ha sido evaluado

rigurosamente por sus respectivos creadores. Además, dado que se ha evaluado al programa genético en una gran variedad de conjuntos de datos estándar podemos afirmar que el método de clasificación es robusto cuando se consideran conjuntos de datos con pocas instancias.

Con base en estos resultados podemos concluir que el PG desarrollado en esta tesis obtiene una buena posición entre aquellos métodos del estado del arte en generación de prototipos para los conjuntos pequeños. No obstante lo anterior, una dirección inmediata para trabajo futuro es mejorar el programa genético desarrollado para posicionarlo mejor con respecto a los métodos relacionados.

En la imagen 5.7, se muestra un gráfico de dispersión sobre la exactitud promedio y la reducción (en porcentaje) con respecto a la cardinalidad del conjunto de prototipos y el conjunto de entrenamiento para todos los métodos, en esta imagen se puede apreciar como solo PSO logra superar al PG en ambos frentes.

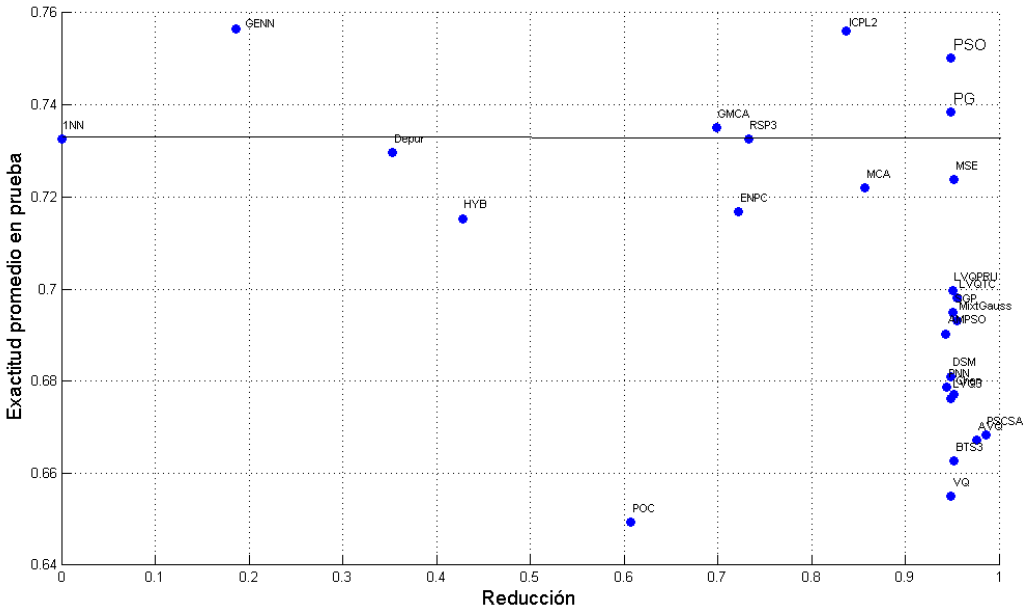


Figura 5.7: Gráfica de dispersión de exactitud en conjunto de prueba y reducción de cardinalidad para conjuntos pequeños, se gráfica la linea obtenida por el método *1-NN* como referencia.

PSO, *ENPC* y nuestro programa genético se clasifican como los mejores méto-

dos dentro de aquellos catalogados como *positioning adjustment*, que se refieren a aquellos métodos que buscan corregir la posición del conjunto de prototipos siguiendo un proceso de optimización, mientras que nuestro programa genético no logra superar a *PSO*, si logra superar a *ENPC* tanto en reducción como en exactitud en el conjunto de prueba como puede apreciarse en la figura 5.7.
Dentro de la taxonomía de los métodos de generación de prototipos presentados en la figura 3.3, se clasifica a nuestro programa genético en las mismas categorías que los algoritmos *ENPC* y *AMPSO*. Como ya se menciono, nuestro programa genético logra superar a *ENPC*, ademas para *AMPSO* el programa genético obtiene mejores resultados en los frentes de reducción y exactitud en los datos de prueba. También podemos observar que nuestro método obtiene una taza de sobreajuste menor que la obtenida por *ENPC*. De entre los algoritmos clásicos solo es *GENN* el que lograr superar a nuestro programa genético en términos de exactitud en el conjunto de prueba, sin embargo, *GENN*, obtiene los peores resultados en términos de reducción (sin contar al método *1-NN*), obteniendo porcentajes de reducción muy bajos que no representan una reducción significativa en tiempo de procesado y uso de memoria usando un enfoque *K-NN*.

Evidentemente, para los conjuntos de datos pequeños nuestro programa genético obtiene mejores resultados que los obtenidos por el clasificador *1-NN*, esto quiere decir, que ademas de realizar una reducción significativa, el conjunto de prototipos generados presenta un nivel de clasificación mayor que el obtenido usando el conjunto de entrenamiento original. Esto es, bajo un esquema de clasificación *1-NN* el conjunto de prototipos obtiene un mejor resultado de clasificación que si se usara el conjunto de entrenamiento inicial. Por lo que podemos concluir que para este tipo de conjuntos, en promedio nuestro programa genético se desempeña de manera satisfactoria como un método de generación de prototipos al obtener un nivel de reducción significativa y mejorar el nivel de clasificación que se obtiene bajo un esquema de clasificación *1-NN*.

Debido a que nuestro programa genético es un método evolutivo, en el que al

pasar las generaciones se busca ir mejorando a los individuos es interesante analizar el comportamiento de la cantidad de prototipos y la exactitud en el conjunto de prueba contra el número de generaciones. Para esto en la figura 5.8 se presentan los resultados obtenidos para una corrida del programa genético sobre el conjunto de datos *Monk*, en esta figura se muestra como varia tanto la exactitud promedio, como el número de prototipos promedio para todos los individuos en cada generación y además se grafica la exactitud del mejor individuo encontrado. Notese que la exactitud fue multiplicada por un factor de 15 para que sus valores pudieran ser apreciados en la imagen.

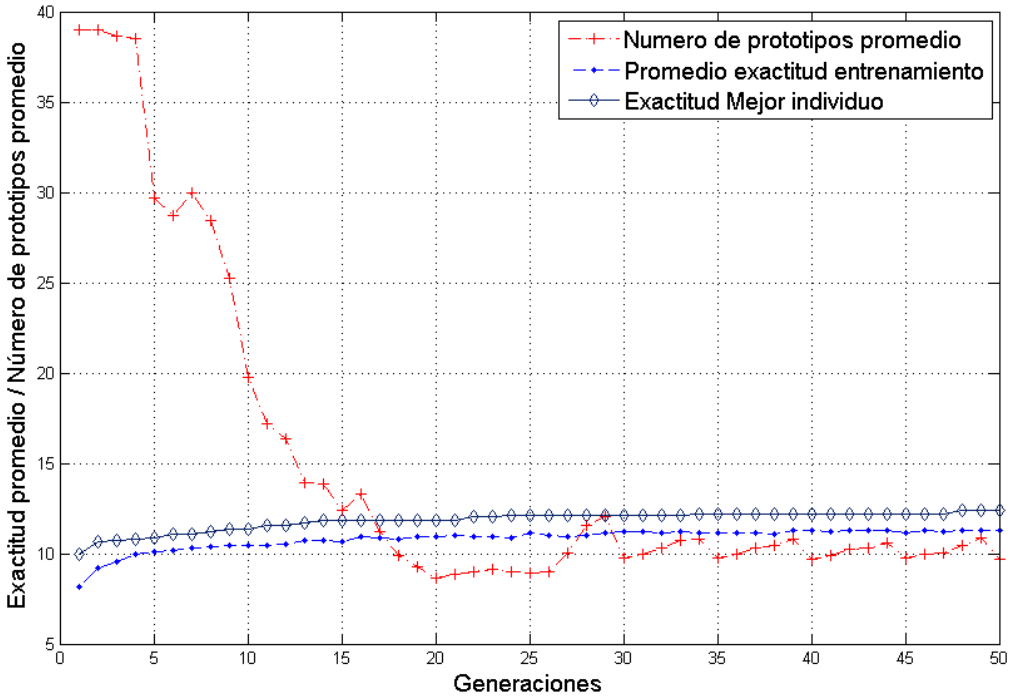


Figura 5.8: Se grafican las generaciones contra la exactitud promedio (multiplicada por 15) y el número de prototipos promedio. Además se grafica la exactitud del mejor individuo encontrado. Estos resultados son para una corrida del programa genético para el conjunto de datos *Monk*

En esta figura podemos apreciar cómo se incrementa la exactitud de forma promedio mientras pasan las generaciones. Luego en promedio conforme pasan las generaciones los individuos se vuelven mas aptos para su medio y aunque en las primeras generaciones se logra encontrar un individuo bastante apto, durante las

siguientes generaciones se siguen realizando búsquedas para tratar mejorar a los individuos. En esta figura también se puede apreciar como una reducción significativa en el número de prototipos no afecta ni a la exactitud promedio ni a la del mejor individuo, con lo que se muestra lo efectiva de la estrategia *tala* al permitir reducir de forma considerable el número de prototipos sin afectar la capacidad de encontrar mejores soluciones.

Como se menciono durante el capitulo 4 cuando se presento la función de aptitud esta función no tiene ningún componente que le haga preferir entre un individuo con mayor o menor número de prototipos siempre y cuando ambos tengan el mismo valor de aptitud. Esta función fue pensada para reducir los efectos del sobreajuste que son característicos de los métodos catalogados como *Mixed* y no para intentar reducir el número de prototipos. Luego una preocupación cuando los operadores genéticos le dan al programa genético la oportunidad de modificar el número del prototipos en cada individuo es que éste solo intente incrementar el número de prototipos por individuo sin analizar que se podrían obtener los mismo o mejores resultados disminuyendo el número de prototipos. Por esta razón es interesante ver que durante las generaciones 10 a 15 se puede apreciar como los operadores genéticos también ayudan a contribuir a la reducción del número de prototipos, esta reducción es una combinación de los operadores genéticos y el uso de la medida incompetencia. Mientras que la incompetencia se encarga de detectar aquellos prototipos en cada individuo que no proporcionan ningún aporte o que perjudican el factor de clasificación del programa genético, son los operadores los que al tener preferencia sobre estos prototipos se desasen de ellos o posicionan nuevos prototipos en lugares que hacen que los prototipos contraproducentes no realicen ningún aporte, con lo que serán eliminados en la siguiente corrida de la estrategia *tala*.

5.3.2 Conjuntos grandes

En esta seeción se describen los experimentos y resultados obtenidos para los conjuntos considerados como grandes que se presentaron en la tabla 5.3.

Debido a problemas de tiempo, varios algoritmos no se pueden correr para los conjuntos grandes, estos son *PNN*, *MCA*, *GMCA*, *ICPL2* y *POC* pues son técnicas demasiado lentas y su tiempo de ejecución aumenta rápidamente al aumentar el número de instancias. En la tabla 5.15 se resumen los resultados obtenidos por nuestro programa genético, así como para aquellos métodos que no presentaron problemas de tiempo de ejecución en los conjuntos de datos considerados como grandes. En esta tabla se muestran los resultados obtenidos para la reducción, exactitud en entrenamiento y exactitud en el conjunto de prueba. Para cada una de estas categorías se ordenan los métodos de acuerdo a su desempeño.

	Reducción		Exactitud entrenamiento		Exactitud prueba
PSCSA	0.9988	ENPC	0.8809	GENN	0.8133
AVQ	0.998	GENN	0.8428	1NN	0.806
LVQTC	0.9975	Depur	0.825	ENPC	0.8029
MSE	0.9936	PSO	0.8158	Depur	0.8004
PG	**0.9871**	1NN	0.8057	PSO	0.8
SGP	0.9823	RSP3	0.7922	MSE	0.7674
BTS3	0.9801	HYB	0.7888	Chen	0.7621
Mixtgauss	0.9801	MSE	0.7759	HYB	0.7618
LVQPRU	0.9801	Chen	0.7682	RSP3	0.7556
Chen	0.9801	AMPSO	0.7436	AMPSO	0.741
LVQ3	0.9799	PG	**0.7421**	BTS3	0.7399
DSM	0.9799	BTS3	0.7393	PG	**0.7397**
VQ	0.9799	LVQPRU	0.7373	LVQPRU	0.7356
PSO	0.9799	DSM	0.7353	DSM	0.7341
AMPSO	0.9797	MixtGauss	0.7345	MixtGauss	0.7318
ENPC	0.8205	LVQ3	0.734	LVQ3	0.7318
RSP3	0.81	VQ	0.7322	VQ	0.7316
HYB	0.5727	LVQTC	0.7065	LVQTC	0.7056
Depur	0.2708	PSCSA	0.673	PSCSA	0.6707
GENN	0.1576	AVQ	0.6546	AVQ	0.6518
1NN	0	SGP	0.6162	SGP	0.6086

Tabla 5.15: Resultados para reducción de la cardinalidad del conjunto de prototipos promedio, la exactitud promedio en el conjunto de entrenamiento y prueba para los conjuntos grandes. Para cada columna se ordena a los métodos de acuerdo a su desempeño.

Aunque para este conjunto de datos el programa genético se destaca en el área de reducción, los resultados obtenidos en la exactitud promedio lo posicionan en un lugar mucho más bajo que en los resultados obtenidos para conjuntos pequeños.

Comparándolo contra *AMPSO* y *ENPC* (los dos métodos que se encuentran en la misma taxonomía que el programa genético desarrollado), vemos que nuestro método obtiene nuevamente mejores resultados en reducción que estos métodos, con *ENPC* obteniendo valores muy bajos, mientras que los obtenidos por *AMPSO* aunque menores no se alejan mucho. En cuanto a la exactitud en el conjunto de prueba los resultados obtenidos por *AMPSO* solo logran superar a nuestro método por un pequeño margen. Mientras que los resultados obtenidos por *ENPC* son mucho mejores, se puede apreciar que los resultados obtenidos por este método sufren de un mayor sobreajuste, teniendo 8 puntos porcentuales en la exactitud en el conjunto de entrenamiento sobre la obtenida en el conjunto de prueba.

La diferencia que se obtiene en el nivel de exactitud en el conjunto de prueba de los conjuntos grandes contra los pequeños no es muy sorprendente si se consideran los resultados obtenidos por *Crammer et. al*, quienes prueban un teorema con respecto a usar clasificación basada en prototipos y una función de aptitud basada en el margen. La prueba de este teorema trae consigo unas cuantas observaciones. Primero que el error de clasificación es independiente de la dimencionalidad de los datos i.e. el número de atributos. Segundo, al aumentar el número de prototipos también se aumenta la dimensión de VapnikChervonenkis, lo cual sugiere que tener muchas instancias resultará en bajos niveles de clasificación (Crammer *et al.*, 2002). Como se menciono previamente nuestro programa genético se caracteriza por utilizar instancias en los nodos terminales, esta estrategia tiene la desventaja que al aumentar el número de instancias también aumenta el espacio de búsqueda, haciendo más difícil encontrar buenas soluciones.

En la figura 5.9, se muestra un gráfico de dispersión sobre la exactitud promedio y la reducción (en porcentaje) para todos los métodos, ademas en la imagen 5.10 se muestra un zoom de esta imagen para distinguir la posición del programa genético desarrollado en esta tesis.

En la tabla 5.16 se muestran los resultados para la exactitud en el conjunto de pruebas para los conjuntos grandes, se muestran los resultados obtenidos por nuestro

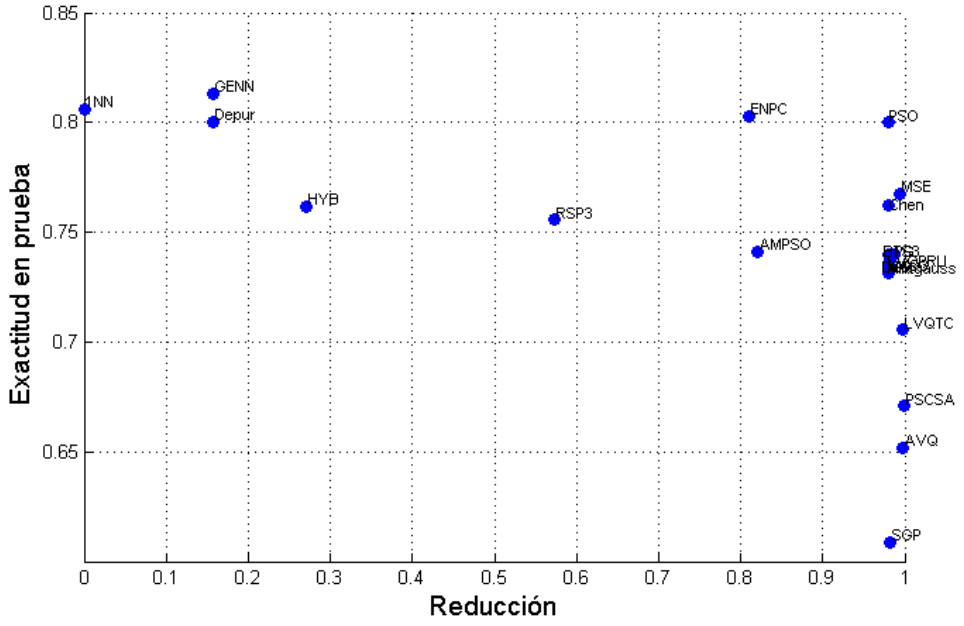


Figura 5.9: Gráfica de dispersión de exactitud en conjunto de prueba y reducción de cardinalidad para conjuntos pequeños.

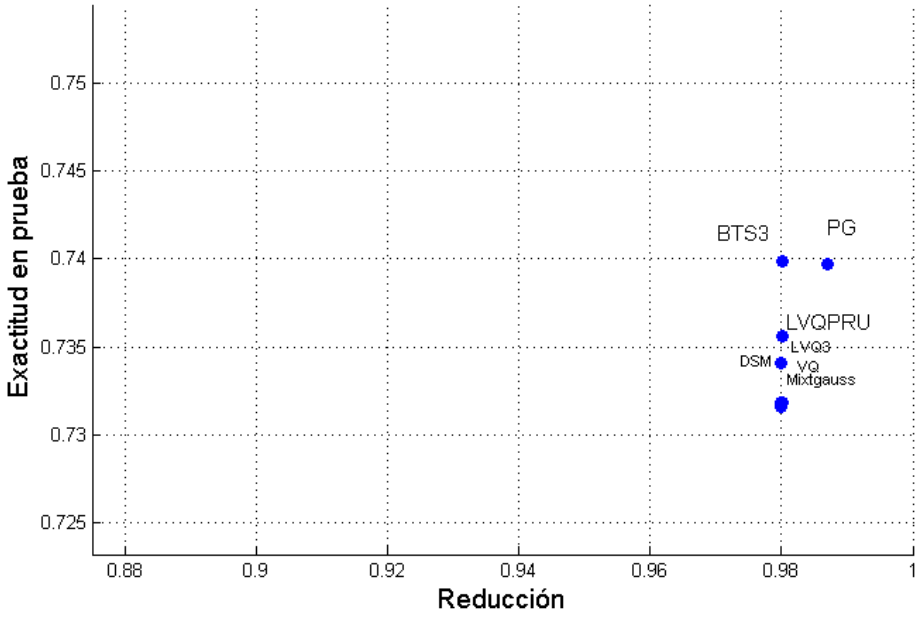


Figura 5.10: Gráfica de dispersión de exactitud en conjunto de prueba y reducción de cardinalidad para conjuntos pequeños. Realizando un zoom.

método, el método *1-NN* y el promedio obtenido por todos los otros métodos. En esta tabla se busca ilustrar que aunque nuestro programa genético no obtiene los mejores resultados para este tipo de conjuntos, se cumple con el objetivo de generación de prototipos, esto es, se logra encontrar un subconjunto de prototipos de tamaño mucho menor que el número de instancias en el conjunto de entrenamiento sin que este conjunto pierda en gran medida su poder de clasificación.
En los resultados presentados en la tabla 5.16 puede verse como para los conjuntos *banana, magic, nursery, pageblock, penbased, phoneme, ring, satimage, segment, spambase, texture, twonorm* el programa genético generado obtiene valores de exactitud en los datos de prueba que son menores a los obtenidos por el clasificador *1-NN*, esto es, usando el conjunto de prototipos generado (en lugar del conjunto de entrenamiento) un clasificador *1-NN* (como el usado por el programa genético) obtiene resultados de exactitud en los datos de prueba que son menores a los que se obtendrían si se usara el clasificador *1-NN* sobre el conjunto de entrenamiento original.

Para los conjuntos *coil200, marketing, thyroid* y *titanic* se logra conseguir además de una reducción significativa en el conjunto de entrenamiento, un incremento en el poder de clasificación bajo un enfoque *1-NN*, esto es, bajo este tipo de enfoque de clasificación si se usa el conjunto de prototipos generados en lugar del conjunto de datos de entrenamiento, se obtiene una exactitud mayor en los conjuntos de datos de prueba.

Esto nos dice que aunque en menor medida que en los conjuntos pequeños, el método desarrollado tiene la capacidad buscada en un generador de prototipos que es aumentar la exactitud para datos no vistos de un clasificador *1-NN* mientras el número de prototipos generados es mucho menor que el número de instancias en el conjunto de datos de entrenamiento. Más aun, para aquellos conjuntos en los que el programa genético obtuvo valores de exactitud en los conjuntos de prueba menores que los obtenidos por el clasificador *1-NN* se encuentran aquellos para los que el programa genético desarrollado obtiene mejores resultados en los valores de

exactitud en datos de prueba que los obtenidos por el promedio de los métodos, estos son *nursery, pageblock, penbased, phoneme* y *ring*. Hay que notar que en promedio para todos los conjuntos considerados como grandes descritos en la tabla 5.3 solo el algoritmo *GENN* logra superar los resultados obtenidos en exactitud en el conjunto de prueba obtenidos por el clasificador *1-NN*. Lo cual indica que los resultados obtenidos por el clasificador *1-NN* fueron en promedio mejor que todos los métodos sin contar a *GENN*. Con esto buscamos explicar que el programa genético logra el objetivo de la generación de prototipos para varios de los conjuntos en los que se realizaron pruebas, mientras que para los que no, lograr obtener resultados que son mejores que los obtenidos por el promedio de todos los métodos probados.

	Promedio	1-NN	GP
banana	0.8088	0.8751	0.66678
coil2000	0.8696	0.8963	0.92461
magic	0.7540	0.8059	0.73668
marketing	0.2606	0.2738	0.29412
nursery	0.5691	0.8267	0.64744
pageblock	0.8801	0.9576	0.95326
penbased	0.8701	0.9935	0.91144
phoneme	0.7916	0.8991	0.79349
ring	0.7063	0.7524	0.72801
satimage	0.8308	0.9058	0.78959
segment	0.8519	0.9662	0.62202
spambase	0.8226	0.8945	0.7566
texture	0.8814	0.9905	0.56709
thyroid	0.8512	0.9258	0.92844
titanic	0.7244	0.6075	0.78187
twonorm	0.9403	0.9468	0.807727

Tabla 5.16: Resultados para la exactitud en el conjunto de pruebas para los conjuntos grandes, se muestran los resultados obtenidos por nuestro método, el método *1-NN* y el promedio obtenido por todos los otros métodos.

Como se menciono en los resultados obtenidos para conjuntos pequeños es interesante analizar el comportamiento del número promedio de prototipos por individuo así como la exactitud promedio de sus individuos y el mejor individuo obtenido hasta el momento. En la figura 5.11 se presentan los resultados obtenidos para una corrida del programa genético sobre el conjunto de datos *chess*, en esta imagen se

muestra como varia tanto la exactitud promedio como el número de prototipos promedio para todos los individuos en cada generación y además se grafica la exactitud del mejor individuo encontrado. Notese que la exactitud fue multiplicada por un factor de 300 para que sus valores pudieran ser apreciados en la imagen.

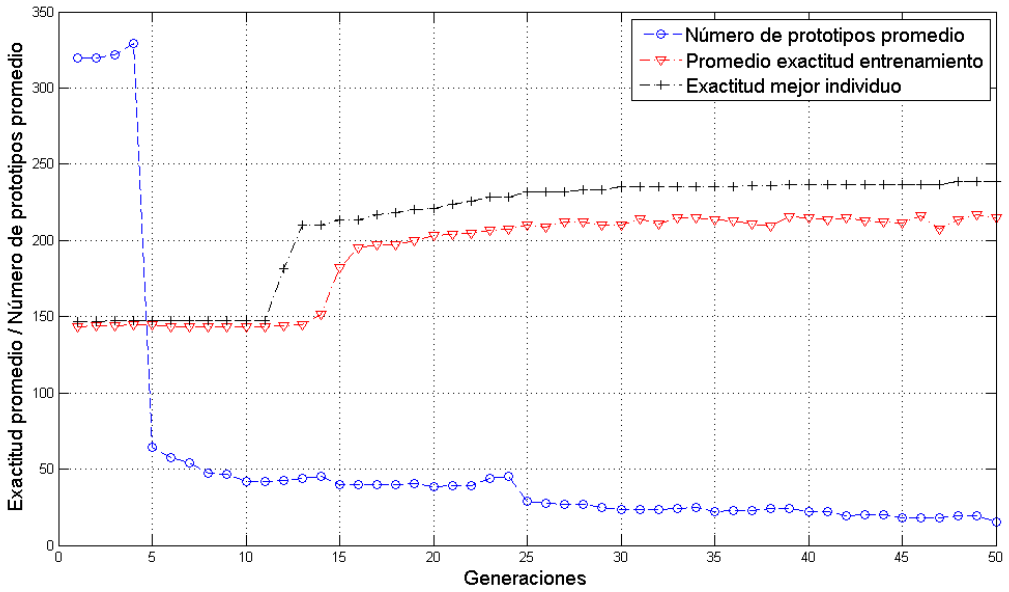


Figura 5.11: Se grafican las generaciones contra la exactitud promedio (multiplicada por 300) y el número de prototipos promedio. Además se grafica la exactitud del mejor individuo encontrado. Esto para el conjunto de datos *Chess*

En esta figura podemos ver el comportamiento de búsqueda del programa genético, mientras que durante las primeras 11 generaciones se realizo una búsqueda sin obtener grandes avances en la aptitud de sus individuos, es en las generaciones 11 a la 25 en que se crea un auge en la mejora de este valor. También se puede apreciar como el uso de la estrategia tala ayuda a reducir en gran medida el número de prototipos promedio por individuo. Note que aunque el número de prototipos iniciales fuera tan alto (320 por individuo), durante las generaciones iniciales que se contaba con este número tan elevado no se consiguió una mejora en la exactitud ni de los individuos en conjunto, ni del mejor individuo. Esto nos dice que aunque teóricamente el tener un mayor número de prototipos le permitiría al programa genético realizar una mejor clasificación, no es este número del que se dependa, si no de las

	ionosphe.	haberman survival	iris	cancer	diabetes	glass	breast	blood	derma.	Balanza
Neural	86.48	73.04	91.11	94.63	73.57	61.56	56.25	77.77	94.21	77.86
SVM	84.95	75.22	79.56	95.71	78.09	50.31	56.25	77.23	96.26	76.04
LB	91.81	64.57	91.56	95.80	76.17	57.81	60.63	64.11	95.70	77.86
RF	92.19	71.96	91.56	95.80	76.78	62.50	64.38	72.77	98.13	72.94
Naive	82.67	75.00	84.89	93.27	75.04	51.56	50.00	76.88	86.54	76.04
K-NN	85.90	65.87	90.67	94.93	70.09	61.88	63.75	67.05	93.83	64.17
PG	85.14	75.22	86.67	92.98	71.74	57.19	51.88	75.80	94.58	69.20

Tabla 5.17: Resultados para los métodos de clasificación comunes.

estructuras que se logren evolucionar dentro de cada prototipo. En otras palabras, no importa la cantidad, si no la calidad.

5.4 Comparación con métodos de clasificación tradicionales

Aunque el programa genético desarrollado en esta tesis no es un método de clasificación estándar, sino un método para la generación de prototipos de clasificación (i.e, tiene como objetivo implícito la descripción de los datos mediante un conjunto reducido de instancias, además de la capacidad de hacer clasificación), se decidió comparar el programa genético contra otros métodos del estado del arte y ampliamente usados de clasificación.

La experimentación se realizó usando los 10 conjuntos de datos descritos en la tabla 5.1. Los métodos usados para compararnos fueron Redes neuronales (*neural*), máquina de soporte vectorial (*SVM*), logitboost (*LB*), random forest (*RF*), clasificador Bayesiano simple (*Naive*) y *K-NN*. En la tabla 5.17 se resumen los resultados obtenidos para estos conjuntos, se muestra para cada método el valor promedio obtenido sobre los 5 pliegues para cada conjunto.

De estos resultados podemos ver como el programa genético desarrollado en esta tesis logra posicionarse como el mejor método para el conjunto de datos *Haberman Survival*, mientras que solo para el conjunto *cancer* se encuentra en ultimo lugar. Si se obtiene el promedio ponderado para cada método sobre todos los conjuntos obtenemos que el método desarrollado se posiciona en el quinto lugar.

Podemos ver que a pesar de que el principal objetivo del programa genético desarrollado es generar prototipos de clasificación que permitan describir el conjunto de datos con un conjunto pequeño de prototipos, el programa genético obtiene resultados comparables con métodos de clasificación del estado del arte y ampliamente usados. El programa genético no le gana a estos clasificadores usuales pero tiene como ventajas el crear un conjunto de prototipos de clasificación que es de un tamaño menor que el número de instancias en el conjunto de entrenamiento. Una de las ventajas de generar prototipos de clasificación es que estos se pueden usar para la interpretación de los datos. Por ejemplo, se puede decir que clase se suele confundir con otra (cuando los prototipos son muy parecidos), que tantas subclases puede contener una clase (analizando en número de prototipos generados por cada clase, entre más prototipos más subclases podrían estar presentes).

Para probar el nivel de clasificación que tienen los prototipos generados por nuestro programa genético se decidió probar cómo funcionaban los otros métodos utilizando para entrenarse, en lugar del conjunto de entrenamiento, el conjunto de prototipos generados. Esto es, se utilizó el conjunto de prototipos generado por el programa genético desarrollado para entrenar a los demás métodos, y se probó su exactitud en el conjunto de prueba. Cabe notar que apesar que es posible utilizar el conjunto de prototipos en lugar del conjunto de entrenamiento para todos los métodos, el conjunto de prototipos fue obtenido mediando un proceso de optimizacion en el que se usa un enfoque *1-NN* para clasificar las instancias. En las tablas 5.18 y 5.19 se presentan los resultados obtenidos para estos experimentos.

Observando los resultados obtenidos se puede ver que la clasificación obtenida por todos los métodos (excepto *K-NN*) utilizando el conjunto de prototipos gene-

	neural		SVN		LB	
	Entre	Proto	Entre	Proto	Entre	Proto
ionosphere	86.48	67.81	84.95	59.24	91.81	52.95
haberman survival	73.04	64.35	75.22	53.26	64.57	60.43
iris	91.11	74.67	79.56	60.89	91.56	71.56
cancer	94.63	81.46	95.71	44.29	95.80	85.95
diabetes	73.57	65.22	78.09	69.30	76.17	55.30
glass	61.56	48.13	50.31	41.56	57.81	30.63
breast	56.25	38.13	56.25	35.63	60.63	36.88
blood	77.77	65.71	77.23	70.45	64.11	66.61
dermatology	94.21	79.63	96.26	67.48	95.70	59.81
Balanza	77.86	62.57	76.04	73.90	77.86	72.62

Tabla 5.18: Resultados para los métodos *neural network*, *SVN*, *LB* utilizando el conjunto de entrenamiento o el conjunto de prototipos para entrenarse, la exactitud mostrada se obtiene sobre el conjunto de pruebas.

	RF		Naive		K-NN	
	Entre	Proto	Entre	Proto	Entre	Proto
ionosphere	92.19	51.43	82.67	49.33	85.90	85.14
haberman survival	71.96	67.39	75.00	59.57	65.87	75.22
iris	91.56	76.89	84.89	56.00	90.67	86.67
cancer	95.80	82.54	93.27	44.59	94.93	92.98
diabetes	76.78	61.48	75.04	63.30	70.09	71.74
glass	62.50	46.56	51.56	39.06	61.88	57.19
breast	64.38	46.88	50.00	32.50	63.75	51.88
blood	72.77	68.57	76.88	67.50	67.05	75.80
dermatology	98.13	74.95	86.54	72.34	93.83	94.58
Balanza	72.94	70.27	76.04	62.35	64.17	69.20

Tabla 5.19: Resultados para los métodos *RF*, *Naive Bayes*, *K-NN* utilizando el conjunto de entrenamiento o el conjunto de prototipos para entrenarse, la exactitud mostrada se obtiene sobre el conjunto de pruebas.

rados es en promedio menor que la obtenida cuando se usan todos los datos. Como se menciono este resultado no es del todo asombroso, puesto que el conjunto de prototipos esta diseñado bajo un enfoque *1-NN*. Esto hace que estos prototipos se posicionen de tal manera que se optimice el nivel de clasificación obtenido utilizando un enfoque *1-NN*. Es de destacar sin embargo que el método *K-NN* logra obtener un beneficio al utilizar el conjunto de prototipos sobre el conjunto de entrenamiento, esto con una reducción sustancial en el tiempo de ejecución y en la cantidad de memoria necesaria para correr este algoritmo.

En la figura 5.12 se presenta una grafica de dispersión para el promedio de exactitud obtenido en los 10 conjuntos de pruebas por el método *K-NN*. Se grafica la exactitud promedio cuando se utiliza el conjunto de entrenamiento para entrenar contra la exactitud promedio cuando se utiliza el conjunto de prototipos para entrenar. Además se grafica la linea $f(x) = x$ como referencia, aquellos puntos que caen por arriba de la linea son los que obtienen mejores resultados utilizando el conjunto de prototipos para entrenarse, mientras que los que caen por debajo obtienen un mejor resultado entrenándose con el conjunto de entrenamiento. Cabe notar que en promedio utilizar el conjunto de prototipos es mejor bajo el método *K-NN* que utilizando el conjunto de entrenamiento.

Sería interesante analizar que pasaría si en lugar de utilizar el enfoque *1-NN* (que es usado actualmente en este trabajo) se utilizara otro enfoque de los mencionados en esta sección, digamos *Naive Bayes* por mencionar alguno. Teóricamente la exactitud utilizando el conjunto de prototipos como forma de entrenamiento debería verse incrementada para este método (*Naive Bayes*). Pues, se realiza una optimizacion de la posición de los prototipos utilizando este enfoque para maximizar la exactitud obtenida por este enfoque.

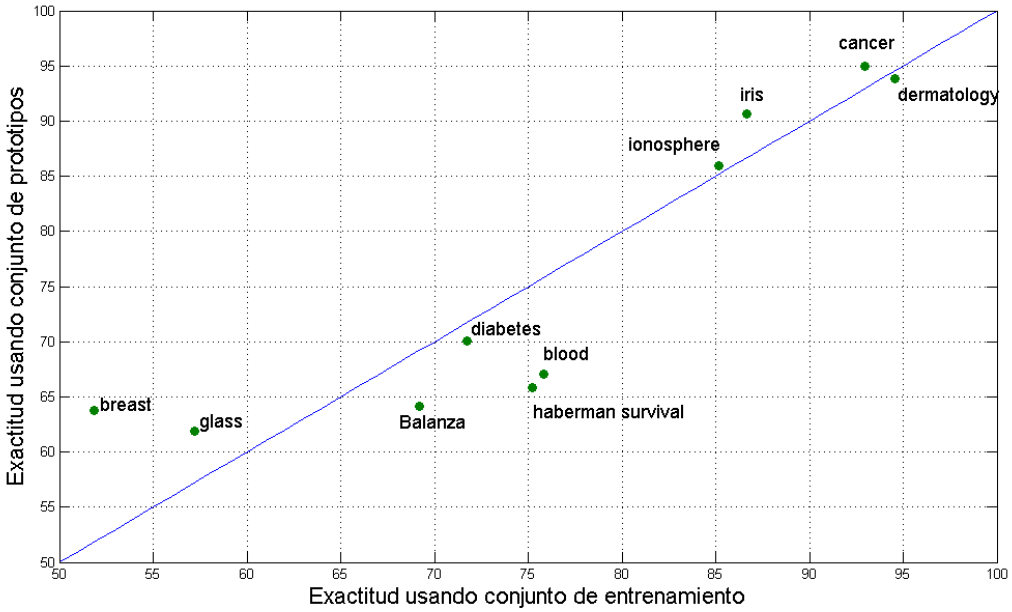


Figura 5.12: Gráfica de dispersión para el promedio de exactitud obtenido en los 10 conjuntos de pruebas por el método *K-NN* utilizando el conjunto de entrenamiento o prototipos para entrenarse.

5.5 Experimentación en atribución de autoría

A diferencia de otros métodos para clasificación usando PG, el programa genético desarrollado en esta tesis no se ve limitado por la dimension de las instancias, es por esto que aunque el método desarrollado no tenga ninguna herramienta especialmente diseñada para conjuntos de alta dimencionalidad se desea probar su rendimiento en este tipo de conjuntos de datos, donde, según nuestro conocimiento, no se han aplicado métodos de clasificación basados en programación genética.

Luego se desea probar al programa genético desarrollado en esta tesis frente a un problema de clasificación de alta dimencionalidad, como lo es la atribución de autoria. En la tabla 5.20 se presentan los resultados obtenidos en estos experimentos para la exactitud en la etapa de prueba.

Puede verse que los resultados obtenidos por el programa genético desarrollado en esta tesis no son tan buenos como los obtenidos por otros clasificadores, pero al

	Naive	SVM	Neural	RF	1-NN	PG
NFL	88.89	91.11	91.11	84.44	77.78	67.56
Business	82.22	81.11	77.78	71.11	50.00	50.44
Travel	70.00	75.00	74.00	75.00	55.00	60.33
Cricket	93.33	98.33	90.00	83.33	70.00	73.33
Poetry	74.55	60.00	65.45	40.00	27.27	52.73
CCAT	73.60	79.00	76.80	73.00	63.60	57.56
Twitter	60.80	52.60	58.20	N/A	26.40	30.50
promedio	77.63	76.74	76.19	71.15	52.86	56.06

Tabla 5.20: Exactitud promedio sobre 5 corridas para los conjuntos de clasificación de autoría.

menos se puede aplicar el programa genético para este tipo de datos y su desempeño no esta tan lejos de lo obtenido con clasificadores más especializados en este tipo de problemas. No obstante lo anterior, hemos podido corroborar que el programa genético puede aplicarse a datos de gran dimensionalidad. Dejamos para trabajo futuro el dotar al programa genético de capacidades para competir con otros algoritmos de clasificación en dominios con alta dimensionalidad.
Es de especial interés notar que se obtienen mejores resultados en 5 los 7 conjuntos que los obtenidos por el clasificador *1-NN*. Esto nos dice que si consideramos la tarea de generación de prototipos el programa genético desarrollado logra obtener una reducción considerable en el conjunto de prototipos, y además estos prototipos logran mejorar el nivel de clasificación que se obtendría al utilizar un clasificador basado en *1-NN*.

Los porcentajes de reducción de prototipos generados contra tamaño del conjunto de entrenamiento para cada conjunto es presentado en la tabla 5.21. De los resultados obtenidos en la tabla 5.21 vemos que para 4 de los 7 conjuntos de datos, se obtiene una reducción mayor de la cantidad inicial de prototipos que usa el programa (10 % del total del conjunto de entrenamiento), mientras que para los restantes 3 conjuntos de datos el incremento no es mayor del 1.3 %.

En la figura 5.13 se muestra una gráfica de como varía el número promedio

Conjuntos	**Instancias de entrenamiento**	**Prototipos**	**Reducción**	**Exactitud entrenamiento**
Poetry	145	11.75	91.89 %	96.34 %
Travel	112	10.54	90.58 %	83.79 %
Cricket	98	11.07	88.70 %	81.19 %
CCAT	500	29.19	94.16 %	70.00 %
Twitter	4500	242.88	94.60 %	37.26 %
NFL	52	7.95	84.71 %	89.25 %
Business	85	9.55	88.76 %	77.22 %

Tabla 5.21: Porcentajes de reducción de prototipos contra instancias del conjunto de entrenamiento.

de prototipos por individuos, en una corrida de nuestro programa genético para el conjunto de datos *twitter*, en esta figura se puede apreciar como el uso de la tala ayuda a remover prototipos sin utilidad (generación 5), a su vez se aprecia claramente como los operadores modifican estas cantidades en menor medida (generación 35 a 45). En esta corrida los individuos tenían una cantidad promedio de 450 prototipos (9 prototipos por clase) y se termina con un promedio de 241.52 (4.83 prototipos por clase).

En la figura 5.14 se muestra como varía la aptitud para esta misma corrida al pasar las generaciones, podemos ver que el PG evoluciona individuos más aptos con una menor cantidad de prototipos con ayuda de la tala y el uso de los operadores genéticos. Este resultado no debe de extrañar, pues al ser los prototipos inicializados al azar pueden caer éstos en malas posiciones donde clasifiquen instancias erróneamente, dando como resultado que eliminar estos prototipos traerá un beneficio directo al individuo. Nótese que la *tala* no eliminará estos prototipos pues aunque lo hagan de forma incorrecta son prototipos activos (predicen instancias), esto permite que el operador *mitosis* use estos prototipos para generar nuevos prototipos con la clase correcta, después del paso de generaciones este prototipo que inicialmente solo perjudicaba a nuestro individuo ayudará a generar prototipos en

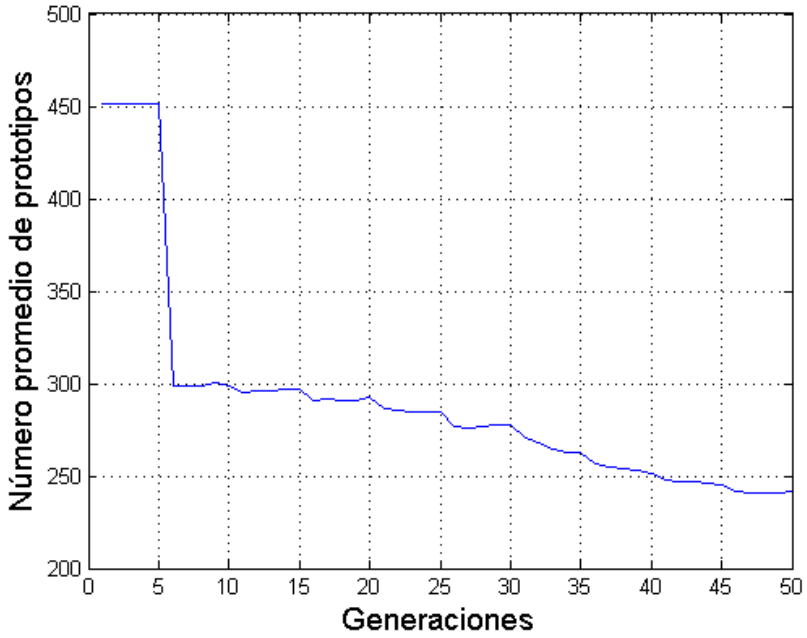


Figura 5.13: Gráfica del número de prototipos promedio por individuo al pasar las generaciones para el conjunto *twitter*.

posiciones estratégicas y dejara de ser activo, con lo cual el operador tala lo eliminará del individuo.

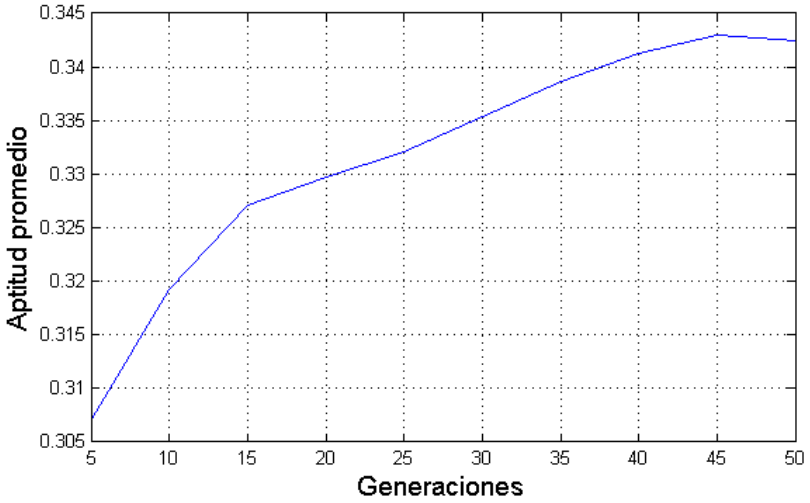


Figura 5.14: Gráfica del número de la aptitud promedio por individuo al pasar las generaciones para el conjunto *twitter*.

Capítulo 6

Conclusiones y trabajo futuro

En este trabajo se abordó el problema de generación de prototipos de clasificación. En este problema se busca generar un subconjunto de prototipos que mediante un clasificador *K-NN* se realice la clasificación de las instancias. Se busca primordialmente dos cosas, la primera obtener un buen nivel de clasificación en el conjunto de prueba, y la segunda, reducir a la menor cantidad posible el número de prototipos en el conjunto generado. Este método se trata de un programa genético en el que cada individuo es un conjunto de árboles. Estos árboles tienen la peculiaridad de se utilizan instancias del conjunto de entrenamiento en los nodos terminales, los nodos no-terminales se componen de funciones que operan sobre las instancias, la función de aptitud usada en este trabajo se define como:

$$Aptitud = Exactitud + Margen$$

Esta función busca encontrar prototipos que tengan buenos resultados en el conjunto de entrenamiento sin caer en el sobreajuste. El programa genético desarrollado se probo en la tarea de generación de prototipos, en la tarea de clasificación contra métodos tradicionales y por ultimo en la tarea de atribución de autoria. Los resultados muestran un resultado competitivo en la tarea de generación de prototipos cuando se trabaja con conjuntos pequeños, se obtienen resultados comparables a los de otros métodos asociados al estado del arte cuando se consideran conjuntos grandes. Un desempeño similar a métodos tradicionales en el área de clasificación y resultados que muestran que aunque es posible utilizarse en problemas de alta dimensionalidad, se ocuparía desarrollar herramientas especiales para poder obtener

resultados competitivos. En estos 3 experimentos se obtuvieron resultados que en general logran incrementar la exactitud en los datos de prueba que obtiene un clasificador *K-NN* a la vez que el número de prototipos generados es mucho menor que el número de instancias en el conjunto de entrenamiento.

6.1 Conclusiones

De manera general, podemos resumir el trabajo realizado en los siguientes puntos:

1. Se desarrolló e implementó un nuevo operador genético (mitosis) diseñado para la generación de prototipos.
2. Se desarrolló e implementó un operador generacional para evitar los problemas de sobrepoblación que a la vez no afecta el nivel de clasificación.
3. Se desarrolló e implementó un programa genético utilizando instancias en los nodos para obtener un método de generación de prototipos de clasificación automático.
4. Se probó que utilizar instancias en los nodos terminales le permite a métodos basados en PG utilizarse en conjuntos con un alta dimencionalidad, aunque faltaría analizar la mejora en los resultados obtenidos utilizando herramientas especialmente diseñadas para este tipo de conjuntos.
5. Se demostró que los operadores genéticos utilizados, en conjunto con la medida *incompetencia* tienen la capacidad de buscar mejores individuos reduciendo el número de prototipos en estos. Aunque en ciertos casos solo estas herramientas no eran suficientes.
6. Se mostró que utilizando un enfoque de PG para la generación de prototipos, se logran obtener resultados comparables a los de métodos de clasificación tradicionales.
7. Se probó que la PG puede usarse con efectividad en la tarea de generación de prototipos.

Con estos puntos se han cumplido los objetivos de la tesis y probado la hipótesis planteada.

En la sección 4.2 se dio una descripción del operador genético diseñado e implementado en esta tesis, el estudio realizado con las probabilidades de los clasificadores (sección 5.2.3) 5.2.1 mostró que el uso de este operador ayuda a la obtención de mejores individuos. Este operador permite al programa genético utilizar una especie de búsqueda voraz, permitiendo explorar en mayor profundidad individuos con buena aptitud. En combinación con los operadores de cruza, es un potente operador, pues suponga que se tenga una buena solución a la que al utilizar el operador cruza se le agregaran prototipos bien posicionados y uno mal posicionado de tal manera que disminuya la aptitud de dicha solución. En estos casos los operadores de cruza no tienen una forma de deshacerse de solo este prototipo que no sea al azar, y aun se corre el riesgo de remover buenos prototipos. El operador mitosis en estos casos tiene la ventaja de enfocarse en estos prototipos mal posicionados desde un principio sin modificar los demás prototipos de los cuales se beneficia dicha solución. Además, al dar preferencia a modificar prototipos con la mayor cantidad de instancias clasificadas incorrectamente, esto permite que la creación de nuevos prototipos a través de este operador sea de forma inteligente en aquellos lugares donde incrementará en mayor medida el desempeño del programa genético. Este operador a su vez no permite la creación de nuevos prototipos para grupos pequeños de instancias, pues si esto no se evita se podría llevar a crear un prototipo por cada instancia, cayendo en el sobreajuste y anulando la capacidad de reducción del método. Concluimos que el operador genético desarrollado en esta tesis es una herramienta que beneficia en gran medida al programa genético, ayudando a generar prototipos en lugares inteligentes del espacio de atributos, ademas de no incrementar el número de prototipos por individuo si no es necesario.

En esta tesis se utilizo una función de aptitud basada en la exactitud y el margen. Se utilizo la exactitud pues es esta la cantidad que en ultima instancia se busca maximizar, y el margen para evitar los problemas de sobreajuste. Durante toda la experimentación se obtuvieron resultados que confirman la buena elección de esta función. Otro gran aporte que trae el uso del margen como parte de la función de aptitud es que al permitir a nuestros individuos generar e intercambiar

cantidades aleatorias de prototipos se abre la puerta a individuos que tengan gran cantidad de prototipos, por ejemplo, se permiten individuos en el que todos los prototipo clasifiquen a una instancia (i.e. para cada instancia el individuo evoluciona un prototipo), este es un claro ejemplo de sobreajuste y un problema que puede afectar gravemente a la metodología desarrollada en esta tesis. Este ejemplo se hace evidente cuando solo se considera la exactitud como función de aptitud, pues un individuo con estas características obtiene una exactitud de 100 %. Mientras que al combinar el margen este tipo de casos es muy raro de encontrar. Durante toda la experimentación realizada no se presento ningún individuo con estas características, esto nos dice que la función de aptitud empleada realiza un buen trabajo al no permitir este tipo de sobreajuste.

Otra estrategia desarrollada para reducir el número de prototipos generados fue la *tala*, los estudios realizados en la sección 5.2.4 nos hacen concluir que esta metodología es eficaz en la tarea reducir el número de prototipos generados sin que esto afecte el desempeño del programa genético. Se mostró además que el incremento en la aptitud de los individuos no se ve reducido al tener a la mano un menor número de prototipos. Si no que depende del posicionamiento inteligente de estos. También se mostró que la reducción del número de prototipos no solamente se debe a la estrategia *tala* si no que los operadores genéticos también ayudan a la reducción de esta cantidad. Se mostró que apesar de que la función de aptitud no tiene ningún componente que le haga preferir entre un individuo con mayor o menor número de prototipos siempre y cuando ambos tengan el mismo valor de aptitud. El uso de la incompetencia junto con los operadores genéticos logra realizar reducciones en el número de prototipos generados, pues mientras la incompetencia se encarga de detectar aquellos prototipos en cada individuo que no proporcionan ningún aporte o que perjudican el factor de clasificación del programa genético, son los operadores los que al tener preferencia sobre estos prototipos se desasen de ellos o posicionan nuevos prototipos en lugares que hacen que los prototipos contraproducentes no realicen ningún aporte, con lo que serán eliminados en la siguiente corrida de la estrategia *tala*.

Durante la fase experimental se pudo comprobar que el programa genético desarrollado logra posicionarse de manera competitiva entre métodos tanto del estado del arte así como los que se usan de forma amplia en la tarea de generación de prototipos. Para la experimentación en conjuntos pequeños se destaco como uno de los mejores algoritmos, para los métodos en su misma taxonomia fue el mejor posicionado por un gran margen. Para este tipo de conjuntos se logro obtener resultados competitivos tanto en el área de exactitud en el conjunto de prueba como en la reducción en el conjunto de prototipos generados.
Mientras que para los conjuntos grandes de generación de prototipos no se logra destacar como uno de los mejores métodos, si se logra obtener una buena posición. Como se mostró en esta fase de la experimentación, se logra obtener un buen desempeño como un método de generación de prototipos al obtener una de las mejores reducciones de entre todos los métodos y además aumentar el nivel de clasificación que obtendría un clasificador basado en *K-NN*.

Este punto se ve reforzado durante las pruebas realizadas contra los métodos de clasificación tradicionales, los que no buscan reducir el tamaño del conjunto de entrenamiento. En estas pruebas se presentan resultados obtenidos por un clasificador *K-NN* tanto usando el conjunto de entrenamiento como usando solamente el conjunto de prototipos. Estos resultados muestran que el conjunto de prototipos generados por nuestro programa genético tiene la capacidad de mejorar el nivel de clasificación obtenido por un clasificador *K-NN*. Concluimos también en base a los resultados obtenidos que a pesar de que el principal objetivo de nuestro programa genético es generar prototipos que permitan describir el conjunto de datos con un conjunto pequeño de instancias, el programa genético obtiene resultados comparables con métodos de clasificación del estado del arte y ampliamente usados.

De los resultados obtenidos para atribución de autoria podemos concluir que aunque nuestro programa genético tenga la capacidad de utilizarse para este tipo de conjuntos no se logra obtener resultados competitivos con respecto a otros métodos de clasificación mas especializados para este tipo de problemas. Cabría resaltar que

aun en este tipo de conjuntos de datos el programa genético logra obtener un alto nivel de reducción. Además que logra posicionarse por arriba de un método basado en *1-NN* para 5 de los 7 conjuntos probados. Esto nos dice que bajo un enfoque *1-NN* los prototipos generados por nuestro método obtienen una caracterización igual o mayor que la que se obtiene al utilizar las instancias en el conjunto de entrenamiento.

6.2 Trabajo futuro

Es posible plantear algunos puntos que representen áreas de estudio abiertas relacionadas al trabajo realizado, por ejemplo:

1. La metodología propuesta puede ser adaptada a un algoritmo evolutivo incluyendo los operadores desarrollados, en este caso los individuos serian conjuntos de prototipos, de esta manera el espacio de búsqueda sería el espacio de atributos. Se tendría como ventaja una mayor velocidad del método al tener directamente los prototipos como individuos, pero un mayor uso de memoria en comparación con la versión de PG.

2. La inicialización de los árboles es hecha completamente al azar; sería interesante analizar cómo afecta al desempeño del algoritmo el inicializar árboles cuyo prototipo sea el centroide de los datos, pues esta forma de clasificación ha mostrado buenos resultados.

3. Comparar el desempeño del programa genético cuando se usan diferentes medidas de aptitud.

4. Estudiar el impacto que tiene el número de instancias en el conjunto de entrenamiento en el poder de predicción del método propuesto.

5. Realizar una comparación extensa del programa genético contra métodos de clasi-

ficación basados en instancias (e.g., *K-NN* y sus variantes principales) en problemas apropiados para métodos de clasificación basados en similitud.

Bibliografía

Alcalá-Fdez, J., A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez y F. Herrera (2011), «KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework», *Multiple-Valued Logic and Soft Computing*, **17**, págs. 255–287.

Aurenhammer, F. (1991), «Voronoi diagrams a survey of a fundamental geometric data structure», *ACM Comput. Surv.*, **23**(3), págs. 345–405.

Beyer, H. G. (2001), *The theory of evolution strategies*, Springer.

Bishop, C. (2006), *Pattern recognition and machine learning*, Information science and statistics, Springer.

Bojarczuk, C. C., H. S. Lopes y A. A. Freitas (2000), «Genetic programming for knowledge discovery in chest-pain diagnosis», *Engineering in Medicine and Biology Magazine, IEEE*, **19**(4), págs. 38–44.

Bojarczuk, C. C., A. S. D. Setembro, H. S. Lopes, A. A. Freitas y R. I. Conceio (1999), «Discovering comprehensible classification rules using genetic programming: a case study in a medical domain», en W. Banzhaf y J. D. et al (editores), *Proc Genetic and Evolutionary Computation Conference (GECCO-99)*, Morgan Kaufmann, Orlando USA, págs. 953–958.

Borgelt, C. (2005), *Prototype-based classification and clustering*, Tesis Doctoral, Otto-von-Guericke-Universität.

CALLEJA, J. y O. FUENTES (2004), «Machine learning and image analysis for morphological galaxy classification», *Montly Notices of the Royal Astronomical Society*, **349**, págs. 87–93.

CARRERAS, X., L. S. MARQUEZ y J. G. SALGADO (2001), «Boosting trees for anti-spam email filtering», en *In Proceedings of RANLP-01, 4th International Conference on Recent Advances in Natural Language Processing, Tzigov Chark, BG*, págs. 58–64.

CHANG, C.-L. (1974), «Finding prototypes for nearest neighbor classifiers», *IEEE Trans. Comput.*, **23**(11), págs. 1179–1184.

CLARK, P. y T. NIBLETT (1989), «The CN2 induction algorithm», *Mach. Learn.*, **3**(4), págs. 261–283.

CORDELLA, L. P., C. D. STEFANO, F. FONTANELLA y A. MARCELLI (2006a), «Evolutionary generation of prototypes for a learning vector quantization classifier.», en *EvoWorkshops*, *Lecture Notes in Computer Science*, tomo 3907, Springer, págs. 391–402.

CORDELLA, L. P., C. D. STEFANO, F. FONTANELLA y A. MARCELLI (2006b), *Looking for prototypes by genetic programming.*, *Lecture Notes in Computer Science*, tomo 4153, Springer, págs. 152–159.

CRAMMER, K., R. GILAD-BACHRACH, A. NAVOT y N. TISHBY (2002), «Margin analysis of the LVQ algorithm», en *In: Advances in Neural Information Processing Systems 2002*, MIT press, págs. 462–469.

DE JONG, K. (2008), «Evolutionary computation: a unified approach», en *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, ACM, págs. 2245–2258.

DEB, K. y S. AGRAWAL (1998), «Understanding interactions among genetic algorithm parameters», en *in Foundations of Genetic Algorithms 5*, tomo 5, págs. 265–286.

DUIN, R. P. W. y P. PACLÍK (2006), «Prototype selection for dissimilarity-based classifiers», *Pattern Recognition*, **39**, págs. 189–208.

EGGERMONT, J., A. E. EIBEN y J. I. V. HEMERT (1999), «Adapting the fitness function in GP for data mining.», en *EuroGP*, *Lecture Notes in Computer Science*, tomo 1598, Springer, págs. 193–202.

EGGERMONT, J., J. N. KOK y W. A. KOSTERS (2004), «Genetic programming for data classification: partitioning the search space», en *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, ACM, New York, NY, USA, págs. 1001–1005.

EIBEN, A. E. y J. E. SMITH (2008), *Introduction to evolutionary computing (natural computing series)*, Springer.

ESPEJO, P. G., S. VENTURA y F. HERRERA (2010), «A survey on the application of genetic programming to classification», *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **40**(2), págs. 121–144.

FABER, V. (1994), «Clustering and the continuous k-means algorithm», *Science*, **22**, págs. 138–144.

FARAOUN, K. y A. BOUKELIF (2000), «Genetic programming approach for multi-category pattern classification applied to network intrusions detection», *International Journal of Computational Intelligence*, págs. 79–90.

FERNÁNDEZ, F. y P. ISASI (2004), «Evolutionary design of nearest prototype classifiers», *Journal of Heuristics*, **10**(4), págs. 431–454.

FRANK, A. y A. ASUNCION (2010), «UCI machine learning repository», .

GUYON, I., G. CAWLEY, G. DROR y A. SAFFARI (2010), «Hands-on pattern recognition: challenges in data representation, model selection, and performance prediction», Microtome Publishing.

Han, E.-H. S. y G. Karypis (2000), «Centroid-based document classification: analysis and experimental results.», en *PKDD*, *Lecture Notes in Computer Science*, tomo 1910, Springer, págs. 424–431.

Hastie, T., T. Robert y F. Jerome (2009), *The elements of statistical learning*, Springer-Verlag.

Iba, H., Y. Hasegawa y T. K. Paul (2009), *Applied genetic programming and machine learning*, primera edición, CRC Press, Inc., Boca Raton, FL, USA.

Jabeen, H. y A. R. Baig (2010), «Review of classification using genetic programming», *International Journal of Engineering Science and Technology*, **2**, págs. 94–103.

Kohonen, T. (1989), *Self-organization and associative memory*, capítulo 5, Springer, Berlin; New York.

Koza, J. (1990), «Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems», *Technical report*, Dept. of Computer Science, Stanford University.

Koza, J. R. (1992), *Genetic programming: on the programming of computers by means of natural selection*, MIT Press, Cambridge, MA, USA.

Lloyd, S. P. (1982), «Least squares quantization in pcm», *IEEE Transactions on Information Theory*, **28**, págs. 129–137.

Luke, S. y L. Panait (2001), «A survey and comparison of tree generation algorithms», en *Proceedings of the Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, págs. 81–88.

Muni, D. P., N. R. Pal y J. Das (2004), «A novel approach to design classifiers using genetic programming», *IEEE Transactions on Evolutionary Computation*, **8**(2), págs. 183–196.

NANNI, L. y A. LUMINI (2009), «Particle swarm optimization for prototype reduction», *Neurocomput.*, **72**(4-6), págs. 1092–1097.

NORTON, S. W. (1989), «Generating better decision trees», en *Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, págs. 800–805.

NÚÑEZ, M. (1991), «The use of background knowledge in decision tree induction», *Mach. Learn.*, **6**, págs. 231–250.

POLI, R., W. B. LANGDON y N. F. MCPHEE (2008), *A field guide to genetic programming*, Lulu Enterprises, UK Ltd.

QUINLAN, J. R. (1993), *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

RAUSS, P. J., J. M. DAIDA y S. A. CHAUDHARY (2000), «Classification of spectral imagery using genetic programming», Morgan Kaufmann, págs. 726–733.

ROSCH, E. (1975), «Cognitive representations of semantic categories», *Journal of Experimental Psychology: General*, **104**(3), págs. 192–233.

ROUWHORST, S., A. ENGELBRECHT y A. ENGELBRECHT (2000), «Searching the forest: using decision trees as building blocks for evolutionary search in classification databases», en *Proceedings of the 2000 Congress on Evolutionary Computation*, págs. 633–638, IEEE.

SILVA, S. y Y.-T. TSENG (2008), «Classification of seafloor habitats using genetic programming», en *Proceedings of the 2008 conference on Applications of evolutionary computing*, EVO'08, Springer-Verlag, Berlin, Heidelberg, págs. 315–324.

STAMATATOS, E. (2009), «A survey of modern authorship attribution methods», *J. Am. Soc. Inf. Sci. Technol.*, **60**, págs. 538–556.

TAN, M. (1993), «Cost-sensitive learning of classification knowledge and its applications in robotics», *Mach. Learn.*, **13**, págs. 7–33.

Tan, S., Y. Wang y G. Wu (2011), «Adapting centroid classifier for document categorization», *Expert Syst. Appl.*, **38**, págs. 10 264–10 273.

Taskar, B., C. Guestrin y D. Koller (2003), «Max-margin markov networks», en *Advances in neural information processing systems 16: proceedings of the 2003 conference*, MIT Press.

Triguero, I., J. Derrac, S. García y F. Herrera (2012), «A taxonomy and experimental study on prototype generation for nearest neighbor classification», *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, **42**(1), págs. 86–100.

Turney, P. D. (1995), «Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm», *Journal of Artificial Intelligence Research*, págs. 369–409.

Zongker, D. y W. Punch (1995), «lilgp», .

Ficha autobiográfica

Karlo Mario Mendoza Mendoza

Candidato para el grado de Maestro en Ciencias en Ingeniería de Sistemas
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

Programación genética para la generación automática de prototipos de clasificación

Nací en la ciudad de La Paz el 2 de diciembre de 1987. Soy el tercer hijo del Sr. Juan Pablo Mendoza Moreno y la Sra. Gloria Amanda Mendoza Cota. Curse la licenciatura en la Facultad de Ciencias Físico Matemáticas de la Universidad Autónoma de Nuevo León obteniendo el grado de Licenciado en Matemáticas en el 2010. Posteriormente ingrese al Posgrado en Ingeniería de Sistemas de la Universidad Autónoma de Nuevo León en la Facultad de Ingeniería Mecánica y Eléctrica donde comienza esta investigación bajo la supervisión del Dr. Hugo Jair Escalante Balderas.