



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

**NUEVO MÉTODO DE CLUSTERING BASADO EN PROGRAMACIÓN
GENÉTICA Y TEORÍA DE LA INFORMACIÓN**

**TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA
INGENIERÍA, MENCIÓN ELÉCTRICA**

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRICISTA

NEVEN TOMISLAV BORIC BARGETTO

PROFESOR GUÍA:
PABLO ESTÉVEZ VALENCIA

MIEMBROS DE LA COMISIÓN:
CLAUDIO PÉREZ FLORES
RICHARD WEBER HAAS
PABLO ZEGERS FERNÁNDEZ

SANTIAGO DE CHILE
ENERO 2009

**RESUMEN DE LA TESIS PARA OPTAR AL
GRADO DE MAGISTER EN CIENCIAS DE LA
INGENIERÍA, MECIÓN ELÉCTRICA Y AL
TÍTULO DE INGENIERO CIVIL ELÉCTRICISTA
POR: NEVEN BORIC BARGETTO
FECHA: 27/01/2009
PROF. GUIA: DR. PABLO ESTÉVEZ V.**

**“NUEVO MÉTODO DE CLUSTERING BASADO EN PROGRAMACIÓN GENÉTICA Y
TEORÍA DE LA INFORMACIÓN”**

El reconocimiento de patrones es la disciplina que se ocupa de la clasificación de objetos en un número de categorías o clases. Para lograr esta clasificación, se requiere un proceso de aprendizaje, el cual puede ser supervisado o no supervisado. En el primer caso, se cuenta con una referencia o etiqueta previamente asignada, la cual puede ser aprendida por un clasificador. En el caso no supervisado, las etiquetas no existen y la clasificación se hace de acuerdo a alguna noción de similitud. A este caso se le denomina en la literatura como *clustering*.

En esta tesis se propone un nuevo método de *clustering* basado en programación genética y teoría de la información. Programación genética (PG) es una técnica de computación evolutiva que permite encontrar de manera relativamente automática un programa que resuelva adecuadamente un problema dado. La teoría de la información mide el contenido de información presente en un conjunto de datos. En esta tesis se utiliza la divergencia de Cauchy-Schwartz, basada en teoría de la información, como medida de desempeño o *fitness* para los individuos de PG. Esta divergencia mide la entropía de todos los *clusters* que conforman la solución generada por cada individuo de PG. El método propuesto considera además una codificación multiárbol para los individuos y una interpretación probabilística para la salida de éstos. Al iterar el método, se minimiza paulatinamente la entropía, logrando que los patrones pertenecientes a un mismo *cluster* sean similares entre sí.

El método propuesto se evalúa en bases de datos artificiales y reales, de múltiples dimensiones y con *clusters* de formas irregulares. Se compara su desempeño en términos del número de clasificaciones correctas (comparando las etiquetas generadas por el método con las etiquetas reales) con el algoritmo *k-means* y el método de Jenssen, otro método reciente que también optimiza la divergencia de Cauchy-Schwartz. También se analiza su capacidad de generalización, ejecutando el método en un subconjunto de los datos (conjunto de entrenamiento) y luego aplicando el programa resultante para clasificar los datos restantes (conjunto de validación).

Los resultados muestran que el método propuesto es capaz de detectar correctamente los clusters en todos los conjuntos de datos evaluados y supera a los dos métodos con que se compara. Se comprueba que el método es capaz de generalizar el conocimiento adquirido para clasificar nuevos patrones, ya que mantiene el desempeño al pasar del conjunto de entrenamiento al de validación.

Posibles mejoras incluyen la detección automática no sólo de la estructura, sino también del número de clusters presentes en los datos, como la reducción del costo computacional del método.

A Carolina y Lucas

Agradecimientos

Quisiera dedicar algunas palabras para agradecer a quienes me apoyaron, ayudaron o simplemente acompañaron durante mis (largos) años de estudio y durante la realización de esta tesis, esperando no olvidar a nadie.

Primero que nada, agradezco a mis padres, por la educación que me brindaron y el apoyo interminable, que me permitió llegar hasta esta instancia. Agradezco también a mis hermanos, Dusan y Ljuba, por apoyarme (y aguantarme) siempre.

Agradezco a Lucas, mi hijo, quien sin saberlo ha sido la mayor fuente de energía para terminar mi tesis. Agradezco a Carolina, mi novia, quien ha sido siempre un pilar de amor y comprensión. Gracias también a mis suegros y cuñados, por aceptarme como un miembro más de la familia.

Agradezco al profesor Pablo Estévez por aceptarme como ayudante primero y luego como tesista. Fueron varios años de trabajo juntos, durante los cuales el laboratorio fue casi un segundo hogar, que sin duda marcaron y fueron parte fundamental de mi formación como ingeniero y persona.

Gracias a Conicyt, por la beca que me permitió completar el Magíster con tranquilidad y al proyecto Fondecyt 1080643 por financiar parte de esta tesis. Agradezco también al profesor Claudio Pérez, por aceptarme en el proyecto FONDEF de imágenes y a los profesores Claudio Held, Javier Ruiz del Solar y Néstor Becerra, con quienes fue un agrado trabajar. Gracias a los profesores Claudio Zegers y Richard Weber por aceptar formar parte de mi comisión examinadora de tesis. Gracias también a Jimmy y Don Luis, por recibirme siempre con una sonrisa.

Agradezco a mis amigos y compañeros de carrera, de sufrimientos, de carretes que terminan siendo uno solo, y a todos los grandes nerd que acompañaron mi paso por el centro mundial de la ñoñez (laboratorios de inteligencia computacional y procesamiento de imágenes): Meme, Pío, Chino, Nico, Leiva, Fran, Piter, Flowers, Vanel, Leo, Jeffrey, Pablo V., Pablo G., Pablo W., Víctor, Carolina, Gonzalo, Michel, Cristian (viejos estandartes), Schultz, Carlos, Daniel B., Daniel D., Rafa, Jorge, Pelao, Andrés C., Alonso, Alejandro, Renzo, ...

Agradezco a mis eternos compañeros de andanzas por estar siempre ahí: Guatón, Pepe y Dana.

Índice general

| | |
|---|-----------|
| 1. Introducción | 1 |
| 1.1. Objetivos generales | 7 |
| 1.2. Objetivos específicos | 7 |
| 1.3. Estructura de la tesis | 8 |
| 2. Antecedentes | 10 |
| 2.1. Reconocimiento de patrones | 10 |
| 2.2. <i>Clustering</i> | 12 |
| 2.2.1. Métodos de <i>clustering</i> | 16 |
| 2.2.2. Validez del <i>clustering</i> | 25 |
| 2.3. Programación genética | 31 |
| 2.3.1. Operadores genéticos | 36 |
| 2.3.2. Operadores de selección | 39 |
| 2.3.3. Ejemplo: regresión simbólica con PG | 41 |
| 2.3.4. Aplicaciones | 43 |
| 2.4. Diseño de clasificadores con programación genética | 43 |
| 2.4.1. Extensión a múltiples clases | 44 |
| 2.4.2. Medida de <i>fitness</i> | 48 |
| 2.4.3. Operadores genéticos | 48 |
| 2.4.4. Complejidad de los programas resultantes | 49 |
| 2.4.5. Costo computacional | 51 |
| 2.5. <i>Clustering</i> con programación genética | 51 |

| | | |
|-----------|--|-----------|
| 2.5.1. | Representación | 52 |
| 2.5.2. | Medida de <i>fitness</i> | 52 |
| 2.5.3. | Comentarios | 54 |
| 2.6. | <i>Clustering</i> usando otros métodos evolutivos | 55 |
| 2.6.1. | Algoritmos genéticos | 55 |
| 2.6.2. | Optimización por enjambre de partículas (PSO) | 56 |
| 2.6.3. | Funciones objetivo | 56 |
| 2.7. | Teoría de la información | 57 |
| 2.7.1. | Teoría de la información y <i>clustering</i> | 61 |
| 2.7.2. | Potencial de información | 64 |
| 2.7.3. | Divergencia de Cauchy-Schwartz | 66 |
| 2.7.4. | Estimación del tamaño del kernel | 69 |
| 3. | Metodología | 71 |
| 3.1. | Introducción al nuevo método | 71 |
| 3.2. | Representación multiárbol | 72 |
| 3.3. | Interpretación probabilística | 73 |
| 3.4. | Medida de <i>fitness</i> | 77 |
| 3.4.1. | Detalles de implementación | 78 |
| 3.5. | Operadores genéticos | 78 |
| 3.6. | Conjunto de terminales y funciones base | 79 |
| 3.7. | Parámetros que controlan la evolución | 79 |
| 3.8. | Reducción de la complejidad | 80 |
| 3.8.1. | Muestreo estocástico | 80 |
| 3.8.2. | Estimación por vecindad | 81 |
| 3.9. | Experimentos | 82 |
| 3.9.1. | <i>Clustering</i> | 83 |
| 3.9.2. | Generalización | 83 |
| 3.9.3. | Aprendizaje de la densidad de probabilidad condicional | 84 |
| 3.9.4. | Bases de datos | 85 |
| 3.9.5. | Indicadores de desempeño | 89 |

| | |
|---|------------|
| 3.9.6. Reducción de complejidad | 90 |
| 4. Resultados | 91 |
| 4.1. <i>Clustering</i> | 91 |
| 4.2. Generalización | 97 |
| 4.3. Aprendizaje de la densidad de probabilidad condicional | 101 |
| 4.4. Reducción de complejidad | 102 |
| 5. Análisis y discusiones | 107 |
| 5.1. Desempeño del método propuesto | 107 |
| 5.2. Generalización | 109 |
| 5.3. Función de <i>fitness</i> | 110 |
| 5.4. Costo computacional | 111 |
| 5.5. Complejidad | 111 |
| 5.6. Número de <i>clusters</i> | 111 |
| 6. Conclusiones | 113 |
| 6.1. Recomendaciones para trabajo futuro | 114 |
| Bibliografía | 115 |
| A. Publicación en IEEE Congress on Evolutionary Computation (CEC 2007) | 126 |

Capítulo 1

Introducción

Una de las tareas primordiales realizadas a diario por la mente humana es la de clasificación de patrones. Cada vez que se recibe un nuevo estímulo, la mente lo procesa y clasifica, agrupándolo con otros estímulos similares y asignándole una etiqueta. Sólo después de este proceso se puede decir que una persona entiende la información recibida. De esta manera la mente pasa, por ejemplo, de ver un conjunto de láminas verdes colgando de varas color café a ver un árbol [1].

Para lograr esta clasificación, se requiere un proceso de aprendizaje, el cual puede ser supervisado o no supervisado. En el primer caso, se cuenta con una referencia o etiqueta previamente asignada, la cual puede ser aprendida por un clasificador. En el caso no supervisado, las etiquetas no existen y la clasificación se hace de acuerdo a alguna noción de similitud [2].

Un ejemplo de clasificador es un sistema que identifica personas utilizando fotografías. En el caso supervisado se le suministra al sistema una serie de fotografías correspondientes a distintos individuos, indicando el nombre (etiqueta) de cada uno. Luego de un proceso de aprendizaje, el sistema debiera ser capaz de identificar a cada individuo observando sólo su fotografía. Además, es de interés que el sistema *generalice*, es decir, que pueda reconocer a los individuos si se le presentan fotografías no vistas con anterioridad de alguno de ellos. En el caso de aprendizaje no supervisado se presentan al sistema las fotografías, pero omitiendo los nombres (referencias). En este caso el sistema agrupará las fotografías similares de acuerdo a un criterio preestablecido. Como resultado, el sistema podría formar grupos según el individuo al cual pertenecen, como también según el color de la piel, sexo, presencia o ausencia de lentes, etc., dependiendo del criterio de similitud utilizado.

Ambos tipos de clasificadores son utilizados comúnmente en múltiples campos de la ingeniería y ciencias naturales, incluso en ciencias sociales. Por esta razón, en la literatura se pueden encontrar innumerables enfoques para crear clasificadores. De hecho, los distintos autores ni siquiera concuerdan en los nombres que se les da a cada tipo de clasificador. La clasificación supervisada puede también ser referida como asignación o discriminación. El caso no supervisado se conoce principalmente como *clustering*, aunque puede encontrarse como taxonomía numérica, tipología, particionamiento o simplemente aprendizaje no supervisado [3]. Incluso hay autores que intercambian los nombres, llamando clasificación al problema de *clustering* y restringiendo la denominación de reconocimiento de patrones (que generalmente engloba a ambos problemas) al caso supervisado [4].

En términos formales, dado un conjunto de patrones multidimensionales $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, con $\mathbf{x}_i \in \mathbb{R}^d$, el problema de clasificación consiste en particionar X en c subconjuntos o *clases* a través de la regla de asignación $D : \mathbb{R}^d \rightarrow \{0, 1\}^c$, con $\sum_{k=1}^c D_k(\mathbf{x}) = 1$.

El reconocimiento estadístico de patrones comienza de la suposición que para cada una de las clases C_k , con $k \in \{1, \dots, c\}$, existe un modelo generador de los datos dado por la función de densidad de probabilidad (*pdf*, por sus siglas en inglés) condicional $p(\mathbf{x}|C_k)$. El clasificador óptimo bayesiano consiste en asignar las clases de manera de minimizar el riesgo condicional [5]:

$$R(C_k|\mathbf{x}) = \sum_{k'=1}^c L(C_k, C_{k'}) \cdot P(C_{k'}|\mathbf{x}) \quad (1.1)$$

donde $L(C_k, C_{k'})$ es el costo de asociar a \mathbf{x} con la clase C_k , cuando en realidad pertenece a $C_{k'}$ y $P(C_{k'}|\mathbf{x})$ es la probabilidad *a posteriori* de que \mathbf{x} pertenezca a $C_{k'}$. En el caso más común, cuando el costo es igual para todas las clases, se tiene:

$$L(C_k, C_{k'}) = \begin{cases} 0, & k = k' \\ 1, & k \neq k' \end{cases}$$

con lo que el riesgo pasa a ser

$$R(C_k|\mathbf{x}) = \sum_{k' \neq k} P(C_{k'}|\mathbf{x}) = 1 - P(C_k|\mathbf{x}) \quad (1.2)$$

La regla de decisión de Bayes consiste en asignar el patrón a la clase con mayor probabilidad *a posteriori*. Es decir, se debe asignar el patrón \mathbf{x} a la clase C_k si y sólo si:

$$P(C_k|\mathbf{x}) > P(C_{k'}|\mathbf{x}) \forall k' \neq k \quad (1.3)$$

En clasificación supervisada existen etiquetas previamente asignadas —generalmente, por un experto en la materia— que indican la pertenencia de cada uno de los patrones a una clase preestablecida. Un clasificador supervisado es *entrenado* usando los patrones y sus respectivas etiquetas. El objetivo último de este proceso es que el clasificador aprenda los modelos que definen las distintas clases y sea capaz de aplicar este conocimiento para clasificar correctamente nuevos patrones que no hayan sido vistos con anterioridad.

En el caso no supervisado (*clustering*), en cambio, no hay etiquetas ni clases preestablecidas. Un algoritmo de *clustering* debe particionar los datos de manera de reflejar su estructura interna, agrupando juntos patrones *similares* y separando aquellos *diferentes*. A cada grupo de datos formado se le denomina *cluster* y al conjunto de todos los grupos (es decir, al resultado del algoritmo) como *clustering*. Esta definición es de naturaleza circular, ya que la estructura de los *clusters* resultantes dependerá de la noción de similitud utilizada [3]. En general, se espera que un algoritmo sea capaz de separar los datos usando los mismos conceptos de similitud que ocuparía un humano al inspeccionarlos visualmente, por ambiguo que esto parezca.

A pesar de que algunos autores señalan que “el objetivo de un algoritmo de *clustering* es simplemente encontrar una organización conveniente y válida de los datos, no establecer reglas para separar datos futuros en categorías” [6], resulta también de interés utilizar el *clustering* resultante para predecir la etiqueta de nuevos datos, es decir, generalizar el conocimiento adquirido. Este proceso se denomina “predicción basada en grupos” en [3]. La mayoría de los algoritmos de *clustering* existentes deja de lado esta opción.

Dada la gran cantidad de información que se maneja en el mundo de hoy, resulta esencial contar con métodos que realicen tareas de clasificación de forma automática, para así poder reducir dicha información de manera que pueda ser procesada y finalmente entendida por personas.

En la literatura se encuentran innumerables ejemplos de técnicas para diseñar clasificadores supervisados. Dentro de las más comunes se encuentran los clasificadores bayesianos, las redes neuronales y las máquinas de soporte vectorial (SVM).

Para el caso de *clustering*, la variedad de métodos es inmensa. Dependiendo del tipo de resultado que generen, los algoritmos pueden ser particionales (*k-means*), jerárquicos (enlace simple y completo), difuso (*fuzzy c-means*), basado en densidad (mapas auto organizativos ó SOM), etc. Existen métodos que intentan encontrar automáticamente el número “correcto” de *clusters*, pero la mayoría lo asume como un dato conocido, que debe ser especificado por el usuario.

Los métodos particionales son los más simples y, quizás por eso, los más comúnmente utilizados. La mayoría de los métodos de *clustering* particionales están basados en optimizar una función de costos que considera estadísticos de segundo orden de los datos. Por ejemplo, un enfoque común es minimizar la distancia entre los patrones y los centroides (promedios) de cada *cluster*, lo que es equivalente a minimizar la varianza de los patrones de cada *cluster*. De esta manera, se está asumiendo implícitamente una distribución gaussiana para cada *cluster* y, por lo tanto, aquellos métodos sólo podrán identificar *clusters* de forma hipersférica o hiperelipsoidal.

Recientemente, han surgido nuevos métodos de *clustering* basados en conceptos de teoría de la información [7–10]. La teoría de la información [11, 12] estudia el contenido de información presente en un mensaje (en el caso de *clustering*, los patrones). A través de conceptos como la entropía y la información mutua, la teoría de la información permite extraer estadísticos de mayor orden de los datos y así capturar de mejor manera su estructura interna, sin imponer restricciones sobre el tipo de distribución. Sin embargo, dado que en reconocimiento de patrones se trabaja con muestras de variables que en la realidad son continuas, los conceptos de teoría de la información son imposibles de calcular de manera exacta y deben ser estimados a través de los patrones disponibles. Al respecto, Gockay y Príncipe [7, 13] propusieron una función de costos para problemas de *clustering* basada en la entropía de Renyi. Li et. al [8] usaron el método de los k vecinos más cercanos para estimar la entropía Havrda-Charvat de cada *cluster*. Ambos enfoques utilizan algoritmos iterativos ad-hoc para la optimización de los funcionales propuestos. Jenssen et al. [9, 10] optimizan la divergencia de Cauchy-Schwartz, recientemente propuesta en [14], usando descenso por gradiente. Dadas las técnicas de optimización utilizadas, todos estos métodos basados en teoría de la información presentan deficiencias en su capacidad de convergencia. Además, ninguno de ellos contempla la posibilidad de generalizar el conocimiento y clasificar nuevos patrones.

Programación genética (PG) es una técnica para generar programas basada en los principios de la evolución de las especies [15]. Al igual que en otros métodos evolutivos, en PG se genera aleato-

riamente una población inicial de “individuos”, que son posibles soluciones al problema tratado. Luego en cada “generación” los individuos son modificados y combinados entre ellos, seleccionando con mayor probabilidad a los individuos más aptos, aquellos que resuelvan mejor el problema. Este proceso de selección y reproducción lleva a encontrar soluciones cada vez mejores, tendiendo eventualmente a un óptimo global. En PG, las estructuras que evolucionan son programas, representados generalmente por árboles de funciones. Estos programas pueden formar estructuras complejas que logren solucionar problemas difíciles. PG ha sido aplicada con éxito a variados campos de la ingeniería, muchas veces sobrepasando métodos clásicos, replicando o mejorando sistemas diseñados a mano [16–18] e incluso patentados [19, 20].

Debido al gran poder demostrado por PG en numerosos campos, resulta interesante estudiar su aplicación al diseño de clasificadores, ya sean supervisados o no supervisados. Según [21], las ventajas esperadas de aplicar programación genética al diseño de clasificadores son:

- No es necesario asumir a priori alguna distribución de probabilidad para los datos.
- Se pueden descubrir automáticamente las variables relevantes para discriminar entre las clases.
- Los árboles resultantes pueden ser analizados para extraer nuevo conocimiento (minería de datos).
- El resultado es un programa clasificador que puede ser aplicado posteriormente.

A estas ventajas esperadas se pueden agregar que la estructura libre de los árboles generados en PG puede ser aprovechada para separar regiones complejas con cualquier forma y en espacios multidimensionales. Otra ventaja, compartida con otros métodos evolutivos, es la capacidad de evitar mínimos locales en la optimización mucho mejor que métodos basados en gradiente.

PG ha sido aplicada desde sus inicios al diseño de clasificadores supervisados binarios ($c = 2$) [15]. El enfoque más común consiste en dividir el espacio de salida (unidimensional) de los árboles de tal manera que una salida menor que cero para un patrón signifique la pertenencia a una clase, mientras que una salida positiva indica la pertenencia a la otra clase. En el caso de múltiples clases ($c > 2$), la aplicación de PG no es directa, por lo que diversos enfoques han sido estudiados en la literatura [21–36]. En [21] el problema de múltiples clases se modela como un conjunto de problemas binarios. Otros enfoques consisten en segmentar el espacio de salida y asignar cada segmento a una de las clases [22–27], o interpretar la salida de los árboles como una nueva característica y utilizar

un clasificador estándar para discriminar entre las clases a través de esta característica generada [31–36]. Hasta ahora, el enfoque con mejores resultados es el propuesto por Muni et al. [28, 29], donde la representación de los individuos se modifica y se usan individuos con múltiples árboles. Un individuo está formado por c árboles para un problema de c clases. Cada árbol dentro de un individuo está asociado a una clase y codifica una regla que se activa cuando un patrón es reconocido como parte de la clase. Con esta interpretación, se puede dar el caso de que múltiples reglas se activen para un mismo patrón, por lo que se requiere una etapa de resolución de conflictos para conocer sin ambigüedad la clasificación generada por un individuo.

Estas y otras aplicaciones han demostrado que PG es una poderosa herramienta para el diseño de clasificadores supervisados. En [28] y [29] se realizan extensas comparaciones del desempeño obtenido por PG y otros clasificadores: métodos estadísticos, clasificadores basados en reglas y redes neuronales. PG supera a los demás métodos en todas las bases de datos utilizadas. La aplicación a clasificadores no supervisados (*clustering*), en cambio, ha sido escasamente explorada. Para diseñar un método de *clustering* basado en PG es necesario determinar:

- Una estructura para los individuos que permita traducir la salida de uno de ellos en una etiqueta al ser evaluados.
- Una interpretación que no permita ambigüedades respecto a la pertenencia, evitando así conflictos.
- Una medida de *fitness* que permita la evolución de programas que separen conjuntos con formas irregulares, no sólo hipersféricas.
- Conjunto de terminales, funciones base y demás parámetros que controlan la evolución.

Hasta la fecha, el único trabajo sobre el tema es el de De Falco et al. [37]. Su enfoque es similar a [28], dado que también usan una representación multiárbol para los individuos, donde cada árbol codifica una regla binaria. Por lo tanto, también requiere una etapa de resolución de conflictos. El método intenta encontrar automáticamente el número óptimo de *clusters* y para esto utiliza un número variable de árboles por individuo. Sin embargo, en vez de especificar c , el usuario debe especificar otro parámetro, posiblemente menos intuitivo, que influye directamente en el número de *clusters* encontrado. La medida de *fitness* está basada en la distancia de los patrones a los

centroides y entre los pares de centroides. Esto restringe al método a formar *clusters* hiperesféricos, disminuyendo quizás la principal ventaja de usar PG.

En esta tesis se propone un nuevo esquema para diseñar clasificadores no supervisados utilizando programación genética. El esquema está basado en individuos multiárboles, pero se introduce una interpretación probabilística que no requiere de resolución de conflictos. La medida de *fitness* propuesta está basada en conceptos de teoría de la información y permite que los programas generados separen *clusters* de cualquier forma, no sólo hiperesférica. Comparado con otros métodos evolutivos y con métodos basados en teoría de la información, la mayor ventaja del método propuesto es que el resultado no es sólo la partición óptima, sino el programa (conjunto de funciones de pertenencia) que genera dicha partición. Como consecuencia, los programas generados por este método pueden ser utilizados para generalizar el conocimiento adquirido y clasificar nuevos patrones no vistos anteriormente. Además, utilizando el teorema de Bayes, la salida de los programas puede ser interpretada como la función de densidad de probabilidad condicional de cada uno de los *clusters* presentes en los datos. Así, el método estima de manera indirecta el modelo generador de los datos, que puede ser evaluado en cualquier punto del espacio de entrada, no sólo en los patrones disponibles.

1.1. Objetivos generales

Diseñar un nuevo método de *clustering* basado en programación genética y conceptos de teoría de la información, que presente en alguna medida las siguientes características:

- Separe *clusters* con cualquier forma y en múltiples dimensiones.
- No necesite especificar un modelo a priori para los datos.
- Logre generalizar el conocimiento obtenido para clasificar nuevos elementos.
- Pueda estimar el modelo generador para cada clase presente en los datos.

1.2. Objetivos específicos

1. Diseñar un método de *clustering* basado en PG y teoría de la información, especificando una configuración adecuada de los siguientes elementos:

- Representación para los individuos (posibles soluciones al problema).
- Interpretación de la salida de los individuos.
- Medida de *fitness* basada en teoría de la información.
- Parámetros de configuración de PG.

de manera de cumplir en la mayor medida posible con las características enunciadas en la sección anterior.

2. Comparar el desempeño del método propuesto con el algoritmo básico *k-means* y el método de Jenssen presentado en [9, 10], con respecto a la capacidad de separar *clusters* con distintas formas en bases de datos reales y artificiales de múltiples dimensiones.
3. Medir la capacidad de generalización del método, entrenándolo con versiones reducidas de las mismas bases de datos y luego aplicando las soluciones resultantes para clasificar nuevos patrones.
4. Verificar la capacidad de aprender el modelo generador de los datos, analizando el comportamiento de las soluciones obtenidas en regiones continuas en el espacio de entrada de los datos, que incluyan a los patrones de entrenamiento.
5. Publicar los resultados obtenidos en una conferencia internacional de alto nivel especializada en computación evolutiva.

1.3. Estructura de la tesis

La estructura de lo que resta de esta tesis es la siguiente:

En el capítulo 2 se describen los conceptos básicos relacionados con esta tesis, como son *clustering*, programación genética y teoría de la información, y se hace una breve revisión de métodos existentes en la literatura. El capítulo 3 describe el método propuesto y los experimentos diseñados para evaluar su desempeño. En el capítulo 4 se presentan los resultados de los experimentos y un breve análisis de cada uno. En el capítulo 5 se presenta un análisis más profundo de los resultados, revisando el desempeño del método propuesto y cómo se comporta comparado con los otros métodos evaluados. Finalmente, en el capítulo 6 se exponen las conclusiones de esta tesis.

Parte de los resultados de esta tesis fueron presentados en la conferencia IEEE Congress on Evolutionary Computation (CEC 2007), desarrollada en Singapur, entre los días 25 y 28 de Septiembre de 2007 [38]. El texto de dicha publicación se incluye en el apéndice A.

Capítulo 2

Antecedentes

2.1. Reconocimiento de patrones

El reconocimiento de patrones es la disciplina que se ocupa de la clasificación de objetos en un número de categorías o clases [3]. El tipo de objetos a clasificar depende de la aplicación particular. En la literatura de reconocimiento de patrones, estos objetos son referidos como *patrones* y en general corresponden a mediciones de algún fenómeno real. Los patrones son representados como vectores y pueden corresponder a imágenes, formas de onda de señales o cualquier otro tipo de mediciones que necesiten ser clasificadas. Cada dimensión del vector corresponde a una característica distinta que se mide del fenómeno.

Formalmente, un patrón es un vector $\mathbf{x} \in \mathbb{R}^d$, es decir, $\mathbf{x} = [x_1, \dots, x_d]^T$, con d el número de dimensiones o características y donde cada x_l , con $l = 1, \dots, d$, es una variable aleatoria que representa una de las características medidas.

El reconocimiento de patrones está basado principalmente en el análisis estadístico de las variables aleatorias asociadas a los datos. Por tratarse de variables aleatorias, para cada una de las características medidas del fenómeno de interés existirá una función de densidad de probabilidad que describe el comportamiento de la variable. Una función de densidad de probabilidad $p(x)$ especifica que la probabilidad de que la variable x se encuentre entre dos puntos cualesquiera $x = a$ y $x = b$ está dada por:

$$P(x \in [a, b]) = \int_a^b p(x) dx \quad (2.1)$$

Asimismo, se puede considerar el vector \mathbf{x} como una variable aleatoria multidimensional que engloba a las d variables aleatorias del problema. Por lo tanto, se define también $p(\mathbf{x})$, la función de densidad de probabilidad del vector \mathbf{x} , de tal manera que la probabilidad de que \mathbf{x} esté en una región $R \subset \mathbb{R}^d$ está dada por:

$$P(\mathbf{x} \in R) = \int_R p(\mathbf{x}) d\mathbf{x} \quad (2.2)$$

En general, para un problema de reconocimiento de patrones se dispone de un conjunto de patrones $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, con $\mathbf{x}_i \in \mathbb{R}^d$, donde cada patrón \mathbf{x}_i corresponde a una muestra de \mathbf{x} . En términos formales, dado X , el problema de clasificación consiste en particionar X en c subconjuntos o *clases* a través de la regla de asignación $D : \mathbb{R}^d \rightarrow \{0, 1\}^c$, con $\sum_{k=1}^c D_k(\mathbf{x}) = 1$. Cada una de las clases se denomina como C_k , con $k \in \{1, \dots, c\}$.

El reconocimiento de patrones estadístico comienza de la suposición de que para cada una de las clases existe un modelo distinto que gobierna el comportamiento de los datos. Esto se traduce en que, en vez de una función de densidad de probabilidad $p(\mathbf{x})$ que determine globalmente el comportamiento de \mathbf{x} , existen c funciones de densidad de probabilidad distintas, denominadas funciones de densidad de probabilidad condicional y denotadas como $p(\mathbf{x}|C_k)$. Cada una de éstas funciones define el comportamiento de \mathbf{x} para una de las clases, por lo que se dice que $p(\mathbf{x}|C_k)$ corresponde al modelo generador de los datos para la clase C_k .

En un problema de clasificación supervisada existen etiquetas previamente asignadas por un instructor que indican la pertenencia de cada uno de los patrones a una clase preestablecida. Un algoritmo de clasificación debe estimar el modelo generador para cada clase, $p(\mathbf{x}|C_k)$, de manera de preservar las etiquetas de los datos de entrenamiento, y lograr predecir lo mejor posible las etiquetas de nuevos datos, es decir, minimizando el riesgo condicional (1.1). Los métodos de clasificación se pueden dividir a modo grueso en paramétricos y no paramétricos: un método paramétrico asume una cierta distribución para el modelo generador de cada clase (por ejemplo, una distribución normal o gaussiana) y a través de los datos estima los parámetros de las distribuciones que minimizan el riesgo condicional; un método no paramétrico no asume ninguna distribución *a priori* para los datos. Como es de esperarse, un método no paramétrico es más poderoso, ya que puede representar distribuciones más complejas, pero esto a cambio de tener más grados de libertad, es decir, posee más parámetros que deben ser ajustados a los datos.

2.2. *Clustering*

El análisis de *clusters*, o simplemente *clustering*, consiste en particionar un conjunto de datos o patrones de manera que aquellos pares de patrones similares entre sí queden en el mismo grupo, mientras que los pares de patrones que no son similares queden en grupos distintos. A cada grupo de patrones resultante se le denomina comúnmente en inglés como *cluster*, de donde proviene el término *clustering* (agrupación o agrupamiento¹). El análisis de *clusters* suele referirse también como clasificación no supervisada, ya que en términos generales consiste en un problema de clasificación (se debe asignar una clase o etiqueta a cada patrón), sólo que en la ejecución del algoritmo no se dispone de etiquetas reales que deban ser aprendidas. El aprendizaje de las clases debe hacerse utilizando sólo los patrones mismos y medidas de similitud entre ellos. De ahí el concepto de aprendizaje no supervisado.

Otras posibles definiciones para *clustering* son:

“Aquellos métodos que se ocupan de alguna manera de la identificación de grupos homogéneos de objetos”

y

“Un *cluster* es un conjunto de entidades que son similares, mientras que las entidades de *clusters* distintos no son similares”

Estas y otras definiciones de *clustering* encontradas en la literatura son más bien informales. De hecho, se puede agregar que la mayoría son ambiguas y hasta circulares, ya que no se define la noción de similitud, y es claro que su definición determinará el tipo de *clusters* a formar.

Por ejemplo [3], considérese los siguientes animales: oveja, perro, gato (mamíferos), gorrión, gaviota (aves), serpiente, lagarto (reptiles), trucha, salmón, tiburón (peces), y rana (anfibio). Si se quisiera separar estos animales en *clusters*, es necesario definir un criterio de similitud o “criterio de *clustering*”. Si el criterio es la forma en que nacen las crías de los animales, la oveja, el perro, el gato y el tiburón quedarían en el mismo *cluster* y todos los demás animales quedarían en un segundo *cluster*. Si el criterio es la existencia de pulmones, la trucha, el salmón y el tiburón quedan en un

¹La traducción más adecuada para *clustering* sería agrupamiento, ya que ambas palabras describen a la vez la acción y el efecto de agrupar (es decir, pueden ser usadas como verbos o sustantivos). Es por esto que es correcto hablar tanto de un algoritmo de *clustering*, como del *clustering* producido por el algoritmo. En esta tesis se ocupa casi sin excepción la denominación en inglés.

mismo *cluster*, dejando al resto en un segundo *cluster*. Por otro lado, si el criterio es el ambiente donde viven los animales (terrestre o acuático), la oveja, el perro, el gato, el gorrión, la gaviota, la serpiente y el lagarto formarán un *cluster* (animales terrestres), la trucha, el salmón y el tiburón formarán un segundo *cluster* (animales acuáticos), mientras que la rana formará un *cluster* por si sola, ya que puede vivir en el agua o fuera de ella. Cabe notar también que si el criterio es la existencia de una columna vertebral, todos los animales quedarían en el mismo *cluster*. Finalmente, se pueden usar también criterios compuestos. Por ejemplo, si el criterio es la forma en que nacen las crías y la existencia de pulmones, se formarán cuatro *clusters*. En definitiva, este ejemplo muestra que el proceso de asignar objetos a *clusters* puede llevar a resultados muy distintos, dependiendo del criterio de similitud utilizado.

El *clustering* es una de las tareas mentales más primitivas realizadas por los seres humanos, que la usan para manejar las inmensas cantidades de información que reciben a diario. Procesar cada pedazo de información como una entidad independiente sería imposible. Por lo tanto, los seres humanos tienden a categorizar las entidades (objetos, personas, eventos) en *clusters*. Cada *cluster* es luego caracterizado por los atributos comunes de las entidades que contiene. Por ejemplo, la mayoría de las personas ha formado inconscientemente un *cluster* “perro”. Si alguien ve a un perro, lo identificará inmediatamente como una entidad del *cluster* “perro”. Por lo tanto, la persona asumirá que esa entidad ladra, a pesar de nunca antes haber visto ladrar a aquel perro en particular.

Tal como se describe en la sección 2.1, en reconocimiento de patrones, las entidades a clasificar son patrones multidimensionales que en general representan mediciones de algún fenómeno real. El problema de *clustering* consiste formalmente en, dado un conjunto de datos $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, particionar X en c subconjuntos (*clusters*) de manera de que los patrones que sean similares entre sí queden en el mismo subconjunto y los patrones que no sean similares entre sí queden en subconjuntos distintos. A diferencia del caso supervisado, en general no se dispone de etiquetas C_k asignadas previamente por un instructor, por lo que el *clustering* se debe generar sólo de acuerdo a las similitudes entre los patrones. En el ejemplo de los animales, se dijo que las distintas agrupaciones que se podían formar dependían del criterio de similitud utilizado y que el problema de *clustering* era ambiguo sin definir este concepto. Al trabajar con patrones, esta ambigüedad se manifiesta en la ausencia de una definición matemática del concepto de similitud entre los pares de patrones. Esto sucede porque el concepto de *cluster* es una generalización a una cantidad arbitraria de dimensiones

de lo que los seres humanos perciben en dos o tres dimensiones como nubes de alta densidad de puntos, una intuición que es difícil de traducir a conceptos matemáticos. Por ejemplo, la figura 2.1, tomada de [6], muestra varios *clusters* de distinta naturaleza. No es claro cuántos *clusters* existen en la figura. A gran escala, se perciben cuatro grandes *clusters*. Pero a una escala más fina, la respuesta probablemente variará según el observador. Algunas personas podrían identificar 9 *clusters* y otras 12, sin que ninguna respuesta sea más correcta que la otra. En la figura se aprecian *clusters* de forma circular, alargada, e incluso uno tipo anillo. Es difícil encontrar una definición matemática precisa de un *cluster* que englobe todos estos posibles casos. Más aún, mientras los algoritmos de *clustering* son diseñados para capturar la noción humana de un *cluster*, su fin último es la aplicación para la clasificación no supervisada de conjuntos de datos complejos, de alta dimensión y no interpretables por humanos. Por lo tanto, medir el desempeño de un método de *clustering* es un problema en sí mismo: dada la falta de una definición formal de un *cluster*, no existe un criterio de desempeño objetivo, a menos que se considere conocimiento humano previo, en algunos casos presente en forma de etiquetas previamente asignadas por un experto. Se han definido muchos criterios para comparar desempeños de métodos de *clustering*, algunos que consideran este conocimiento previo y otros que no. Estos criterios se denominan índices de validez y se describen en la sección 2.2.2.

Los métodos de *clustering* son aplicados en las más diversas disciplinas. Donde quiera que se reúna gran cantidad de datos, existirá la necesidad de agruparlos de manera no supervisada, en general como una etapa de análisis preliminar. Según [3], se pueden identificar cuatro direcciones básicas en que el *clustering* es de utilidad:

- Reducción de datos. En muchas aplicaciones la cantidad de datos disponibles, n , es muy grande y, como consecuencia, su procesamiento se vuelve muy demandante. El análisis de *clusters* se puede usar para agrupar los datos en un número “inteligente” de *clusters* M , con $M \ll n$, para luego procesar cada *cluster* como una entidad independiente. Por ejemplo, en transmisión de datos, se define un representante para cada *cluster*. Luego, en vez de transmitir cada patrón, se transmite un código que corresponde al representante del *cluster* donde se ubica el patrón. Así se obtiene una compresión de los datos.
- Generación de hipótesis. En este caso se aplica el análisis de *clusters* a un conjunto de datos de manera de inferir algunas hipótesis acerca de la naturaleza de los datos. Así, el *clustering* se usa como una manera de sugerir hipótesis, las cuales deben ser verificadas usando otros

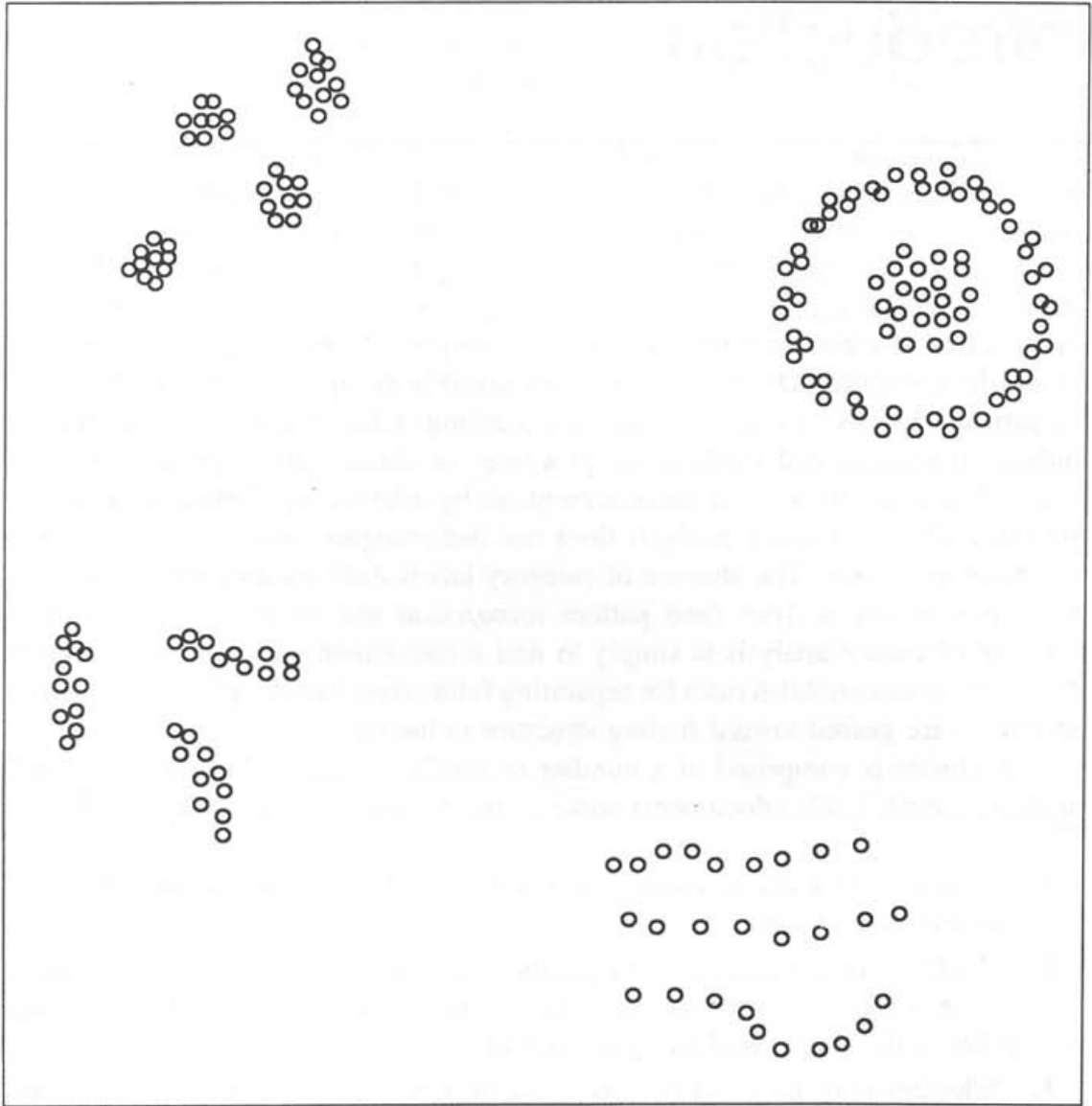


Figura 2.1: Ejemplos de distintos tipos de *clusters*. No es claro cuantos *clusters* existen en la figura, ya que no existe una definición precisa del concepto de *cluster*

conjuntos de datos.

- Verificación de hipótesis. En este contexto, el análisis de *clusters* se usa para verificar la validez de cierta hipótesis específica. Considérese, por ejemplo, la siguiente hipótesis: “las grandes compañías invierten en el extranjero”. Una forma de verificar si esto es cierto es aplicar análisis de *clusters* a un grupo grande y representativo de compañías. Supóngase que cada compañía está representada por su tamaño, sus actividades en el extranjero y su capacidad de llevar a cabo proyectos de investigación. Si luego de aplicar análisis de *clusters*, se forma un *cluster* que contiene a las compañías que son grandes y tienen inversiones en el extranjero (independiente de su habilidad para los proyectos de investigación), entonces la hipótesis es apoyada por el análisis de *clusters*.
- Predicción basada en grupos. En este caso se aplica el análisis de *clusters* al conjunto de datos disponible y los *clusters* resultantes son caracterizados basándose en las características comunes de los patrones que los forman. Luego, si se recibe un patrón desconocido, se puede determinar el *cluster* al cual es más probable que el patrón pertenezca y asumir que el patrón presentará las mismas características que se aprecian en los demás patrones del *cluster*. Supóngase, por ejemplo, que se aplica análisis de *clusters* a un conjunto de datos de pacientes que padecen de la misma enfermedad. Como resultado del análisis se obtienen varios *clusters* de pacientes, de acuerdo a cómo reaccionan a ciertas drogas. Luego, para un nuevo paciente, se identifica el *cluster* más apropiado y basado en esto se decide la droga a utilizar para su tratamiento.

2.2.1. Métodos de *clustering*

Dada la gran cantidad de disciplinas donde es de utilidad aplicar análisis de *clusters*, se han desarrollado numerosas metodologías utilizando los más diversos enfoques. Esto, añadido al hecho de que no existe una definición clara y universal de lo que es un *cluster*, ha promovido la proliferación de algoritmos de *clustering*. Además, muchos resultados sólo se comunican dentro del círculo relacionado con una disciplina en particular, por lo que cada área de investigación puede tener sus propios métodos de *clustering* preferidos. Esto incluso ha llevado a algunos autores a plantearse el porqué de esta situación [39], buscando una especie de teoría unificadora del *clustering*.

Los autores concuerdan en que no existe un método de *clustering* que sea capaz de encontrar todos los tipos de *clusters* posibles en cualquier conjunto de datos [40]. Un trabajo interesante al

respecto es el de Kleinberg [41], que realiza un análisis axiomático del problema de *clustering*. Se asume que un algoritmo de *clustering* sólo recibe como entrada los pares de distancias entre un conjunto finito de datos, y debe producir una partición de éstos. Luego se definen tres propiedades que debieran esperarse de un algoritmo de *clustering*:

- Invarianza a la escala. El método produce los mismos resultados si las distancias entre patrones se multiplican por un valor $\alpha > 0$.
- Riqueza. El método es capaz de producir todas las posibles particiones de los datos.
- Consistencia. Si se tiene una nueva función de distancia que reduce la distancia entre patrones del mismo *cluster*, pero aumenta la de patrones en *clusters* distintos, entonces el algoritmo debiera producir la misma partición.

En su trabajo, Kleinberg demuestra la imposibilidad de diseñar un método de *clustering* que satisfaga las tres condiciones simultáneamente. Además, se muestra que al relajar algunas de las propiedades, se evidencian las concesiones asociadas a muchos algoritmos clásicos, varios de los cuales se describen a continuación. En definitiva, de [41] se puede concluir que no existe el algoritmo de *clustering* perfecto, y distintos algoritmos serán adecuados para encontrar distintos tipos de *clusters* en los datos.

Debido a la gran extensión de posibilidades para hacer análisis de *clusters*, está fuera del alcance de esta tesis hacer una revisión completa de todas las metodologías disponibles. A continuación se realiza una breve descripción de distintos enfoques. Revisiones más completas pueden encontrarse en las publicaciones [5, 40, 42] y en los libros [3, 4, 6, 43, 44].

Existen muchas formas de las que se pueden clasificar los métodos de *clustering*. Primero, es necesario hacer una diferencia entre *clustering* y cuantización vectorial. Ambos conceptos están muy relacionados e incluso muchas publicaciones ni siquiera hacen una diferencia, ya que ambos tratan de particionar los datos de manera no supervisada. La gran diferencia está en el objetivo de la clasificación: en cuantización vectorial se debe buscar una representación aproximada de los datos de manera de minimizar el error cometido con esta representación. En *clustering* simplemente se busca identificar grupos de patrones similares. En la figura 2.2 se puede ver el resultado típico de aplicar ambas técnicas en el mismo conjunto de datos. La figura 2.2(a) muestra un resultado posible de aplicar un método de *clustering*; el método separa los datos en dos grandes *clusters*.

La figura 2.2(b) muestra un posible resultado de un método de cuantización vectorial; el método no se preocupa de encontrar los dos *clusters*, sino de buscar prototipos (en rojo en la figura) que representen fielmente a los datos; al representar los patrones por sus prototipos más cercanos, se logra una compresión de los datos. Se podría decir entonces que cuantización vectorial está orientada a la compresión de datos mientras que el *clustering* a la clasificación, aunque fácilmente se podría argumentar que ambas técnicas son equivalentes, sólo que operan a distinta resolución o escala. De hecho, los dos resultados de la figura 2.2 fueron obtenidos ejecutando el mismo algoritmo (*k-means* [45]), pero indicando al método que encontrara 2 *clusters* en un caso y 20 en el otro. Algunos métodos de cuantización vectorial conocidos son los mapas auto-organizativos (SOM) [46], Neural Gas [47], y el algoritmo LBG [48].

Otra posible clasificación es según el resultado obtenido por el método de *clustering*. Según este criterio, los algoritmos se pueden clasificar principalmente en particionales y jerárquicos [40]. La figura 2.3 muestra un diagrama con una posible clasificación de los distintos enfoques para hacer análisis de *clusters*. A continuación se describen brevemente las distintas categorías.

Algoritmos jerárquicos

Un método de *clustering* jerárquico produce una serie de divisiones anidadas de los datos. El diagrama que representa esta clasificación jerárquica se denomina dendrograma. Un ejemplo de dendrograma se muestra en la figura 2.4. Las líneas representan las separaciones encontradas por el algoritmo. En el nivel superior, los datos se separan en dos grandes *clusters*. A medida que se desciende por el dendrograma, se obtienen particiones más finas de los datos, llegando hasta el nivel individual, donde cada patrón forma su propio *cluster*. Los largos de las líneas representan las distancias entre los pares de *clusters*. Los métodos jerárquicos son más apropiados que los particionales para datos con características nominales u ordinales [5]. Dentro de los métodos jerárquicos más conocidos están los de enlace único y de enlace simple, siendo la mayoría de los restantes variaciones de éstos. Los dos métodos difieren en la manera que caracterizan la similitud entre pares de *clusters*. En el método de enlace único, la distancia entre dos *clusters* es el mínimo de las distancias de todos los pares de patrones de cada *cluster*. En enlace completo, la distancia entre dos *clusters* corresponde al máximo de las distancias entre los pares de patrones. En ambos casos, dos *clusters* se fusionan en uno más grande basado en un criterio de mínima distancia.

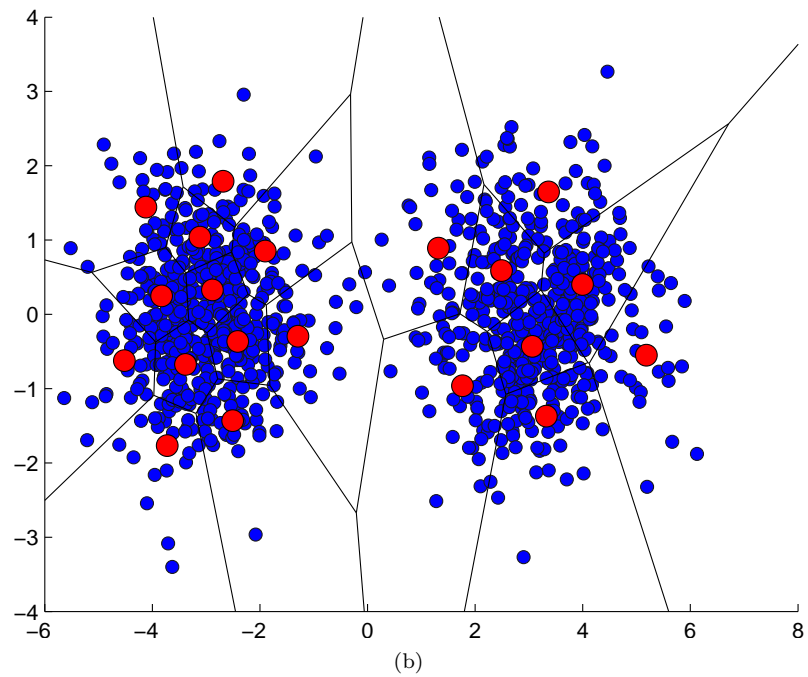
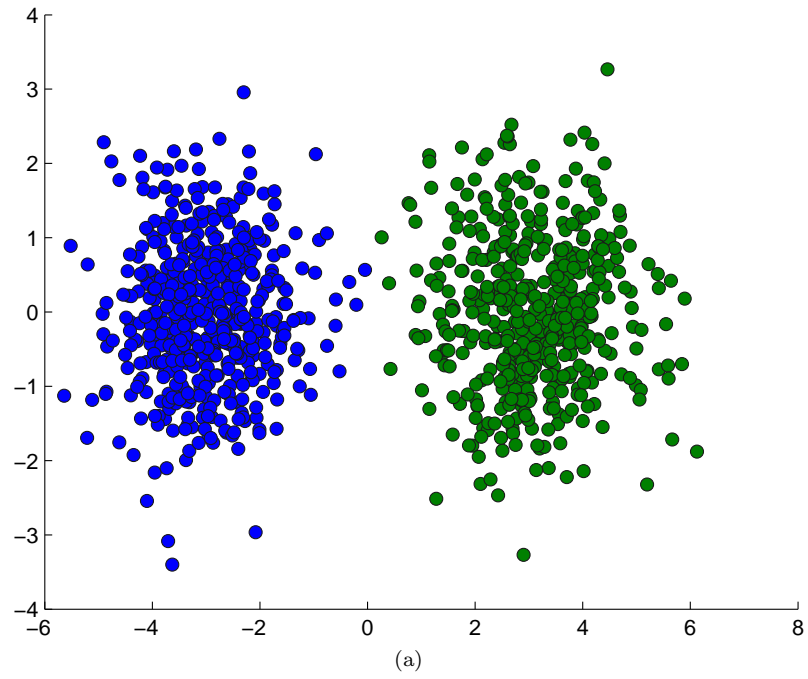


Figura 2.2: Resultados de dos enfoques distintos de clasificación no supervisada: (a) *clustering*; (b) cuantización vectorial. Las líneas identifican la frontera de decisión entre *clusters* y forman divisiones del espacio denominadas celdas de Voronoi [3]. La forma de representar el resultado es distinta en cada caso: en cuantización vectorial se buscan los prototipos que mejor representen a los datos; en *clustering* se busca una clasificación global de los datos y no necesariamente existen prototipos

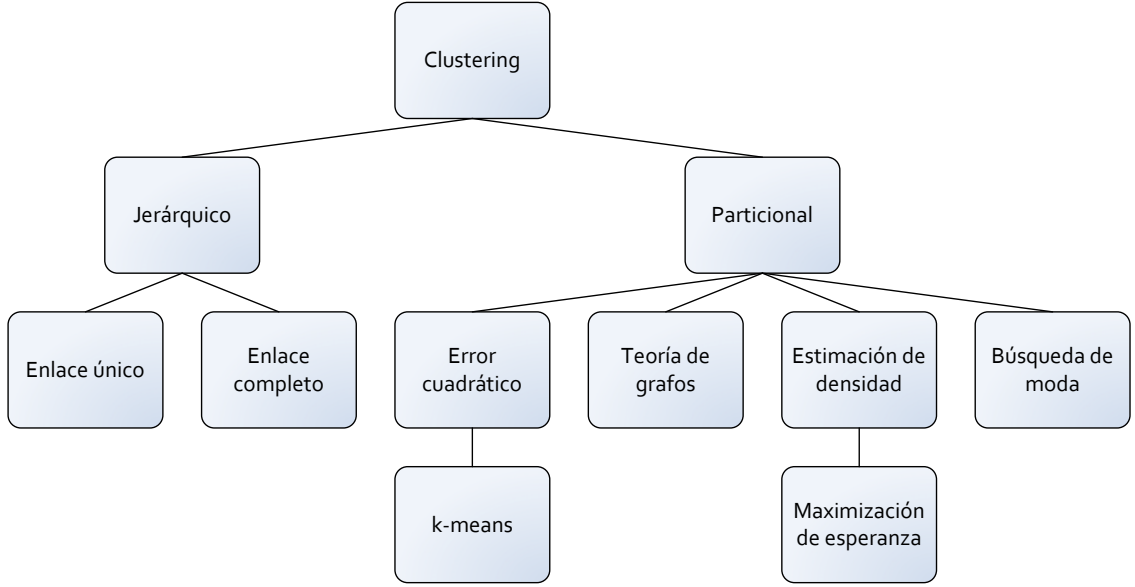


Figura 2.3: Clasificación de métodos de *clustering* según [40]

Algoritmos particionales

A diferencia de los métodos jerárquicos, los algoritmos particionales producen una sola partición de los datos. Formalmente, el conjunto de particiones que pueden formar estos algoritmos está dado por:

$$M_h = \left\{ U \in \mathbb{R}^{nc} : u_{ik} \in \{0, 1\} \ \forall (i, k); \sum_{k=1}^c u_{ik} = 1 \ \forall i; 0 < \sum_{i=1}^n u_{ik} < n \ \forall k \right\} \quad (2.3)$$

donde U se denomina la matriz de partición o de pertenencia y cada componente u_{ik} define la pertenencia del patrón \mathbf{x}_i al *cluster* C_k . Las restricciones impuestas para el conjunto M_h definen particiones “duras” (*hard* o *crisp* en inglés), donde los *clusters* son disjuntos (un patrón pertenece a un sólo *cluster*) y no puede haber *clusters* vacíos. Los algoritmos particionales generalmente están basados en centroides, que corresponden a vectores prototipo que se ocupan como representantes de todos los patrones de un *cluster*. Un método basado en centroides sólo utiliza las distancias entre los patrones y sus respectivos prototipos, por lo cual presenta ventajas en aplicaciones que involucren conjuntos de datos relativamente grandes, donde la construcción de un dendrograma puede ser muy costoso. Una gran desventaja es que generalmente es necesario especificar previamente el número de *clusters* deseados. Las técnicas particionales usualmente operan optimizando un funcional de

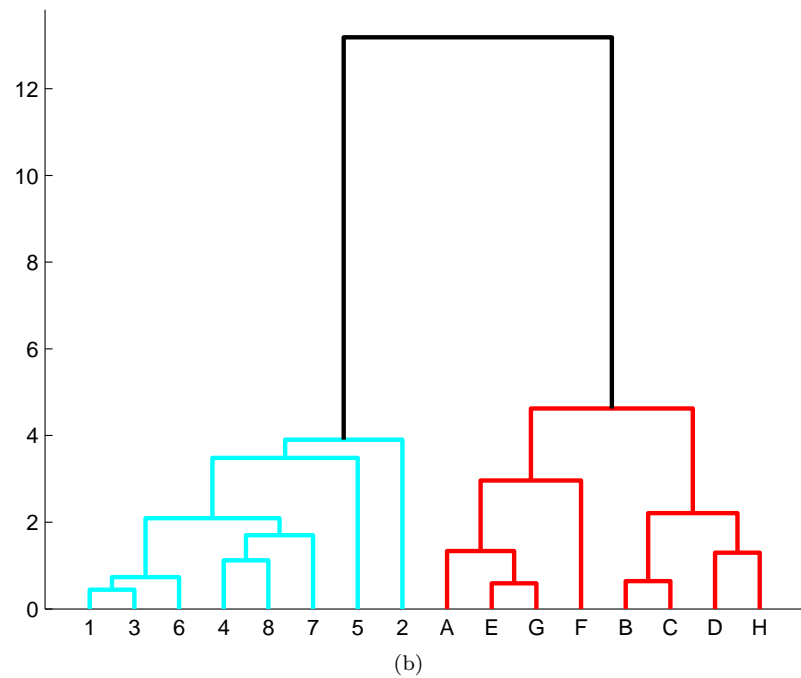
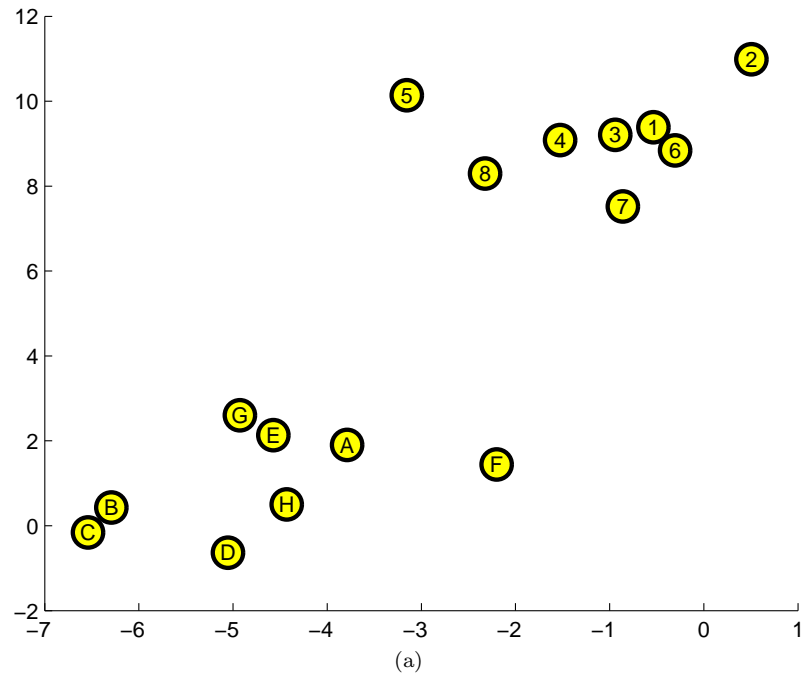


Figura 2.4: Ejemplo del resultado de un método de *clustering* jerárquico: (a) un conjunto de datos simple; (b) su respectivo dendrograma, obtenido utilizando el método de enlace completo

costos definido ya sea localmente (en un subconjunto de los patrones) o globalmente (sobre todos los patrones). La búsqueda combinatorial sobre todo el espacio de particiones posibles (M_h) es claramente infactible en la práctica ($\#\{M_h\} \approx c^k$). Es por esto que típicamente un algoritmo particional es ejecutado múltiples veces con diferentes condiciones iniciales y la mejor configuración obtenida en todas las ejecuciones es designada como el resultado del método.

Error cuadrático El criterio más intuitivo y más frecuentemente usado como función de costos para algoritmos particionales es el error cuadrático. Esta medida estima el error que se comete al representar a los patrones de cada *cluster* por su centroide y tiende a funcionar bien cuando los *clusters* son compactos, isotrópicos (hiperesféricos) y están aislados entre sí. De forma implícita estos métodos son paramétricos, ya que el procedimiento es equivalente a asumir una distribución gaussiana para cada *cluster* y buscar la media de cada distribución, que corresponde al centroide de cada *cluster*. El funcional genérico de error cuadrático es el siguiente:

$$J_1(U, V) = \sum_{k=1}^c \sum_{i=1}^n (u_{ik}) D(\mathbf{v}_k, \mathbf{x}_i)^2 \quad (2.4)$$

donde $V = [\mathbf{v}_1, \dots, \mathbf{v}_c]$ es la matriz cuyas columnas son los centroides de cada *cluster*.

Algoritmo *k-means* El método más simple y difundido de los que ocupan el criterio del error cuadrático es el algoritmo *k-means* [45]. El método comienza con una partición inicial aleatoria de los datos, y continúa reasignando los patrones a cierto *cluster* basado en la similitud entre el patrón y los centroides, hasta que se cumpla cierto criterio de detención (por ejemplo, que no haya reasignación de un patrón desde un *cluster* a otro, o que el error cuadrático no descienda significativamente por un número de iteraciones). El algoritmo *k-means* es popular porque es muy simple de implementar y porque su complejidad computacional es $O(n)$, donde n es el número de patrones. Un problema serio con este método es su sensibilidad a la partición inicial escogida, ya que puede converger fácilmente a un mínimo local del funcional de costos si la partición inicial es deficiente. El procedimiento exacto que aplica el algoritmo *k-means* es el siguiente:

1. Elegir c centroides aleatoriamente, ya sea haciéndolos coincidir con c patrones cualesquiera, o definiéndolos al azar dentro del hipervolumen que contiene al conjunto de patrones.
2. Asignar cada patrón al *cluster* representado por el centroide más cercano.

3. Recalcular los centroides como las medias de todos los patrones que fueron asignados a sus respectivos *clusters*.
4. Repetir desde el punto 2 hasta que se cumpla algún criterio de detención.

Estimación de densidad y búsqueda de la moda

Los métodos de estimación de densidad se basan principalmente en la mezcla de distribuciones. Estos métodos están orientados en la línea clásica de reconocimiento de patrones, donde se asume que los datos dentro de cada *cluster* provienen de una distribución de probabilidad. En el caso de mezcla de distribuciones, se trata de algoritmos paramétricos, ya que se asume que cada distribución corresponde a una suma ponderada de varias distribuciones (generalmente gaussianas) y se deben estimar tanto sus parámetros como los coeficientes de mezcla (ponderadores de cada distribución). Un enfoque clásico consiste en aplicar el método de maximización de la esperanza (EM, Expectation Maximization), que es un algoritmo de propósito general de máxima verosimilitud con datos incompletos [49], pero adaptado al análisis de *clusters*. El método comienza con una estimación inicial de los parámetros e iterativamente reasigna los valores de pertenencia de los patrones a cada *cluster*, a través de la mezcla de distribuciones que produce el vector de parámetros. Luego, se actualiza la estimación de los parámetros ocupando los patrones con sus nuevos grados de pertenencia.

Vecino más cercano Dado que la proximidad entre patrones juega un rol trascendental en la noción intuitiva que se tiene de un *cluster*, las distancias a los vecinos más cercanos pueden usarse como base para métodos de *clustering*. El funcionamiento básico de este tipo de algoritmos consiste en asignar cada patrón al mismo *cluster* de su vecino más cercano, mientras la distancia entre ellos no supere cierto umbral. Este proceso iterativo puede comenzar desde un patrón aleatorio y termina cuando todos los patrones sean asignados a un *cluster*, o cuando no sea posible realizar más asignaciones.

Clustering difuso

Los algoritmos de *clustering* tradicionales generan particiones “duras” o disjuntas, donde cada patrón pertenece a un solo *cluster*. Los algoritmos de *clustering* difuso extienden esta noción al asociar cada patrón con cada *cluster* a través de una función de pertenencia [50]. El resultado de

un algoritmo difuso se denomina partición difusa, y el conjunto de resultados posibles se obtiene relajando las restricciones impuestas en (2.3) para particiones disjuntas:

$$M_f = \left\{ U \in \mathbb{R}^{nc} : u_{ik} \in [0, 1] \ \forall (i, k); \sum_{k=1}^c u_{ik} = 1 \ \forall i; 0 < \sum_{i=1}^n u_{ik} < n \ \forall k \right\} \quad (2.5)$$

El método más representativo de esta familia de algoritmos de *clustering* es *fuzzy c-means* [50].

Una descripción de alto nivel de un algoritmo genérico de *clustering* difuso sería la siguiente:

1. Elegir una partición difusa inicial de los n patrones en c *clusters*, que implica definir la matriz de pertenencia U , de tamaño $n \times c$. Tal como en el caso disjunto, cada elemento u_{ik} de U representa el grado de pertenencia del patrón \mathbf{x}_i al *cluster* C_k , sólo que ahora se tiene $u_{ik} \in [0, 1]$.
2. Usando U , encontrar el valor de una función objetivo difusa asociado a la partición. Reasignar los patrones a los distintos *clusters* de manera de reducir el valor de la función objetivo y recalcular U . Una función objetivo genérica para el análisis de *clusters* difusos es

$$J_m(U, V) = \sum_{k=1}^c \sum_{i=1}^n (u_{ik})^m D(\mathbf{v}_k, \mathbf{x}_i)^2 \quad (2.6)$$

donde m es un parámetro suministrado por el usuario que indica el grado de “difusidad” (*fuzzyness* en inglés) de las particiones.

3. Repetir el paso 2 hasta que los valores de U no cambien significativamente. En *clustering* difuso, cada *cluster* corresponde a un conjunto difuso de todos los patrones.

Otro enfoque relacionado es el denominado *clustering* “posibilístico” [51], donde las restricciones que definen el conjunto de particiones posibles se relajan aún más, aceptando que las pertenencias de un mismo patrón a todos los *clusters* no sumen 1:

$$M_p = \left\{ U \in \mathbb{R}^{nc} : u_{ik} \in [0, 1] \ \forall (i, k); 0 < \sum_{i=1}^n u_{ik} < n \ \forall k \right\} \quad (2.7)$$

Es común en este ámbito que al enfoque difuso clásico se le refiera como probabilístico, ya que las pertenencias, al sumar 1, pueden interpretarse como probabilidades. Recientemente, también se han desarrollado métodos que combinan ambos enfoques [52].

2.2.2. Validez del *clustering*

El análisis de *clusters* es un proceso no supervisado. De por sí, esto implica que no existe una forma intrínseca de validar los resultados: en una aplicación real no existen etiquetas con las cuales comparar la partición generada. Tampoco es de utilidad utilizar la misma función objetivo que fue optimizada por el algoritmo a modo de validación: la elección de la función objetivo determinará el tipo de *clustering* generado y el hecho que el algoritmo efectivamente logre encontrar un óptimo no implica que los resultados obtenidos sean útiles (es por esto que se habla de la definición circular del problema de *clustering*). Si no se realiza una validación de los resultados, el usuario de un algoritmo de *clustering* puede cometer tres tipos de errores: forzar una estructura de *clusters* a un conjunto de datos que en realidad no la tiene, buscar un número de *clusters* muy distinto al que existe en la realidad, o aceptar un resultado subóptimo o simplemente erróneo entregado por el algoritmo.

En el caso en que los datos sean visualizables (en dos o tres dimensiones), la validación de los resultados puede hacerse a través de un análisis visual de los *clusters* resultantes por parte del usuario. Sin embargo, es claro que este proceso de validación no es automático, repetible ni objetivo. En caso que los datos no sean visualizables directamente, pueden usarse técnicas de visualización que proyecten los datos a un espacio de dos o tres dimensiones [44]. Esta alternativa, además de las ya mencionadas desventajas de la validación visual, requiere confiar además en el resultado del método de proyección. En definitiva, surge la necesidad de contar con un método de validación automática y objetiva del resultado de un algoritmo de *clustering* [6]. Muchas propuestas para solucionar esto se pueden encontrar en la literatura [53]. En general, los métodos propuestos se reducen a calcular un índice de validez, es decir, un valor que indica cuan válido es el resultado. Existen tres criterios posibles para definir estos índices de validez [3, 53]: criterios externos, internos y relativos.

Validez externa

Un índice de validez externa compara dos particiones distintas de los datos y trata de medir cuánto se parecen o en qué medida concuerdan dichas particiones. Ya que se trata de validar el resultado de un método de *clustering*, las particiones a comparar son la que entregue el método y la partición ideal, que se determina a través de las etiquetas asignadas por un experto. Se denomina validez “externa” ya que se necesita esta información adicional a los datos para realizar la compara-

ción. Si $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_s\}$ es la partición ideal (etiquetas) y $C = \{C_1, \dots, C_m\}$ es la partición (el *clustering*) generada por el método, entonces se pueden usar los siguientes términos para referirse a un par de patrones $\{\mathbf{x}_i, \mathbf{x}_j\}$ del conjunto de datos:

- SS: si ambos patrones pertenecen al mismo *cluster* en C y al mismo grupo en \mathcal{P} .
- SD: si los patrones pertenecen al mismo *cluster* en C , pero a diferentes grupos en \mathcal{P} .
- DS: si los patrones están en distintos *clusters* en C , pero en el mismo grupo en \mathcal{P} .
- DD: si los patrones están en distintos *clusters* en C y en distintos grupos en \mathcal{P} .

Nótese que no necesariamente se tiene que $s = m$, es decir, el número de grupos en \mathcal{P} puede ser distinto al número de *clusters* en C . Supóngase que a , b , c y d corresponden al número de pares tipo SS, SD, DS y DD, respectivamente, con lo que se tiene que $a + b + c + d = M$ (que corresponde al total de pares que se pueden formar con los datos, es decir, $M = n(n-1)/2$). Se definen los siguientes índices de validez, que indican la similitud entre C y \mathcal{P} :

- Rand [54]:

$$R = \frac{a + d}{M} \quad (2.8)$$

- Jaccard:

$$J = \frac{a}{a + b + c} \quad (2.9)$$

- Fowlkes y Mallows:

$$FM = \frac{a}{\sqrt{m_1 m_2}} = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}} \quad (2.10)$$

donde $m_1 = a + b$ es el número de pares que pertenecen al mismo *clusters* en C , y $m_2 = a + c$ es el número de pares que pertenecen al mismo grupo en \mathcal{P} .

Para los tres índices anteriores, un valor más alto indica una mayor correspondencia entre C y \mathcal{P} . Otro índice muy popular es el estadístico Γ de Hubert, diseñado para comparar la similitud entre dos matrices cualesquiera X e Y .

- Estadístico Γ de Hubert [55]:

$$\Gamma = \frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{i,j} Y_{i,j} \quad (2.11)$$

El estadístico Γ se usa para comparar dos particiones asignando $X_{i,j} = 1$ si $\{\mathbf{x}_i, \mathbf{x}_j\}$ están en el mismo *cluster* en C y $X_{i,j} = 0$ en caso contrario. Lo mismo se hace con $Y_{i,j}$ considerando los grupos de \mathcal{P} . Valores muy grandes de Γ indican una alta similitud entre las particiones. Existe también una versión normalizada de Γ , cuyos valores fluctúan entre -1 y 1 :

- Estadístico Γ normalizado:

$$\hat{\Gamma} = \frac{\frac{1}{M} \sum_{i=1}^{n-1} \sum_{j=i+1}^n (X_{i,j} - \mu_X)(Y_{i,j} - \mu_Y)}{\sigma_X \sigma_Y} \quad (2.12)$$

donde $\{\mu_X, \sigma_X\}$ y $\{\mu_Y, \sigma_Y\}$ son las medias y varianzas de las matrices X e Y , respectivamente.

Una estructura de *clusters* se considera “válida” si es que es “inusual” en cierta forma [6]. Esto significa que la probabilidad de obtener la partición analizada por mero azar es baja. Un índice de validez externa debiera estimar la probabilidad de que la partición analizada sea la correcta. Los estadísticos mostrados anteriormente no reflejan esta propiedad, ya que, aunque en todos existe un máximo si la partición analizada es idéntica a la ideal, pueden presentar valores cercanos al máximo para particiones completamente aleatorias. Es por esto que es necesario ajustar los índices de manera que se obtenga un valor bajo para una partición cualquiera, muy distinta de la ideal. Existen versiones ajustadas de todos los índices anteriores y se han formulado distintas hipótesis para realizar el ajuste de cada uno. De estos, el índice más aceptado en la literatura de *clustering* es el índice de Rand ajustado, propuesto en [55]:

$$R' = \frac{\left[\frac{a+d}{M}\right] - E\left[\frac{a+d}{M}\right]}{1 - E\left[\frac{a+d}{M}\right]} = \frac{\sum_i \sum_j \binom{n_{ij}}{2} - \frac{1}{M} \sum_i \binom{n_{i\cdot}}{2} \sum_j \binom{n_{\cdot j}}{2}}{\frac{1}{2} \left[\sum_i \binom{n_{i\cdot}}{2} + \sum_j \binom{n_{\cdot j}}{2} \right] - \frac{1}{M} \sum_i \binom{n_{i\cdot}}{2} \sum_j \binom{n_{\cdot j}}{2}} \quad (2.13)$$

donde n_{ij} corresponde al número de patrones asignados al *cluster* C_i en C y al grupo \mathcal{P}_j en \mathcal{P} , $n_{i\cdot}$ es el número de patrones asignados al *cluster* C_i en C , independiente de su grupo en \mathcal{P} , y $n_{\cdot j}$ es el número de patrones asignados al grupo \mathcal{P}_j en \mathcal{P} , independiente de su *cluster* en C .

Dado que se trata de medir el grado de aleatoriedad de la partición generada respecto a la partición ideal, resulta adecuado (como se explica en la sección 2.7) usar conceptos de teoría de la información. Recientemente han aparecido publicaciones que buscan explotar estas herramientas para la validez externa de resultados de *clustering* [56].

Los criterios de validación externa son útiles cuando se dispone de una partición de referencia. Debido a esto, en la práctica su uso se limita a evaluar el desempeño de un algoritmo de *clustering*

en conjuntos de datos tipo benchmark, es decir, aquellos para los que se dispone de etiquetas previamente asignadas por un experto. Este análisis es de utilidad para caracterizar el comportamiento de un método en conjuntos de datos con diferentes estructuras de *clusters*. Aplicando este análisis a varios métodos de *clustering* se puede concluir cuáles son los más adecuados para cada tipo de estructura. Sin embargo, en una aplicación real de *clustering*, no se dispone de etiquetas previas, y en este caso los índices de validez externa no son de utilidad.

Validez interna

Los criterios internos miden la validez ocupando sólo los datos y el resultado del método de *clustering* (es decir, la matriz de pertenencia). En general, se trata de medir el grado en que se cumplen los conceptos básicos que busca optimizar un método de *clustering*:

- Compacidad (*compactness*). Los miembros de cada *cluster* debieran estar lo más cerca posible entre ellos. Una medida comúnmente utilizada para medir la compacidad es la varianza.
- Separación. Los *clusters* entre ellos debieran estar lo más espaciados posible. Para medir la distancia entre dos *clusters* se utilizan generalmente alguno de los siguientes conceptos:
 - Enlace único: Mide la distancia entre los miembros más cercanos de ambos *clusters*.
 - Enlace completo: Mide la distancia entre los miembros más distantes.
 - Comparación de centroides: Mide la distancia entre los centros de ambos *clusters*.

Se han definido muchos índices de validez interna en la literatura de *clustering*. Algunos de los más populares son:

- Davies–Bouldin [57]. Se calcula la varianza muestral de cada *cluster*, cuyo centroide (media) es \mathbf{v}_k , como

$$S_k = \frac{1}{n_k} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \mathbf{v}_k\|^2 \quad (2.14)$$

y la distancia entre dos *clusters* como

$$d_{kk'} = \|\mathbf{v}_k - \mathbf{v}_{k'}\|^2 \quad (2.15)$$

entonces se define

$$R_k = \max_{k, k' \neq k} \left\{ \frac{S_k + S_{k'}}{d_{kk'}} \right\} \quad (2.16)$$

con lo que finalmente el índice Davies–Bouldin es:

$$DB = \frac{1}{c} \sum_{k=1}^c R_k \quad (2.17)$$

- Dunn [58]. Se define la distancia entre dos *clusters* como:

$$\delta(C_k, C_{k'}) = \min_{\mathbf{x}_i \in C_k, \mathbf{x}_j \in C_{k'}} D(\mathbf{x}_i, \mathbf{x}_j) \quad (2.18)$$

y el diámetro de un *cluster* como:

$$\Delta(C_k) = \max_{\mathbf{x}_i, \mathbf{x}_j \in C_k} D(\mathbf{x}_i, \mathbf{x}_j) \quad (2.19)$$

entonces el índice de Dunn es:

$$V_D = \min_k \left\{ \min_{k' \neq k} \left\{ \frac{\delta(C_k, C_{k'})}{\max_{k''} \{\Delta(C_{k''})\}} \right\} \right\} \quad (2.20)$$

Bezdek y Pal [59] han definido generalizaciones de este índice que son usadas con frecuencia.

- PBM [60, 61]. Se define como:

$$PBM = \left(\frac{1}{c} \times \frac{E_1}{E_k} \times D_k \right)^2 \quad (2.21)$$

donde $E_k = \sum_{i=1}^c \sum_{i=1}^{n_k} \|\mathbf{x}_i - \mathbf{v}_k\|^2$, $D_k = \max_{k, k'=1}^c \|\mathbf{v}_k - \mathbf{v}_{k'}\|^2$, n_k es el número de patrones en el *cluster* C_k , y \mathbf{v}_k es el centroide del *cluster* C_k .

Como se puede apreciar de su definición, todos estos índices de validez interna tratan de medir qué tan bien se ajusta la solución obtenida a una estructura definida a priori. A pesar de que no se diga explícitamente, la elección de las medidas de distancia al interior y entre pares de *clusters* implican la suposición de que los *clusters* son gaussianos (hiperesféricos o hiperelipsoidales). Cabe recordar el problema de la definición circular del *clustering*: el algoritmo busca la solución óptima asumiendo cierta estructura; para validar el resultado, se mide la correspondencia entre la solución

y el mismo tipo de estructura. En el caso en que los datos posean una estructura diferente, el proceso de validación no será capaz de detectarlo.

Si se tuviera un índice confiable para validar el resultado de un algoritmo de *clustering*, sería posible utilizar dicho índice directamente como función objetivo para algún método de optimización. Varios autores han utilizado métodos evolutivos que tratan de optimizar algunos de los índices presentados anteriormente, o combinaciones de ellos [60, 62, 63]. Claramente, dichos métodos sólo podrán encontrar *clusters* que cumplan con el criterio utilizado para definir la función objetivo, en este caso sólo *clusters* gaussianos. Recientemente se han propuesto nuevas enfoques para la validez interna de soluciones de *clustering* basadas en teoría de la información [7, 9, 10, 13]. Como se ve en la sección 2.7, la teoría de la información trata de medir el grado de estructura presente en los datos (en este caso, en una partición de los datos), sin asumir una estructura a priori.

Validez relativa

Los criterios de validez relativa están orientados a seleccionar la mejor configuración de parámetros para un algoritmo de *clustering*. La idea básica consiste en ejecutar reiteradamente el mismo algoritmo utilizando distintas configuraciones y seleccionar aquella que obtenga un mejor índice de validez. Como se vio anteriormente, los algoritmos particionales normalmente requieren que el usuario ingrese como un parámetro el número de *clusters* a buscar. En muchas aplicaciones, este número es desconocido y ni siquiera se tiene una idea de su posible valor. Sería de utilidad un método que logre estimar la cantidad de *clusters* presentes en los datos. Muchos autores han utilizado los índices de validez relativa para este efecto. De hecho, en muchas publicaciones, el concepto de validez del *clustering* se ocupa para referirse al procedimiento de estimar el número de *clusters*. La metodología es la misma utilizada para seleccionar la mejor configuración de parámetros, sólo que en este caso el único parámetro a considerar es el número de *clusters* (c). Específicamente, se ejecuta un algoritmo de *clustering* particional varias veces, especificando cada vez un mayor valor de c , dentro de un rango de valores predefinido. Para cada uno de las particiones obtenidas se calcula el índice de validez. Dado que los resultados pueden variar entre ejecuciones con los mismos parámetros, para cada valor de c se ejecuta varias veces el algoritmo y se calcula la media o el valor máximo del índice de validez. Con estos datos, se construye una curva que muestra la relación entre el índice de validez y c . Del análisis de esta curva se puede estimar el mejor valor de c , utilizando alguno de los siguientes criterios:

- Si el índice de validez utilizado no exhibe un comportamiento creciente ni decreciente al aumentar c , entonces se debe buscar el máximo (o el mínimo, según corresponda) valor del índice.
- En el caso que el índice de validez sea creciente o decreciente al aumentar el valor de c , se debe buscar un punto de la curva donde se produzca un cambio abrupto en el valor del índice. Este punto es llamado comúnmente “codo” o “rodilla” (*knee* en inglés) en la literatura de *clustering*.

La ausencia de un máximo o de un “codo” puede interpretarse como un indicio de que los datos no poseen estructura de *clusters*.

2.3. Programación genética

Programación genética (PG) [15] es una técnica para la generación automática de programas computacionales basada en los principios de la evolución de las especies según Darwin, especialmente en el concepto de supervivencia del más apto. PG se formula como una variante de los algoritmos genéticos [64–66] y se considera parte de la familia de algoritmos evolutivos. En programación genética los individuos que evolucionan son programas computacionales que representan una posible solución al problema que se está tratando de resolver. El tamaño, la forma y el contenido de estos programas pueden cambiar dinámicamente durante el proceso de evolución. Existen diversas formas para representar dichos programas, pero la más común es la utilizada por Koza [15], que consiste en una combinación jerárquica de nodos en forma de árboles. Es por esto que es usual que a los individuos en PG se les llame simplemente árboles.

En esta sección se realiza una descripción introductoria a PG. Para información más detallada, se sugiere al lector consultar el primer libro de Koza sobre el tema [15] y las referencias [67–69], que hacen una revisión general del método y sus aplicaciones.

Los nodos de un árbol se clasifican en nodos internos y nodos terminales. Los nodos internos representan funciones y tienen subnodos que se desprenden de ellos, comúnmente llamado nodos *hijo*, los que corresponden a los argumentos de la función. Por lo tanto, para obtener el resultado de un nodo interno es necesario evaluar primero a sus nodos hijo, los cuales a su vez pueden también ser nodos internos y tener sus propios nodos hijo. Este procedimiento de evaluación es recursivo y

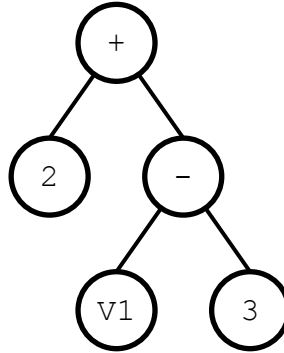


Figura 2.5: Ejemplo de un programa representado como árbol en PG

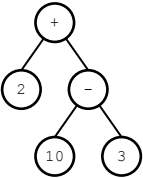
culmina cuando se llega a un nodo terminal, que es aquel que no tiene nodos hijo. Así, el resultado o salida de un programa se obtiene evaluando recursivamente el árbol desde el nodo raíz hasta los terminales. Los terminales pueden ser vistos como las entradas al programa computacional que se creará mediante PG, mientras que la salida del programa corresponde al valor retornado por el nodo raíz.

El conjunto de funciones y terminales disponibles para formar los árboles debe ser especificado por el usuario de acuerdo al problema que se quiere resolver. Ejemplos de posibles funciones son las operaciones aritméticas básicas, funciones trigonométricas, exponenciales, booleanas, etc. Ejemplos de terminales son constantes aleatorias y variables de interés del problema. El espacio de búsqueda de PG será el conjunto de todos los programas que se pueden formar con los nodos disponibles, es decir, todas las combinaciones posibles entre nodos funciones y terminales, mientras estas combinaciones cumplan con ciertas restricciones que pueda imponer el usuario, como por ejemplo un tamaño máximo para los árboles.

En la figura 2.5 se puede ver un ejemplo de un árbol simple. En este caso, al evaluar recursivamente el programa representado por el árbol se obtiene la fórmula $f = 2 + v_1 - 3$, donde v_1 corresponde a una variable de interés del problema. Usualmente el árbol se evaluará en distintos casos de prueba especificados por el usuario, cada uno con un distinto valor para v_1 y las demás variables de interés que pudieran existir.

Otra forma de entender la representación de los individuos en PG es tratarlos como expresiones-S, que corresponden al tipo de instrucciones utilizadas en el lenguaje de programación LISP, originalmente usado por Koza para desarrollar PG. Una expresión-S puede contener dos tipos de elementos:

Tabla 2.1: Representación infijo, prefijo y tipo árbol para la misma expresión

| Notación infijo | Expresión-S (notación prefijo) | Representación tipo árbol |
|-----------------|--------------------------------|---|
| $2 + (10 - 3)$ | $(+ 2(- 10 3))$ |  |

listas y átomos. Los átomos son elementos unitarios cuyo valor es independiente, aunque puede ser variable. La constante 7 y la variable `TIME` son ejemplos de átomos en LISP. Una lista, en cambio, corresponde a un conjunto ordenado de elementos rodeados por un par de paréntesis. Ejemplos de listas son `(A B C D)` y `(+ 1 2)`.

Las expresiones-S son interpretadas por LISP de la siguiente manera: los átomos constantes son evaluados como el valor representado por el átomo (por ejemplo, 7) y los átomos variables son evaluados como el valor actual de la variable representada por el átomo (por ejemplo, `TIME` se evalúa como la hora actual). Una lista, en cambio, se evalúa interpretando el primer elemento como una función y luego aplicando dicha función a los restantes elementos de la lista. Es decir, los demás elementos son evaluados primero y luego pasados como argumentos a la función. Por ejemplo, el resultado de evaluar la expresión-S `(+ 1 2)` es 3. La expresión-S anterior se puede reescribir como $1 + 2$ en notación infijo, que es la notación clásica usada en matemáticas. La notación usada por LISP es una variación de la notación prefija o también llamada notación polaca.

La transformación entre una expresión-S y un árbol es directa: el primer elemento de una lista corresponde a un nodo interno y los demás elementos a sus nodos hijo. Cada elemento de una lista puede ser a su vez otra lista, lo que significa que los nodos hijo también son internos y tienen sus propios nodos hijo. Cuando se encuentre un átomo en una expresión, corresponderá a un nodo terminal en el árbol. La tabla 2.1 muestra una expresión matemática simple y su equivalente en cada una de las representaciones.

En esta tesis se utiliza más frecuentemente la representación tipo árbol que las expresiones-S. Asimismo, se habla más de nodos, funciones y terminales, que de listas y átomos. La representación tipo árbol es más útil para introducir conceptos como mutación y recombinación, así como los individuos multiárbol que se presentan más adelante para la resolución de problemas de *clustering*.

Una simulación de PG (usualmente referida como evolución) comienza con una población de árboles generados aleatoriamente utilizando los nodos disponibles. Cada árbol es evaluado en los casos de prueba especificados y se le asigna un valor de *fitness* según qué tan bien resuelve el problema en cuestión. Los casos de prueba y la medida de *fitness* serán dependientes del problema y deben ser especificados por el usuario. De acuerdo al *fitness* asignado, se selecciona un grupo de árboles que actuarán como padres para la siguiente generación. Aquellos árboles con un mejor *fitness* tienen mayores probabilidades de ser elegidos. A los árboles seleccionados como padres se les aplican operadores genéticos que los modifican (mutación) y combinan entre ellos (recombinación o crossover). Los árboles resultantes de estas operaciones formarán la siguiente generación. Este proceso se repite hasta que se satisfaga alguna condición de término.

En resumen, el paradigma de programación genética evoluciona programas computacionales ejecutando los siguientes tres pasos:

1. Generar una población inicial de programas, a través de composiciones de las funciones y terminales especificados por el usuario.
2. Iterativamente realizar los siguientes pasos hasta que se satisfaga el criterio de término:
 - a) Ejecutar cada programa en la población y asignarle un *fitness* de acuerdo a qué tan bien resuelve el problema.
 - b) Crear una nueva población de programas aplicando los siguientes operadores genéticos. Las operaciones se aplican a los programas ya existentes, los cuales son elegidos con una probabilidad basada en su *fitness*.
 - 1) Copiar el programa ya existente en la nueva población.
 - 2) Crear dos nuevos programas a través de la recombinación de partes elegidas al azar de dos programas ya existentes.
3. El mejor programa que haya aparecido en alguna generación (denominado el mejor hasta ahora) se designa como el resultado de programación genética. Este resultado puede ser una solución (o una solución aproximada) al problema.

El diagrama de flujo original presentado por Koza se muestra en la figura 2.6. Este diagrama sólo considera modificaciones a los árboles a través del operador crossover (recombinación). Posteriormente, Koza también introducirá el operador de mutación.

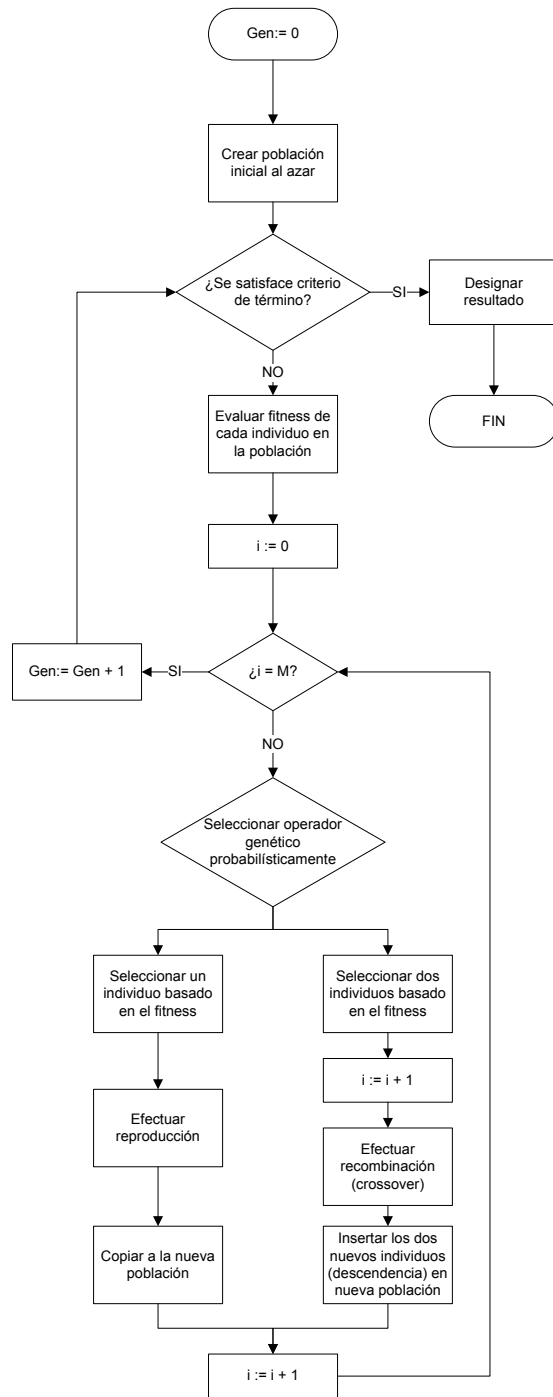


Figura 2.6: Diagrama de flujo de PG [15]

Este esquema de funcionamiento es genérico y se aplica de manera idéntica para cualquier problema a resolver con PG. Sin embargo, para poder aplicar PG a un problema en particular, el usuario debe ejecutar los siguientes paso preparatorios:

- Determinar el conjunto de terminales.
- Determinar el conjunto de funciones base.
- Determinar la medida de *fitness*.
- Determinar los parámetros que controlan la evolución.
- Determinar el método para designar el resultado y el criterio para terminar la evolución.

Esta etapa definitoria es la única en la que PG demanda un esfuerzo por parte del usuario, quien debe aplicar sus conocimientos acerca del problema para ejecutar correctamente cada paso preparatorio. Una vez realizadas estas definiciones, el método se ejecuta de acuerdo al diagrama de la figura 2.6 y se obtiene como resultado una solución (probablemente aproximada) al problema. Sin embargo, se debe recalcar que la correcta determinación de cada paso preparatorio es crucial para el éxito de la optimización mediante PG.

2.3.1. Operadores genéticos

El bloque encargado de modificar los individuos para generar su descendencia se denomina operador genético. PG hereda los operadores más comunes utilizados en algoritmos genéticos, como son la recombinación o *crossover* y la mutación. Estos operadores mezclan material genético entre dos individuos (recombinación) o generan nuevo material de forma aleatoria (mutación). La descendencia generada por este proceso no necesariamente superará en *fitness* a sus progenitores. Será el proceso de selección de la siguiente generación el encargado de evaluar la adaptación de los nuevos individuos. Si las modificaciones fueron exitosas, tendrán un valor elevado de *fitness* y se propagarán. A pesar de conservar el nombre de sus contrapartes en algoritmos genéticos, los operadores de *crossover* y mutación deben ser redefinidos para su aplicación en PG, dado el cambio en la representación de los individuos de tiras binarias a árboles de funciones.

Recombinación

Consiste en intercambiar entre dos individuos ramas seleccionadas aleatoriamente de cada uno de ellos. Este operador se aplica sobre dos individuos (padres) y produce dos nuevos individuos (hijos). El procedimiento de recombinación es el siguiente:

1. Seleccionar un nodo de forma aleatoria en el primer individuo. Este nodo se denomina punto de crossover 1.
2. Seleccionar un nodo de forma aleatoria en el segundo individuo. Este nodo se denomina punto de crossover 2.
3. Reemplazar en el individuo 1 la rama que se desprende del punto de crossover 1 con la rama que se desprende el punto de crossover 2.
4. Reemplazar en el individuo 2 la rama que se desprende del punto de crossover 2 con la rama que se desprende el punto de crossover 1.

En la figura 2.7 se muestra un diagrama explicativo del proceso de recombinación.

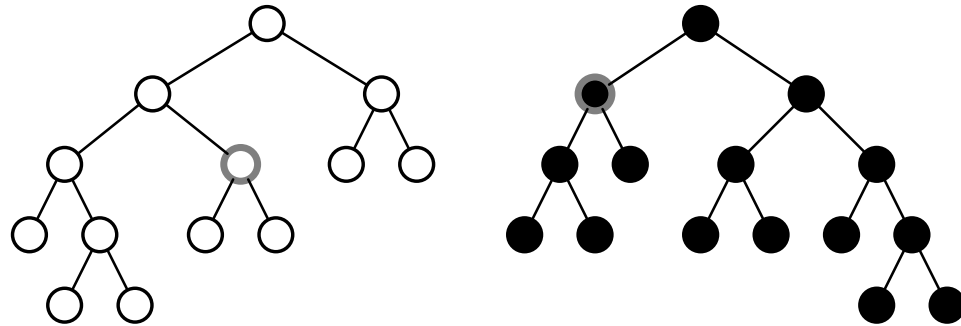
Mutación

Consiste en reemplazar la rama que se desprende de un nodo seleccionado al azar por una nueva rama, generada aleatoriamente. Este operador se aplica sobre un sólo individuo a la vez (padre) y produce un nuevo individuo (hijo). El procedimiento de mutación es el siguiente:

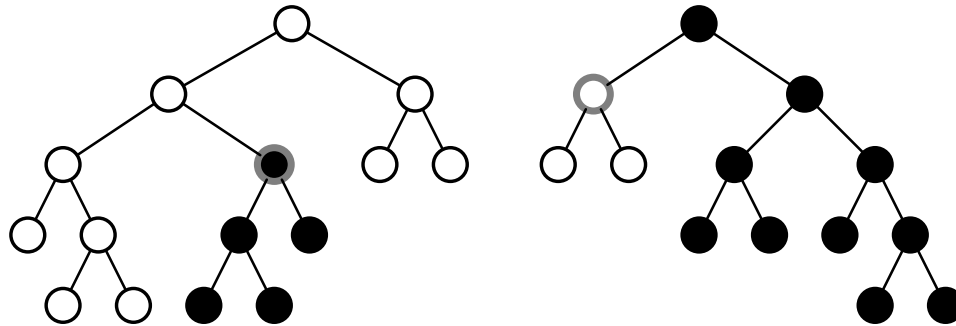
1. Seleccionar un nodo de forma aleatoria. Este nodo se denomina punto de mutación.
2. Generar un árbol aleatoriamente.
3. Reemplazar la rama que se desprende del punto de mutación por el árbol generado aleatoriamente.

En la figura 2.8 se muestra un diagrama explicativo del proceso de mutación.

A esta lista de operadores también se suele agregar el de reproducción, aunque técnicamente no es un operador, ya que lo único que hace es copiar el individuo tal cual está a la siguiente generación. Existen también otros operadores genéticos menos comunes, generalmente modificaciones sutiles de los presentados aquí. En la sección 2.4.3 se describen algunos operadores específicamente orientados al diseño de clasificadores con PG.

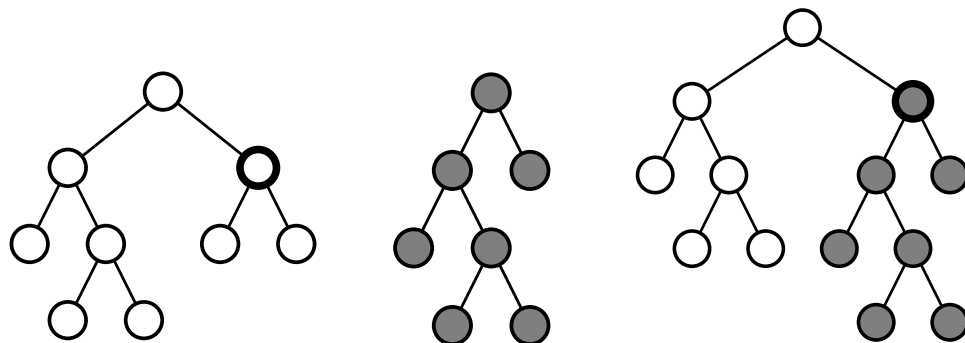


(a)



(b)

Figura 2.7: Operador de recombinación o *crossover*: (a) dos individuos antes de la recombinación; (b) nuevos individuos producidos por el operador. Con borde gris se destacan los puntos de crossover.



(a)

(b)

(c)

Figura 2.8: Operador de mutación: (a) individuo antes de la mutación; (b) nuevo sub árbol generado aleatoriamente; (c) nuevo individuo producido por el operador. Con borde ennegrecido se destaca el punto de mutación

2.3.2. Operadores de selección

Hasta ahora se ha hablado de seleccionar los individuos que pasarán a la siguiente generación de acuerdo a su *fitness*. El bloque encargado de esta función se suele denominar como operador de selección. El rol del proceso de selección no es simplemente elegir a los mejores individuos para que generen descendencia. Si este fuera el caso, la evolución se estancaría rápidamente, ya que es poco probable que en la población inicial se generen individuos que resuelvan satisfactoriamente el problema tratado. Más aún, ni siquiera es probable que exista todo el “material genético” (bloques funcionales) necesario para producir una solución aceptable. Un proceso de selección correcto debiera favorecer con mayor probabilidad aquellos individuos más aptos con respecto a sus pares, pero tratando de mantener la diversidad de la población. Es decir, la selección debe balancear la explotación de las “vetas” ya encontradas con la exploración de otras nuevas [66].

Dada su formulación como una variante de los algoritmos genéticos, PG hereda los mismos operadores de selección comúnmente utilizados en dicha técnica. La selección generalmente se realiza en dos etapas (excepto en selección tipo torneo, que se explica más adelante). Primero se asigna la probabilidad de selección (o equivalentemente, el número de copias esperadas) para cada individuo y luego se realiza el muestreo, que en realidad es la selección propiamente tal. Existen dos opciones comunes para asignar la probabilidad de selección:

Selección proporcional.

La probabilidad de selección de un individuo corresponde a su *fitness* dividido por la suma de los *fitness* de los demás individuos, es decir:

$$P_s(I_i) = \frac{f(I_i)}{\sum_{j=1}^N f(I_j)} \quad (2.22)$$

El problema que tiene esta metodología es que puede asignar una probabilidad de selección demasiado alta a unos pocos individuos altamente adaptados. Los genes de estos individuos pueden rápidamente llegar a dominar la población, causando la convergencia a un máximo local. El crossover no sirve en estas circunstancias y la mutación hace una búsqueda aleatoria muy lenta.

Selección por ranking

En esta variante, propuesta por Baker [65], primero se ordenan los individuos de acuerdo a su *fitness*. Luego, la probabilidad se asigna de acuerdo a la posición que ocupe el individuo en el ordenamiento (ranking), independiente del valor absoluto del *fitness*. Generalmente la probabilidad se asigna aplicando una función lineal de la posición en el ranking:

$$P_s(I_i) = \frac{1}{N} \left[\eta^+ - (\eta^+ - \eta^-) \frac{\text{rank}(I_i) - 1}{N - 1} \right] \quad (2.23)$$

Una vez obtenidas las probabilidades de selección, se procede al muestreo de los individuos. Este proceso se modela generalmente como una ruleta. A cada individuo se le asigna una tajada del área total de la ruleta, proporcional a su probabilidad de selección. Existen dos variantes de ruleta comúnmente usadas:

Ruleta simple

La ruleta posee sólo un puntero. Para seleccionar un individuo se echa a rodar la ruleta y se realizará una copia del individuo cuya tajada sea apuntada. La ruleta se debe girar N veces para producir los N individuos necesarios.

Muestreo estocástico universal (SUS)

La ruleta posee N punteros equiespaciados en ángulo. Al echar a rodar la ruleta se seleccionan los N individuos de una sola vez. Si una tajada es apuntada, entonces el individuo correspondiente recibe una copia. Más de un puntero puede acabar en la misma tajada, por lo que un individuo puede recibir varias copias.

Selección por torneo

Otra forma de selección que no cumple con el paradigma antes descrito de asignación de probabilidad y muestreo es la denominada selección por torneo. El procedimiento es el siguiente:

1. Escoger un tamaño de torneo q .
2. Crear una permutación de los primeros N enteros, donde N es el tamaño de la población.
3. Comparar el *fitness* de los próximos q miembros de la población y seleccionar el mejor.

4. Si se acaba la permutación, generar una nueva permutación.

5. Repetir hasta llenar la población.

Por ejemplo, con $q = 2$, se requieren 2 pasadas para llenar la población. El mejor individuo de la generación se lleva 2 copias, el peor, 0, y el promedio, 1. Se puede usar un tamaño de torneo más grande para aumentar la presión selectiva.

2.3.3. Ejemplo: regresión simbólica con PG

A modo de clarificar algunos conceptos y explicar en detalle la metodología para aplicar PG a un problema particular, se presenta el siguiente ejemplo de aplicación. Muchos otros ejemplos detallados se pueden encontrar en [15].

Un problema común de fácil solución con PG es el de regresión simbólica. Este problema consiste en encontrar una expresión matemática para una función de la cual se conocen sólo los valores de salida para algunos valores de entrada. La forma clásica de resolver este problema es especificar *a priori* un modelo para la función (por ejemplo, lineal o cuadrático) y calcular los coeficientes que minimizan el error entre las salidas del modelo y las de la función real. Es claro que la selección del modelo determinará que tan bien se podrá ajustar la función a los datos existentes. La ventaja de aplicar PG es que no es necesario especificar un modelo, sino sólo el conjunto de funciones base y el método encontrará automáticamente una expresión que se ajuste correctamente a los datos. Más formalmente, un problema de regresión simbólica consiste en que se tiene una función $f : \mathbb{R}^d \rightarrow \mathbb{R}$ desconocida, para la cual se conocen un conjunto de n pares $\{\mathbf{x}_i, y_i\}$, $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, $i \in \{1, \dots, n\}$, donde se cumple que $f(\mathbf{x}_i) = y_i$. Para resolver este problema con PG, se deben aplicar los pasos preparatorios, que en este caso resultan ser en su mayoría bastante directos:

- Conjunto de terminales. Se dijo anteriormente que los terminales corresponden a las entradas para el programa que se quiere generar. En este caso se busca un programa que obtenga una salida igual o similar a la salida deseada para ciertos valores de entrada conocidos. Si los valores de entrada son de dimensión d , es decir, cada caso conocido considera d valores de entrada, es claro que el conjunto de terminales, comúnmente designado con la letra T, debe considerar una variable por cada dimensión de los datos de entrada. En definitiva, se tiene $T = \{v_1, \dots, v_d\}$, donde cada v_l , con $l \in \{1, \dots, d\}$ representa a una variable de entrada. Es

común agregar además un tipo de terminal especial, denominado constante aleatoria efímera. Cada instancia de este tipo de terminal almacena una constante cuyo valor se asigna de manera aleatoria al ser creado y permanece fijo durante la evolución. Generalmente se especifica un rango dentro del cual se escogen valores para las constantes.

- Conjunto de funciones base. Como la naturaleza de la función original es desconocida, no es posible especificar *a priori* un conjunto que contenga todas las funciones base necesarias para recrear la función original y ninguna de más. Sólo se puede especificar un conjunto de funciones que el usuario crea suficientes basado en su conocimiento limitado del problema. Un ejemplo de conjunto de funciones base que a través de composiciones puede representar casi cualquier expresión es $F = \{+, -, *, /, \sin, \cos, \exp, \log\}$.
- Medida de *fitness*. El individuo óptimo es aquel que genera salidas idénticas a las salidas deseadas para cada uno de los casos de prueba. En caso de no encontrar este individuo óptimo, el mejor individuo será el que genere salidas más cercanas a las salidas deseadas. Es claro entonces que el *fitness* debe corresponder a alguna medida de error entre las salidas del árbol y las salidas deseadas, tomando en cuenta todos los casos de prueba. Una posibilidad es usar el error cuadrático medio, definido como $e_{RMS} = \sqrt{\sum_{i=1}^n (y_i - \tilde{y}_i)^2}$, donde \tilde{y}_i corresponde a la salida del árbol para la entrada \mathbf{x}_i .
- Parámetros que controlan la evolución. Se debe especificar el tamaño de la población, largo máximo de los árboles, método de selección, operadores genéticos y sus probabilidades, entre otros parámetros.
- Método para designar el resultado y el criterio para terminar la evolución. Un criterio apropiado para terminar la evolución en este caso sería especificar un máximo de error aceptable. Una vez que se obtenga un individuo que produzca un error inferior al máximo aceptado, se termina la evolución. Se suele especificar además un número máximo de generaciones. Si se alcanza este número máximo sin encontrar una solución aceptable, se termina la evolución. Por último, el método más común para designar el resultado es elegir al mejor individuo que se haya producido en cualquiera de las generaciones.

Una aplicación similar a la presentada en esta sección, junto con resultados de simulaciones se puede ver en [67].

2.3.4. Aplicaciones

Programación genética ha sido aplicada con éxito a una amplia variedad de problemas, superando muchas veces a métodos tradicionales. El autor principal de PG, John Koza, se ha encargado de documentar aplicaciones donde esto ha sucedido [16–18]. Entre los campos donde se ha aplicado con éxito PG se pueden mencionar los autómatas celulares, biología molecular, diseño automático de circuitos electrónicos, algoritmos de búsqueda en bases de datos, algoritmos para jugar fútbol y otros juegos simulados computacionalmente, estrategias de control automático y diseño de antenas de transmisión [19, 20]. En todos los casos mencionados, PG ha replicado o mejorado soluciones que fueron creadas manualmente o usando herramientas clásicas y que eran consideradas la mejor solución disponible en el momento, habiendo sido incluso algunas de ellas patentadas.

2.4. Diseño de clasificadores con programación genética

Desde sus comienzos, programación genética ha sido aplicada al desarrollo de clasificadores supervisados. Ya en su primer libro, Koza muestra su utilización para el diseño de clasificadores binarios [15]. Esta aplicación es bastante simple, ya que separa el espacio de salida de los árboles en dos, asignando los valores positivos a una clase y los negativos a la otra. Otra opción comúnmente usada es restringir el nodo raíz de los árboles a funciones booleanas, asignando las salidas verdaderas a una clase y las falsas a la otra. En ambos casos, la medida de *fitness* más simple que se puede utilizar es el porcentaje de aciertos (ejemplos correctamente clasificados).

Las dificultades comienzan cuando se intenta usar programación genética para clasificadores en problemas con múltiples clases, ya que no existe una forma directa de mapear la salida escalar de los árboles hacia alguna de las clases. Kishore et al. [21] discuten las preguntas esenciales que se deben resolver para aplicar programación genética a un problema de c clases:

- Ya que una expresión de PG entrega un valor unidimensional, ¿cómo se puede aplicar PG a un problema de c clases?
- ¿Cuál debiera ser la medida de *fitness* durante la evolución?
- ¿Cómo afecta la elección del conjunto de funciones base al desempeño de la clasificación?
- ¿Cómo debieran crearse los conjuntos de entrenamiento para evaluar el *fitness* durante la

evolución?

- ¿Cómo se puede mejorar el conocimiento de las distribuciones presentes en los datos a través de PG?

2.4.1. Extensión a múltiples clases

La primera pregunta es la que merece mayor atención, por los distintos enfoques encontrados en la literatura para resolverla. Aunque no es enteramente verdad que una expresión de PG siempre entrega un valor unidimensional, ya que desde su primer libro, Koza [15] utiliza el operador `LIST` del lenguaje LISP para agrupar valores y formar salidas de varias dimensiones. Sin embargo, esta opción no ha sido considerada por la mayoría de los autores, creando cada uno su propia alternativa.

El enfoque utilizado por Kishore et al. [21] consiste en modelar el problema de c clases como c problemas de 2 clases. En una ejecución del algoritmo, se produce un árbol con salida booleana que aprende a reconocer elementos pertenecientes a una clase en particular y a rechazar el resto. Es decir, se espera que el árbol genere una salida positiva (+1) para las muestras que pertenecen a la clase que representa y negativa (−1) para las muestras restantes, que forman conjuntamente una clase no deseada. De esta manera, se necesitan c ejecuciones para producir una solución completa al problema. El resultado de este método es el conjunto de árboles ganadores de cada una de la ejecuciones.

Aplicando este enfoque, necesariamente surgirán conflictos cuando dos o más árboles se adjudiquen el mismo elemento. Este problema se intenta solucionar mediante una medida de “nivel de asociación” para cada uno de los árboles. En caso de conflicto se asigna el ejemplo al árbol con mayor nivel de asociación. Luego se aplican reglas heurísticas adicionales para reducir las clasificaciones incorrectas.

Mapeo del espacio de salida

Otro enfoque más simple adoptado por varios autores es dividir el espacio de salida unidimensional en diferentes segmentos, y asignar cada segmento a una de las clases [22–27]. El caso más sencillo, denominado “Static Range Selection” (SRS) [22–24], consiste en definir segmentos fijos de forma previa a la ejecución de PG y que permanecen constantes durante la evolución. Cada segmento se asocia con una de las clases de manera consecutiva. Al evaluar un árbol en el patrón

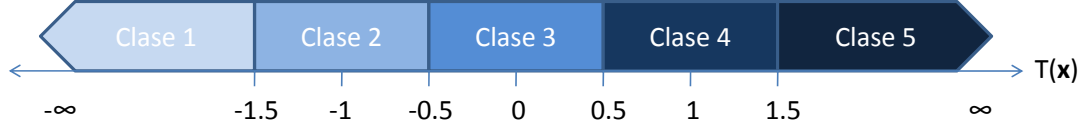


Figura 2.9: Ejemplo de mapeo estático (SRS) con 5 clases

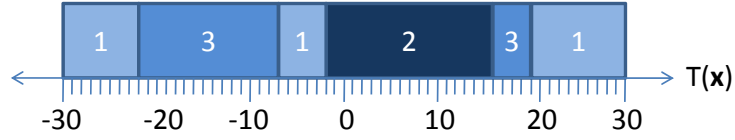


Figura 2.10: Ejemplo de mapeo dinámico (DRS) con 3 clases

\mathbf{x}_i , si la salida del árbol $T(\mathbf{x}_i)$ se encuentra dentro del segmento k , entonces al patrón se le asigna la clase C_k . Un ejemplo del mapeo producido por este método en un problema de 5 clases se muestra en la figura 2.9. Un problema de esta metodología es que la determinación de los segmentos para cada clase recae sobre el usuario.

Otro enfoque es utilizar segmentos determinados de forma dinámica para cada árbol según sus salidas [23]. Este método se denomina *Dynamic Range Selection* (DRS). Pervio a la evolución, el espacio de salida se cuantiza en segmentos unitarios entre un límite inferior y superior preestablecidos. Los límites deben ser suficientemente grandes para abarcar las posibles salidas de los árboles. Para cada árbol, el primer paso consiste en analizar las salidas para un subconjunto de los datos. A cada uno de los segmentos unitarios se le asigna la clase correspondiente a la mayoría de los patrones que provoquen en el árbol una salida ubicada en ese segmento. Todos aquellos segmentos que queden sin una clase, se les asignará la clase del segmento más cercano. Así se formarán segmentos que abarquen varios bloques unitarios y no necesariamente contiguos asociados a las distintas clases. Un ejemplo del mapeo producido por este método se puede ver en la figura 2.10.

Dos variaciones a los métodos anteriores son *Centred Dynamic Range Selection* (CDRS) y *Slotted Dynamic Range Selection* (SDRS) [25, 26]. Estos enfoques forman los segmentos dinámicamente, pero considerando las salidas de todos los árboles a la vez, dando mayor importancia a aquellos con mejor *fitness*. En el primer caso, los segmentos son contiguos y están centrados en el lugar donde se acumulen más salidas para cada clase, mientras que en el segundo se usa una cuantización similar a DRS.

Los trabajos previos han demostrado la efectividad de estos enfoques, particularmente los métodos dinámicos, en variados problemas de clasificación. En los métodos estáticos, las divisiones del espacio de salida de los programas deben ser predefinidos y permanecen fijos. En los métodos dinámicos, las divisiones entre las clases se definen automáticamente durante la evolución. A pesar de su relativa efectividad, estos enfoques presentan problemas. Mientras que los métodos estáticos necesitan que se definan manualmente las divisiones de forma adecuada, los métodos dinámicos generalmente requieren de mucho tiempo de búsqueda para encontrar buenas divisiones. Ambos enfoques usualmente requieren largos tiempos de entrenamiento, obteniendo árboles excesivamente complejos, a veces incluso sin buenos resultados [27].

Otros enfoques han considerado modelar la salida de cada clase con una distribución gaussiana y tratar de minimizar el traslape entre las distintas distribuciones [27], o favorecer la creación de nichos en la población, donde cada uno esté dedicado a una de las clases [70].

Enfoque multiárbol

Muni et al. [28] proponen que la representación clásica tipo árbol no es lo suficientemente poderosa para generar clasificadores de múltiples clases. Para solucionar esto introducen el enfoque denominado multiárbol, donde se crea una nueva representación para los individuos, en la cual cada individuo está formado por c árboles para un problema de c clases. Cada árbol dentro de un individuo representa una regla para una clase, que se activa cuando un patrón es reconocido como perteneciente a esa clase. Esta propuesta es similar al enfoque de los c clasificadores binarios propuesto en [21], con la gran diferencia de que todos los clasificadores evolucionan a la vez en una sola ejecución de PG, en vez de necesitar c ejecuciones. Esto tiene la ventaja de que los individuos representan soluciones completas al problema de clasificación.

Las estructuras obtenidas con la representación multiárbol propuesta en [28] son similares a las que se pueden obtener con el operador `LIST` presente en el lenguaje `LISP`, aunque ambos enfoques presentan diferencias importantes. En el enfoque multiárbol, los árboles dentro de cada individuo son distinguibles como entidades independientes, por lo que se pueden crear operadores genéticos de más alto nivel (que operen al nivel de los árboles, a diferencia de los operadores clásicos, que funcionan al nivel de los nodos). Por ejemplo, en [28] se introducen operadores que intercambian árboles entre dos individuos de acuerdo a la clase que representan. Estos operadores aceleran la convergencia y tratan de evitar la destrucción de árboles aptos dentro de los individuos. Esto no

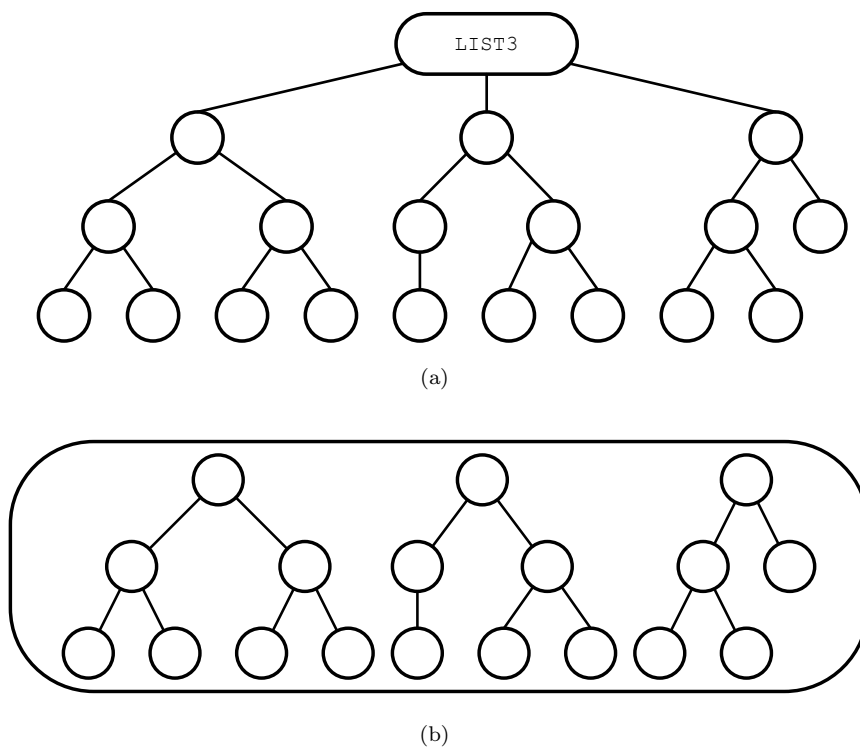


Figura 2.11: (a) Un árbol con salida en 3 dimensiones que utiliza el operador **LIST**; (b) Un individuo multiárbol que contiene 3 árboles

sería posible usando la representación clásica de un sólo árbol, ya que el operador **LIST** es un nodo más que simplemente agrupa las salidas de sus nodos hijo en un vector. Es necesario aclarar que la diferencia entre ambos enfoques es netamente conceptual, ya que es totalmente factible que la implementación interna de un enfoque multiárbol utilice el operador **LIST** para obtener la salida de un individuo. La figura 2.11 muestra un ejemplo de un árbol cuya raíz es un operador **LIST** comparado con un individuo multiárbol. La salida de ambas estructuras en un vector de 3 dimensiones.

Al interpretar los árboles como reglas lógicas, el enfoque propuesto en [28] presenta las mismas deficiencias que [21] en el sentido de que se generan conflictos cuando más de un árbol se atribuye la pertenencia de un mismo patrón. Los autores también introducen sus propias reglas heurísticas para tratar de solucionar los conflictos.

En [30] se ocupa un esquema similar, pero se considera un número variable de reglas para cada clase (lo que se traduce en un número variable de árboles por individuo). La idea de este esquema es que con varios árboles por clase se pueden capturar mejor la estructura de subclases presentes

en los datos.

Por último, en [29], Muni et al. presentan mejoras graduales a su enfoque multiárbol, agregando selección de características relevantes en forma simultánea a la evolución de los clasificadores.

Extracción de características

Otro enfoque utilizado para el diseño de clasificadores con PG es el de extracción de características [31–36]. En este enfoque se utilizan los individuos clásicos de PG, conformados por un solo árbol. La salida del árbol se interpreta como una nueva característica, que al depender de las variables de entrada, consiste en combinaciones de las características propias del problema estudiado. La idea básica consiste en generar la característica con la cual se logre la mejor discriminación posible entre las clases. Para eso, se aplica algún clasificador estándar sobre las salidas de cada árbol. Por ejemplo, en [34] se ocupa un clasificador basado en los k vecinos más cercanos (KNN). Aquellos árboles que generen una característica con la cual el clasificador logre el mejor desempeño tendrán un mejor *fitness*. En este sentido, este enfoque es similar al de mapeo del espacio de salida y se podría considerar una versión más avanzada de éste, ya que se ocupan clasificadores más poderosos, mientras que en el caso del mapeo los clasificadores corresponden a heurísticas aplicadas de acuerdo a los segmentos formados en el espacio de salida.

2.4.2. Medida de *fitness*

En casi todas las publicaciones de clasificación con PG, la medida de *fitness* utilizada es simplemente la razón entre los elementos bien clasificados y aquellos usados para el entrenamiento, es decir:

$$\text{fitness} = \frac{\text{número de ejemplos clasificados correctamente}}{\text{número de ejemplos utilizados para entrenamiento}} \quad (2.24)$$

2.4.3. Operadores genéticos

Dado que la mayoría de los trabajos en el área utilizan la representación clásica de los individuos (un árbol por individuo), no es necesario definir nuevos operadores genéticos, por lo que generalmente se utilizan los operadores clásicos de crossover y mutación. En [28], en cambio, al usar la representación multiárbol, es necesario adaptar estos operadores para que tomen en cuenta

la nueva estructura de los individuos. No es claro, por ejemplo, cómo seleccionar un árbol de cada individuo para efectuar el crossover entre ellos. También es posible introducir nuevos operadores especialmente diseñados para el problema de clasificación a resolver.

Crossover guiado

Muni et al. [28] definieron un operador de crossover que se aplica entre árboles que representan a la misma clase, es decir, que están en la misma posición en cada uno de sus individuos respectivos. Así, para los individuos padre I_{i_1} y I_{i_2} , el crossover se realiza entre los árboles $T_k^{i_1}$ y $T_k^{i_2}$. Este operador está basado en la idea de que la detección de patrones de cada clase corresponden a problemas independientes y, por lo tanto, sólo los árboles que representan a la misma clase debieran mezclarse entre ellos.

Desadaptación o *unfitness*

Otro concepto introducido en [28] es la desadaptación o *unfitness*, que consiste en asignar una medida adicional de *fitness* a cada árbol dentro de un individuo, independiente del *fitness* que tenga el individuo. Esta medida adicional se utiliza para guiar a los operadores genéticos en la selección de árboles dentro de un individuo. El concepto se denomina desadaptación ya que es una medida inversa al *fitness*, es decir, mide que tan *mal* resuelve un árbol el problema. La base de introducir esta medida es que dentro de un individuo que ya ha sido seleccionado como padre (que debido al proceso de selección probablemente tenga un *fitness* alto) pueden existir árboles que perjudican el desempeño del individuo como clasificador y debieran ser dichos árboles los modificados para que puedan ser mejorados. Así se evita alterar los árboles que sí aportan al desempeño. En [28] la selección del árbol a modificar dentro de un individuo se realiza por ruleta, donde los casilleros son proporcionales al *unfitness* de cada árbol.

2.4.4. Complejidad de los programas resultantes

Es sabido que en programación genética existe un fuerte compromiso entre desempeño y simplicidad [71]. La representación utilizada para los individuos y los operadores aplicados sobre ellos generan el denominado problema del *bloat*, que consiste en hacer crecer los árboles más de lo necesario a través de ramas que no cumplen ninguna utilidad. En muchas aplicaciones, es posible que

la complejidad de los árboles resultantes no sea una limitante para el uso de PG. Para el diseño de clasificadores, en cambio, existen razones para buscar modelos sencillos. Al respecto, cabe recordar el principio de la “Navaja de Occam” [72], que básicamente dice que se debe preferir un modelo más simple por sobre uno más complejo si los dos tienen un desempeño similar. Según [71], existen dos posibles interpretaciones de este principio cuando se trata del diseño de clasificadores:

1. Dados dos modelos con el mismo error de generalización, el más simple debiera preferirse porque la simplicidad es deseable por sí sola.
2. Dados dos modelos con el mismo error de entrenamiento, el más simple debiera preferirse porque es probable que tenga un menor error de generalización.

En [71] se mencionan varios argumentos que invalidan la segunda propuesta, asegurando que corresponde incluso a una interpretación errónea de la Navaja de Occam. La primera propuesta, en cambio es claramente válida en el caso de programación genética, ya que un modelo más simple requiere menos memoria para ser almacenado, se ejecuta más rápido y además permitiría analizar manualmente los árboles resultantes para extraer información adicional sobre las reglas que generan los *clusters*.

Se han estudiado muchas técnicas para solucionar o al menos aliviar el problema del *bloat* en PG. Una posibilidad es restringir fuertemente el tamaño máximo que puedan tomar los árboles, ya sea especificando un largo máximo o un número máximo de nodos [73]. También se han estudiado técnicas de optimización multiobjetivo, agregando el tamaño de los árboles como un segundo objetivo a optimizar [74, 75]. Otra metodología es simplificar o “podar” los árboles ya sea durante o al finalizar la evolución [76–78]. También se han estudiado modificaciones a los operadores genéticos que intentan controlar automáticamente el crecimiento de los árboles [79, 80].

Un área donde existe mucho interés es la aplicación de PG para extracción de reglas, especialmente en ámbitos de biomédica. La idea es evolucionar clasificadores de tamaño limitado que sean fácilmente interpretables para poder extraer conocimiento de los datos. La necesidad de que estas reglas sean pocas y comprensibles obliga a preocuparse del problema de la simplicidad. Algunas aplicaciones interesantes en esta área pueden revisarse en [81–86].

2.4.5. Costo computacional

Otro problema conocido de PG es su alto costo computacional. Al igual que en otras técnicas evolutivas, cada individuo representa una posible solución al problema tratado y se deben evaluar todos los individuos en cada generación. En contraste, métodos clásicos como descenso por gradiente optimizan considerando sólo una solución a la vez. Esto, unido al problema del *bloat*, que hace que los árboles tiendan a crecer sin control, eleva considerablemente los costos computacionales de PG.

Este problema ha sido estudiado desde los comienzos de PG. Algunos enfoques recientes para tratar el problema y acelerar los cálculos son adaptaciones para tratar con alta cantidad de datos [87–89], implementaciones paralelas de PG [90] e incluso implementaciones en procesadores de tarjetas gráficas (GPU, por sus siglas en inglés) [91], que presentan un alto grado de paralelismo.

2.5. *Clustering* con programación genética

Dados los buenos resultados obtenidos con PG en problemas de clasificación supervisada, resulta de interés estudiar su aplicación a problemas de *clustering*. En teoría, el uso de PG para *clustering* permitiría particionar conjuntos con cualquier forma, ya que como PG es capaz de aproximar cualquier tipo de función, no sería necesario asumir ninguna distribución *a priori* de los datos.

Por su similitud con un problema de clasificación supervisada, es posible utilizar para *clustering* las mismas representaciones para los individuos revisadas en la sección 2.4.1. En efecto, desde el punto de vista de PG, la única diferencia entre un problema de clasificación supervisada y no supervisada es que en el primer caso, las etiquetas reales pueden ser usadas en el *fitness*, mientras que en *clustering*, el *fitness* se debe construir sólo en base a criterios internos, calculados directamente desde los datos. Inmediatamente surge la idea de utilizar como *fitness* alguna de las funciones de validez del *clustering* vistas en la sección 2.2.2, ya que los individuos con mayor *fitness* serían aquellos que generen un *clustering* más *válido*. Sin embargo, como se ve también en la misma sección, el uso de estas funciones de validez permitiría obtener *clusters* sólo de formas hiperesféricas.

Hasta el momento, el único trabajo encontrado en la literatura que estudia la aplicación de PG a *clustering* es el de De Falco et. al [37]. En dicho trabajo se presenta un esquema basado en individuos multiárbol, donde cada árbol está asociado a uno de los *clusters*, de forma similar a los clasificadores supervisados con PG presentados en la sección 2.4.1. Este trabajo se analiza en

detalle a continuación, por tratarse de la única publicación dedicada a *clustering* con PG.

2.5.1. Representación

Al igual que en [28], cada árbol representa una regla binaria que se activa cuando el patrón pertenece al *cluster* asociado al árbol. Debido a esto es necesaria una fase de resolución de conflictos cuando varios árboles dentro de un individuo se asignan el mismo patrón. La resolución de conflictos está basada en el número de predicados (nodos con funciones lógicas) que contiene la regla. Al igual que en [30], y a diferencia de otros esquemas presentados, en [37] el número de árboles por individuo es variable. Esto, con el fin de que el método sea capaz de identificar automáticamente el número de *clusters* presente en los datos, sin usar información *a priori*. Para representar los individuos, se utiliza una gramática libre de contexto (básicamente, una serie de reglas para generar fórmulas o, en este caso, árboles), incluyendo un símbolo que designa el fin de un árbol y el comienzo de otro. Así, los individuos pueden aumentar o disminuir la cantidad de árboles libremente.

2.5.2. Medida de *fitness*

El *fitness* utilizado en [37] está basado en estadísticas de segundo orden de los datos. Como se ha mencionado anteriormente, esto permite al método encontrar *clusters* sólo de formas hipersféricas. Para asignar el *fitness* a un individuo se forman primero *clusters* estrictos de acuerdo a las salidas de cada árbol, aplicando el siguiente procedimiento:

1. Obtener la salida binarizada (1 ó 0 según sea positiva o negativa) de cada uno de los árboles del individuo.
2. Se dirá que un patrón activa una regla (árbol) si el árbol presenta una salida positiva para dicho patrón.
3. Puede ocurrir cualquiera de las situaciones siguientes:
 - a) el patrón activa sólo una regla (árbol): se asigna el patrón al *cluster* representado por el árbol;
 - b) el patrón activa más de una regla (árbol), cada una con diferentes tamaños (número de predicados): el patrón es asignado al *cluster* cuya regla presente una mayor cantidad de

predicados con respuesta positiva para el patrón²;

- c) el patrón activa más de una regla con el mismo número de predicados: el patrón se clasifica como indefinido y no se le asigna un *cluster* en esta etapa.

4. Asignar los patrones que no han activado ninguna regla y aquellos indefinidos:

- a) para cada regla, calcular la media de todos los patrones asignados hasta el momento al *cluster* que representa; este punto corresponde al centroide del *cluster*;
- b) el patrón se asigna al *cluster* cuyo centroide sea el más cercano

Una vez asignados todos los patrones a algún *cluster*, se calculan los estadísticos necesarios para asignar el *fitness*. Primero se procede a calcular, para cada *cluster* C_k , su centroide, denominado \mathbf{p}_k . En base a los centroides, se define la homogeneidad del *cluster* como

$$\mathcal{H}(C_k) \equiv -\frac{\sum_{i=1}^n D(\mathbf{x}_i, \mathbf{p}_k)}{\omega_k} \quad (2.25)$$

donde ω_k es la cardinalidad de C_k y $D(\mathbf{x}_i, \mathbf{p}_k)$ es la distancia entre el patrón \mathbf{x}_i y el centroide \mathbf{p}_k . En un principio, la medida de distancia podría ser cualquiera, pero en [37] se inclinan por una norma tipo Manhattan (o norma 1) definida como:

$$\delta(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{d} \sum_{l=1}^d \frac{1}{R_l} |x_{il} - x_{jl}|, \quad (2.26)$$

donde R_l es el rango de los datos para cada dimensión, es decir:

$$R_l = \max_i x_{il} - \min_i x_{il} \quad (2.27)$$

Esto asegura que $D(\mathbf{x}_i, \mathbf{x}_j) \in [0, 1] \forall \{\mathbf{x}_i, \mathbf{x}_j\} \in X^2$.

A través de la homogeneidad de cada *cluster*, se calcula la homogeneidad del *clustering* completo:

$$\mathcal{H}(C) \equiv \frac{\sum_{k=1}^c \omega_k \mathcal{H}(C_k)}{\sum_{k=1}^c \omega_k} \quad (2.28)$$

²La idea es asignar el patrón al árbol que delimite una parte más pequeña del espacio de entrada, para obtener *clusters* más especializados.

Por otro lado, se define la separabilidad entre el conjunto de *clusters* como

$$\mathcal{S}(C) \equiv \frac{\sum_{k=1}^{c-1} \sum_{k'=k}^c \omega_k \omega_{k'} D(C_k, C_{k'})}{\sum_{k=1}^{c-1} \sum_{k'=k}^c \omega_k \omega_{k'}} \quad (2.29)$$

donde $D(C_k, C_{k'})$ corresponde a la distancia entre *clusters* y se calcula como la distancia entre sus respectivos centroides, $\delta(\mathbf{p}_i, \mathbf{p}_j)$.

Finalmente, para un individuo I que produce un *clustering* C , el *fitness* asociado es una combinación lineal entre la homogeneidad y la separabilidad del *clustering*, dada por:

$$f(C) = \mathcal{H}(C) + \mu \mathcal{S}(C) \quad (2.30)$$

El parámetro μ controla la importancia relativa de ambos términos.

2.5.3. Comentarios

El trabajo presentado en [37] demuestra la factibilidad de aplicar PG a tareas de *clustering*. A pesar de esto, muestra algunas deficiencias que se mencionan a continuación:

- El *fitness* no depende directamente del número de *clusters* formados por cada individuo. En teoría, esto podría lograr el objetivo de encontrar el número correcto de *clusters* de forma automática. Sin embargo los resultados presentados en [37] muestran que el número óptimo de *clusters* depende fuertemente de la elección del parámetro μ . Por lo tanto, en vez de especificar el parámetro c , el usuario debe especificar otro parámetro de más difícil interpretación, pero que igualmente determinará el número de *clusters* formado.
- El *fitness* está basado en estadísticos de segundo orden de los datos. De hecho, su formulación es bastante similar a muchas funciones de validez presentes en la literatura: un balance entre similitud intra *clusters* y separabilidad inter *clusters*. Es claro, por lo tanto, que el método propuesto sólo logrará encontrar *clusters* de forma hiperesférica.
- No queda claro que el criterio para resolver los conflictos sea el más adecuado. No se efectúan pruebas para confirmar que el criterio ayude a mejorar los resultados.
- La evaluación del método no es la más adecuada, ya que éste es probado en sólo una base de datos, que por lo demás no es usada en otras publicaciones sobre *clustering*. A su vez, los

índices de validez usados para la evaluación tampoco corresponden a los comúnmente encontrados en la literatura sobre *clustering*. Por lo tanto, resulta difícil efectuar comparaciones del método propuesto con otros existentes en la literatura.

En conclusión, el esquema propuesto no aprovecha las potenciales ventajas de aplicar PG para *clustering*. Queda entonces abierta la posibilidad de diseñar un enfoque alternativo que sí aproveche estas ventajas, obteniendo un método de *clustering* que sea aplicable a una mayor cantidad de problemas.

2.6. *Clustering* usando otros métodos evolutivos

Aparte de programación genética, otros métodos evolutivos han sido ampliamente aplicados a problemas de *clustering*. Estas aplicaciones pretenden explotar su alta capacidad de exploración que les permite optimizar funciones complejas evitando mínimos locales. Los métodos más ocupados hasta ahora son algoritmos genéticos (AG) y optimización por enjambre de partículas (PSO). En esta sección se presentan algunos trabajos encontrados en la literatura y se estudian las distintas representaciones y medidas de *fitness* utilizadas. Está fuera del alcance de esta tesis describir en detalle los métodos AG y PSO. Para mayores detalles se sugiere al lector referirse a [65] y [92], respectivamente.

2.6.1. Algoritmos genéticos

Los algoritmos genéticos pueden considerarse como el precursor de la computación evolutiva y en particular, de programación genética [15]. En AG, los individuos consisten en tiras o “strings” binarios de largo fijo. El usuario debe decidir la representación o codificación de las posibles soluciones al problema, a través de la interpretación dada a cada bit y el largo de la tira.

En la literatura se pueden encontrar numerosos ejemplos de AG aplicados a *clustering* [60, 62, 63, 93–96]. Las distintas aplicaciones difieren principalmente en la representación de las soluciones y la medida de *fitness* utilizada. La representación más básica consiste en codificar los centroides de cada *cluster* con cierta resolución (con cierto número de bits) en la tira binaria. Así, cada individuo representa los centros de todos los *clusters*. La pertenencia a cada *cluster* se asigna eligiendo el centroide más cercano, aunque en algunas variantes la pertenencia es difusa. Con esta representación,

el problema se convierte en el mismo que resuelven los algoritmos *k-means* y *fuzzy c-means*, es decir, encontrar el conjunto de centroides que optimicen cierta función objetivo. Por lo tanto, estas mismas funciones objetivo (en su versión discreta o difusa, dependiendo del tipo de pertenencia) u otras similares se utilizan como medida de *fitness* en los enfoques evolutivos mencionados. Algunos métodos ocupan directamente como medida de *fitness* algunas de las funciones de validez interna vistas en la sección 2.2.2, o incluso ponderaciones de varias de ellas [63]. En general, todas estas aplicaciones basadas en AG logran un mejor desempeño que los métodos clásicos, ya que evitan los problemas de estancamiento en mínimos locales que presentan los métodos basados en gradiente. Por otro lado, tanto por la representación utilizada, como por las medidas de *fitness* basadas en la distancia a los centroides, estos métodos sólo logran detectar *clusters* hiperelipsoidales.

2.6.2. Optimización por enjambre de partículas (PSO)

A pesar de que PSO no es exactamente un método evolutivo, sino más bien un método “social”, generalmente en la literatura se le clasifica como una rama de la computación evolucionaria. En PSO, en vez de individuos se habla de partículas que se mueven a través del espacio de búsqueda. El método trata de modelar de manera muy simplificada el comportamiento de grupos de organismos presentes en la naturaleza, como bandadas de pájaros o cardúmenes de peces, en cuyos movimientos se aprecia una fuerte componente social (el grupo de organismos parece moverse como un todo, a pesar de tener cada uno libertad en su movimiento a nivel local). Al igual que en el caso de AG, el enfoque más común para aplicar PSO a *clustering* consiste en codificar directamente en las partículas los centroides de cada *cluster* (es decir, cada partícula es de dimensión $d \cdot c$ y corresponde a la concatenación de todos los centroides) [97, 98]. Dado que en PSO las partículas son vectores de variables continuas (a diferencia de tiras binarias como en AG), es posible lograr una mayor precisión en la localización de los centroides.

2.6.3. Funciones objetivo

En todos los trabajos descritos anteriormente se han utilizado funciones de *fitness* basadas en estadísticos de segundo orden de los datos, como la varianza de los *cluster* y las distancias entre los centroides de cada *cluster*. La mayoría de las publicaciones propone sus propias funciones objetivo, aunque algunas de ellas optimizan uno de los índices de validez interna vistos en la sección 2.2.2,

o combinaciones de ellos. Como ya se ha mencionado anteriormente, esto permite a estos métodos encontrar sólo *clusters* hiperelipsoidales.

2.7. Teoría de la información

La teoría de la información, desarrollada por Shannon en 1948 [11], estudia el contenido de información presente en un mensaje. A través de su teoría, Shannon resolvió dos problemas fundamentales asociados a la comunicación: cómo lograr la máxima compresión de datos y cómo lograr una comunicación perfecta a través de canales imperfectos.

“El problema fundamental de la comunicación consiste en reproducir en un punto, ya sea exacta o aproximadamente, un mensaje producido en otro punto” [11].

Dada una variable aleatoria discreta x , ésta tendrá asociada un alfabeto $\mathcal{A} = \{a_1, \dots, a_I\}$, que corresponde al conjunto de posibles valores que puede tomar x , y una función de probabilidad $P : \mathcal{A} \rightarrow [0, 1]$, de manera que $P(a_i) := P(x = a_i)$. Por simplicidad, en casos que no haya posibilidad de confusión se denotarán los eventos simplemente como x , y la probabilidad de un evento cualquiera como $P(x)$.

El contenido de información de Shannon de un evento de la variable aleatoria x está dado por:

$$h(x) = \log \frac{1}{P(x)} \quad (2.31)$$

y mide la cantidad de información ganada al conocer el valor de x . Al usar el logaritmo en base 2, la información está medida en *bits*. Dada esta definición, los eventos más probables tienen menor contenido de información, lo cual tiene mucho sentido. Supóngase, por ejemplo, que se juega un partido de fútbol entre Brasil y las Islas Feroe. Los posibles eventos dan lugar al alfabeto $\mathcal{A} = \{B, F\}$, donde B significa que el ganador es Brasil y F , las Islas Feroe (para simplificar, no hay posibilidad de empate). Dada la historia de ambas selecciones, se podrían asumir *a priori* valores para las probabilidades de cada resultado posible, por ejemplo, $P(x = B) = 0,9$ y $P(x = F) = 1 - P(x = B) = 0,1$. La definición de contenido de información propuesto por Shannon pretende rescatar el hecho de que se gana más información al conocer un resultado que otro. En este ejemplo, es poco valioso (se gana poca información) conocer el resultado si el ganador fue Brasil, ya que este

era el resultado más probable. De hecho, es claro que causaría mucho mayor interés la noticia sobre un triunfo del equipo feroés: el evento contiene mayor información.

Esta definición cumple una propiedad que intuitivamente debiera tener cualquier medida de información, y es que la información contenida en dos eventos independientes debiera ser la suma de la información de cada evento por separado:

$$h(a_1, a_2) = h(a_1) + h(a_2) \quad (2.32)$$

La entropía de una variable aleatoria se define como la esperanza del contenido de información para todos los posibles eventos:

$$H(x) := E\{h(x)\} = \sum_{x \in \mathcal{A}} P(x) h(x) = \sum_{x \in \mathcal{A}} P(x) \log \frac{1}{P(x)} \quad (2.33)$$

tomando como convención para $P(x) = 0$ que $0 \times \log 0 \equiv 0$, dado que $\lim_{\theta \rightarrow 0^+} \theta \log \frac{1}{\theta} = 0$. Es claro de la definición que $H(x) \geq 0$.

La entropía mide entonces el contenido medio de información de una variable aleatoria. Así, si la entropía es baja, significa que en promedio los resultados de x entregan poca información, lo cual sucede si es que existen algunos eventos muy probables comparados con los demás. Por el contrario, si todos los eventos posibles son igualmente probables, cualquiera de ellos entrega la misma información y la entropía será comparativamente alta.

Nótese que la entropía es una función que actúa sobre la función de probabilidad P . La entropía no depende de los valores que pueda tomar x , sino sólo de sus probabilidades. Retomando el ejemplo del partido de fútbol (o cualquier otro conjunto con dos eventos), si se define $P(x = B) := p$ (con lo que $P(x = F) = 1 - p$), se puede graficar la entropía de x en función del valor de p . La figura 2.12 muestra la dependencia de $H(x)$ al variar p . La entropía alcanza un máximo de 1 bit cuando $p = \frac{1}{2}$, es decir, cuando la distribución es uniforme. Por el contrario, si alguno de los eventos tiene probabilidad 1, la entropía es igual a cero. Estas propiedades se cumplen para alfabetos de cualquier tamaño y muestran una característica esencial de la entropía. Al tener una distribución uniforme, todos los eventos son igualmente probables, con lo que la incerteza sobre el resultado es máxima. En cambio, si algún evento es muy probable comparado con los demás, la incerteza sobre el resultado disminuye. De esta forma la entropía es una medida de la incerteza que existe sobre los resultados

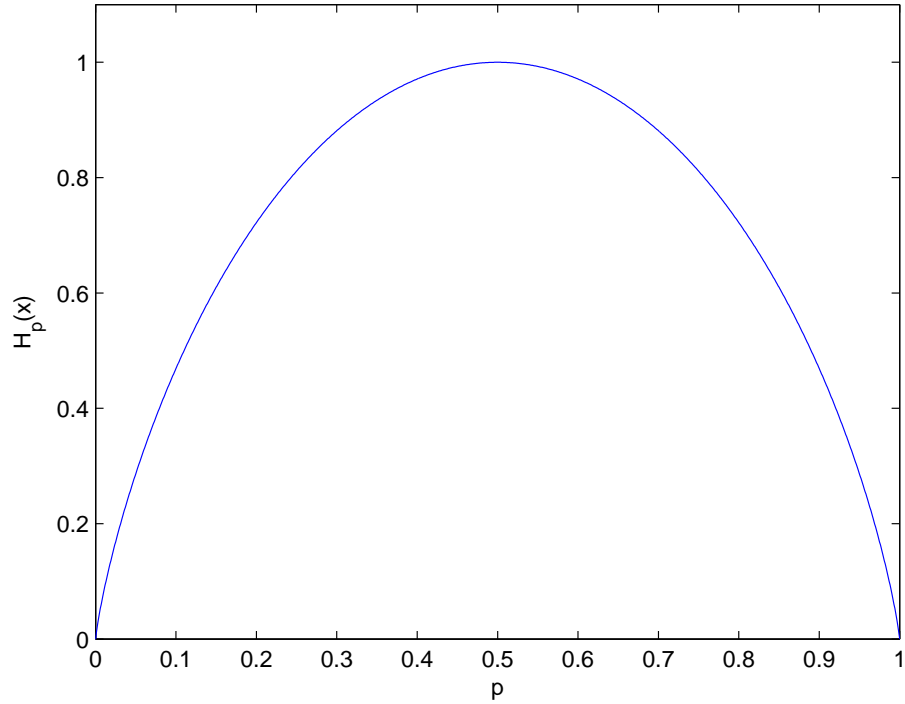


Figura 2.12: Entropía de una variable aleatoria binaria, dependiendo del valor de p

de un evento aleatorio.

Un resultado importante demostrado por Shannon es que la entropía, al medir la cantidad de información promedio de una variable aleatoria, corresponde al grado máximo en que se puede comprimir dicha variable. Es decir, para representar un mensaje, la cantidad mínima de bits que se necesitan corresponde exactamente a la entropía de la variable aleatoria que genera el mensaje. Esta propiedad se aplica a cualquier tipo de mensajes, por ejemplo, un texto, una imagen. En la terminología de reconocimiento de patrones, el mensaje transmitido corresponde a un patrón.

La entropía se puede extender fácilmente al dominio continuo. Para una variable aleatoria continua $x \in \mathfrak{R}$, se define su entropía como

$$H(x) = \int_{\mathfrak{R}} p(x) \frac{1}{\log p(x)} dx \quad (2.34)$$

donde $p(x)$ es la función de densidad de probabilidad de x , como se define en la sección 2.1. Dado que en reconocimiento de patrones se trabaja principalmente con variables continuas que provienen de fenómenos reales, todos los conceptos descritos en adelante se refieren a variables continuas.

Además de la entropía de una variable, es interesante considerar cómo se relacionan dos variables aleatorias entre sí. Para esto se define la entropía conjunta entre x e y como

$$H(x, y) := E \left\{ \frac{1}{\log p(x, y)} \right\} = \int \int p(x, y) \frac{1}{\log p(x, y)} dx dy \quad (2.35)$$

donde $p(x, y)$ es la densidad de probabilidad conjunta de x e y . Esta definición es idéntica a la de la entropía de una variable, considerando a (x, y) como la variable aleatoria a estudiar.

También se puede definir la entropía condicional entre dos variables aleatorias de la siguiente forma:

$$H(x|y) = \int \int p(x, y) \frac{1}{\log p(x|y)} dx dy \quad (2.36)$$

La entropía conjunta y condicional están relacionadas de una manera muy natural, denominada la regla de la cadena para entropías:

$$H(x, y) = H(x) + H(x|y) = H(y) + H(y|x) \quad (2.37)$$

Esta igualdad puede interpretarse como que la incertidumbre presente en ambas variables proviene de dos fuentes: de la incertidumbre presente en la primera variable por sí sola y de la incertidumbre que se mantiene en la primera variable al conocer el valor de segunda. La regla de la cadena refleja además muy claramente la noción de independencia: si (x, y) son independientes, se tiene que $H(x|y) = H(y)$, y por lo tanto, $H(x, y) = H(x) + H(y)$. Es decir, la incertidumbre presente en ambas variables corresponde a la suma de las incertidumbres presentes en cada una de ellas.

Otra medida importante es la información mutua entre dos variables:

$$I(x; y) = \int \int p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy \quad (2.38)$$

que mide la cantidad de información presente en ambas variables a la vez.

Es fácil demostrar que:

$$I(x; y) = H(x) - H(x|y) = H(y) - H(y|x) = H(x) + H(y) - H(x, y) \quad (2.39)$$

es decir, la información mutua mide la reducción en la incerteza sobre una variable al conocer la otra. Además, se deduce que si dos variables aleatorias son independientes entre sí, la información mutua entre ellas será cero. El diagrama de Venn de la figura 2.13 resume de manera didáctica las interrelaciones presentes entre las medidas descritas hasta ahora. Cada variable tiene su cantidad de información, $H(x)$ y $H(y)$. La unión de ambas informaciones corresponde a la entropía conjunta; la intersección, a la información mutua.

Finalmente, la teoría de la información también provee una definición general para una entropía relativa entre dos funciones de densidad de probabilidad. Esta es la denominada divergencia de Kullback-Leibler:

$$D_{KL}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx \quad (2.40)$$

que mide la distancia en el espacio de las probabilidades entre dos funciones de densidad. Se puede considerar como el error cometido al asumir que la distribución es q , cuando en realidad es p . Por ejemplo, si se conociera la distribución p , como ya se vio, se podría construir una codificación que usara en promedio $H(p)$ bits para describir los eventos de x . En cambio si se usa el código para una distribución q , se necesitarían en promedio $H(p) + D_{KL}(p||q)$ bits para describir la variable aleatoria.

Se puede ver que la información mutua corresponde a un caso particular de la divergencia de Kullback-Leibler:

$$I(x; y) = D_{KL}(p(x, y) || p(x)p(y)) \quad (2.41)$$

2.7.1. Teoría de la información y *clustering*

La mayoría de los algoritmos de *clustering* que optimizan una función de costos usan estadísticos de segundo orden, como la varianza de los patrones que pertenecen a un *cluster*. Esto lleva a formar *clusters* hiperesféricos o hiperelipsoidales. La entropía, en cambio, considera estadísticos de alto orden, que logran capturar mucho mejor la verdadera estructura de los datos [7]. Es por esto que resulta de interés considerar la entropía como una medida a optimizar en problemas de *clustering*. Si se minimiza la entropía de los patrones que pertenecen a un mismo *cluster*, significa que hay poca

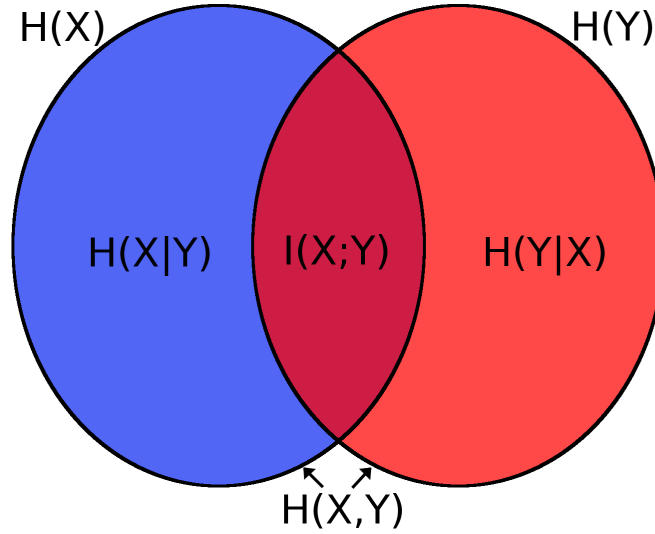


Figura 2.13: Diagrama de Venn que muestra la relación entre entropía, entropía condicional, entropía conjunta e información mutua

variación entre dichos patrones, o equivalentemente, que éstos son muy similares entre sí. Nada se dice sobre la forma que debe tener el *cluster*. La medida que calcula la entropía considerando sólo aquellos patrones que pertenecen al *cluster* C_k es la entropía condicional de \mathbf{x} dado C_k :

$$H(\mathbf{x}|C_k) = \int p(\mathbf{x}|C_k) \log p(\mathbf{x}|C_k) d\mathbf{x} \quad (2.42)$$

recordando el supuesto de reconocimiento de patrones, donde cada clase de los datos se puede modelar por una función de densidad de probabilidad condicional, $p(\mathbf{x}|C_k)$.

Para encontrar un *clustering* óptimo se debe minimizar la entropía condicional para cada uno de los *clusters*. Para esto, se debe calcular la entropía condicional de x dado el conjunto de *clusters* C :

$$H(\mathbf{x}|C) = - \sum_{k=1}^c P(C_k) \int p(\mathbf{x}|C_k) \log p(\mathbf{x}|C_k) d\mathbf{x} \quad (2.43)$$

Por otro lado, la entropía, al medir el grado de aleatoriedad de los datos de un *cluster*, puede considerarse como el opuesto al nivel de complejidad de la estructura presente en el *cluster*. Es decir, si los patrones dentro de un *cluster* estuvieran distribuidos aleatoriamente, la entropía sería muy alta y el grado de estructura muy bajo. Al contrario, para una estructura compleja y organizada,

la entropía es baja. Otra forma de ver esto es recordar que la entropía también mide la cantidad mínima de bits que se necesitan para representar los resultados de una variable aleatoria. Volviendo al supuesto de que cada clase de los datos está generada por una función de densidad de probabilidad condicional distinta, la entropía de cada *cluster* mide la cantidad de bits necesaria para representar a los patrones del *cluster*. Mientras más bits sean necesarios (mayor entropía), menos predecibles son los resultados de la variables aleatoria, es decir, menos similares entre sí son los patrones de un mismo *cluster*.

Otra forma de asegurar que el *clustering* sea adecuado es buscar que los patrones que pertenezcan a distintos *clusters* sean muy diferentes entre sí. Aplicando conceptos de teoría de la información, se puede maximizar una medida de entropía cruzada entre las distribuciones de los datos de cada *cluster*. Una medida existente para tal efecto es la divergencia de Kullback-Leibler definida en (2.40). La divergencia de Kullback-Leibler indica cuan diferentes son las dos distribuciones, cumpliéndose que $D_{KL}(p, q) = 0 \Leftrightarrow p(\mathbf{x}) = q(\mathbf{x}) \forall \mathbf{x}$. En el caso de *clustering* interesaría maximizar $D_{KL}(p(\mathbf{x}|C_k) || p(\mathbf{x}|C_{k'}))$ para cada par de *clusters* k, k' .

Así se obtienen dos posibles criterios basados en teoría de la información, que podrían ser optimizados en aplicaciones de *clustering*. Específicamente, para el objetivo de esta tesis, podrían usarse como medida de *fitness* para programación genética. Lamentablemente, tanto (2.43) como (2.40) son imposibles de calcular de forma exacta y son difíciles de calcular o estimar directamente desde los datos. Se debe recordar que la variable \mathbf{x} es continua, ya que en reconocimiento de patrones se trabaja con mediciones de fenómenos reales y cada dimensión de \mathbf{x} representa una de estas medidas. Entonces, cualquier indicador que se desee utilizar de los descritos anteriormente, se debe usar en su versión continua, con integrales en vez de sumatorias. La existencia de un conjunto de muestras de \mathbf{x} no implica que se pueda evitar el cálculo de las integrales y reemplazarlo por sumas sobre las muestras, como se suele asumir erróneamente. Por otro lado, como la función de densidad de probabilidad no es conocida (si lo fuera, el problema estaría resuelto), en ningún caso pueden las integrales ser calculadas de forma exacta. Se debe buscar a la vez una forma de estimar la pdf utilizando las muestras disponibles y otras medidas de entropía que sean más fácilmente calculables en la práctica. Los enfoques clásicos de estimación de probabilidad a través de la segmentación del espacio e histogramas son irrealizables en dimensiones mayores a 3, y generalmente se tendrá que los datos son de dimensión $d > 3$.

2.7.2. Potencial de información

Príncipe et. al [14] propusieron un método para estimar la entropía directamente de los datos. La base de este método consiste en utilizar la entropía de Renyi en vez de la entropía de Shannon y estimar la densidad de probabilidad de los datos mediante el método de ventana de Parzen. El desarrollo matemático que sigue lleva a eliminar la integral del cálculo de la entropía, haciendo factible su estimación en la práctica.

La entropía de Renyi [99] para una variable aleatoria x está dada por:

$$H_\alpha(\mathbf{x}) = \frac{1}{1-\alpha} \ln \int p^\alpha(\mathbf{x}) d\mathbf{x} \quad (2.44)$$

Esta definición es una generalización de la entropía de Shannon, introduciendo el parámetro α . De hecho, se cumple que

$$\lim_{\alpha \rightarrow 1} H_\alpha(\mathbf{x}) = H(\mathbf{x}) \quad (2.45)$$

con $H(\mathbf{x})$ la entropía de Shannon de la variable \mathbf{x} .

Esta definición alternativa de entropía tiene la ventaja de que el logaritmo se encuentra fuera de la integral, lo que facilita su cálculo en la práctica. Además, está demostrado que la entropía de Renyi es equivalente a la de Shannon en términos de optimización, es decir, comparten los mismos óptimos globales [100]. Si en (2.44) se usa $\alpha = 2$ se obtiene la entropía cuadrática de Renyi:

$$H_2(\mathbf{x}) = -\ln \int p^2(\mathbf{x}) d\mathbf{x} \quad (2.46)$$

llamada así por usar el cuadrado de la distribución de probabilidad. En [14] se combina la entropía de Renyi con un estimador por ventana de Parzen [101]. Dado un conjunto de datos $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathfrak{R}^d$ y donde cada patrón \mathbf{x}_i corresponde a una muestra de la variable aleatoria \mathbf{x} , la función de densidad de probabilidad de \mathbf{x} se puede estimar como

$$\tilde{p}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n G(\mathbf{x} - \mathbf{x}_i, \Sigma) \quad (2.47)$$

donde

$$G(\mathbf{x}, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} \Sigma^{\frac{1}{2}}} \exp \left\{ -\frac{1}{2} \mathbf{x}^T \Sigma^{-1} \mathbf{x} \right\} \quad (2.48)$$

La función G es un kernel gaussiano simétrico en un espacio de dimensión d y con matriz de covarianza Σ . Para un kernel gaussiano cualquiera se puede mostrar que la siguiente relación se cumple:

$$\int G(\mathbf{x} - \mathbf{x}_i, \Sigma_1) G(\mathbf{x} - \mathbf{x}_j, \Sigma_2) d\mathbf{x} = G(\mathbf{x}_i - \mathbf{x}_j, \Sigma_1 + \Sigma_2) \quad (2.49)$$

Si se asume una sola varianza para todas las dimensiones de manera que $\Sigma = \sigma^2 I$, se obtiene una expresión simplificada:

$$G(\mathbf{x} - \mathbf{x}_i, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp \left\{ -\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2\sigma^2} \right\} \quad (2.50)$$

El parámetro σ^2 se denomina tamaño del kernel y debe ser especificado a priori por el usuario. En la sección 2.7.4 se detalla la influencia de este parámetro en la estimación de $p(\mathbf{x})$ y algunas sugerencias para su elección de forma automática.

En el método de Parzen se puede usar cualquier función que cumpla con la definición de kernel, pero la elección del kernel gaussiano facilita en extremo los cálculos. Siguiendo con el desarrollo, el siguiente paso es sustituir $p(\mathbf{x})$ por su estimador en (2.46), con lo que se obtiene:

$$H_2(\mathbf{x}) \approx -\ln \int \frac{1}{n} \sum_{i=1}^n G(\mathbf{x} - \mathbf{x}_i, \sigma^2) \frac{1}{n} \sum_{j=1}^n G(\mathbf{x} - \mathbf{x}_j, \sigma^2) d\mathbf{x} \quad (2.51)$$

luego, intercambiando las sumas con la integral, sigue:

$$H_2(\mathbf{x}) \approx -\ln \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \int G(\mathbf{x} - \mathbf{x}_i, \sigma^2) G(\mathbf{x} - \mathbf{x}_j, \sigma^2) d\mathbf{x} \quad (2.52)$$

y finalmente, usando (2.49) se obtiene

$$H_2(\mathbf{x}) \approx -\ln \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G(\mathbf{x}_i - \mathbf{x}_j, 2\sigma^2) \quad (2.53)$$

Se debe notar que en (2.53) el kernel es evaluado a través de todos los pares de patrones, eliminando la integral del cálculo de la entropía. Además, para obtener este resultado no se realiza ninguna aproximación, más allá de la estimación de $p(\mathbf{x})$. La doble sumatoria implica que el cómputo de (2.53) será $O(n^2)$. A pesar de esto, este desarrollo permite la estimación de la entropía directamente desde un conjunto de datos de manera relativamente eficiente comparado con otros

métodos presentados en la literatura. En adelante se usa la notación simplificada

$$G_{ij,2\sigma^2} \equiv G(\mathbf{x}_i - \mathbf{x}_j, 2\sigma^2) \quad (2.54)$$

para indicar que el kernel se evalúa directamente sobre los patrones disponibles. Es posible usar otros valores de α para calcular la entropía de Renyi. Con $\alpha > 2$ se obtendría aún mayor información sobre las relaciones entre los patrones, pero la complejidad del algoritmo aumentaría rápidamente, ya que aplicando el mismo desarrollo se obtendrían α sumatorias en vez de dos, pasando la estimación a ser $O(n^\alpha)$.

A la expresión

$$V(\mathbf{x}) = \sum_{i=1}^n \sum_{j=1}^n G_{ij,2\sigma^2} \quad (2.55)$$

se le llama el potencial de información en [14], haciendo una analogía con el potencial gravitatorio entre partículas.

2.7.3. Divergencia de Cauchy-Schwartz

Príncipe et. al [14] también propusieron medidas alternativas para estimar la divergencia entre dos distribuciones de probabilidad. Una de estas medidas es la divergencia de Cauchy-Schwartz, dada por:

$$D_{CS}(p, q) = -\ln \frac{\int p(\mathbf{x}) q(\mathbf{x}) d\mathbf{x}}{\sqrt{\int p^2(\mathbf{x}) d\mathbf{x} \int q^2(\mathbf{x}) d\mathbf{x}}} \quad (2.56)$$

Se puede ver que (2.56) es estrictamente positiva ($D_{CS}(p, q) > 0 \Leftrightarrow \forall p \neq q$), que obedece la propiedad de identidad ($D_{CS}(p, q) = 0 \Leftrightarrow p = q$) y que es simétrica ($D_{CS}(p, q) = D_{CS}(q, p) \forall p, q$). Sin embargo, no obedece la desigualdad triangular, por lo que no satisface la definición matemática estricta de distancia.

En el caso de *clustering*, es de interés maximizar la divergencia entre las distribuciones condicionales de cada *cluster*. Así, las distribuciones a comparar pasarían a ser $p_k = p(\mathbf{x}|C_k)$, $p_{k'} = p(\mathbf{x}|C_{k'})$, donde C_k y $C_{k'}$ son dos *clusters* distintos. Jenssen et. al [9] utilizaron esta medida de divergencia como función objetivo para problemas de *clustering* de la siguiente manera. Al igual que para el cálculo del potencial de información descrito anteriormente, para estimar las distribuciones se ocupa el estimador por ventana de Parzen, pero considerando sólo los patrones que pertenecen

a cada *cluster*. Es decir, se estiman $p(\mathbf{x}|C_k)$, $p(\mathbf{x}|C_{k'})$ usando

$$p(\mathbf{x}|C_k) \approx \frac{1}{n} \sum_{\mathbf{x}_i \in C_k} G(\mathbf{x} - \mathbf{x}_i, \sigma^2 I), \quad (2.57)$$

$$p(\mathbf{x}|C_{k'}) \approx \frac{1}{n} \sum_{\mathbf{x}_j \in C_{k'}} G(\mathbf{x} - \mathbf{x}_j, \sigma^2 I) \quad (2.58)$$

Reemplazando los estimadores en (2.56) y aplicando la misma técnica descrita para el potencial de información, la fórmula para la divergencia de Cauchy-Schwartz entre dos *clusters* queda:

$$D_{CS}(p_k, p_{k'}) \approx -\ln \frac{\sum_{\mathbf{x}_i \in C_k} \sum_{\mathbf{x}_j \in C_{k'}} G_{ij, 2\sigma^2}}{\sqrt{\sum_{\mathbf{x}_i, \mathbf{x}_{i'} \in C_k} G_{ii', 2\sigma^2} \sum_{\mathbf{x}_j, \mathbf{x}_{j'} \in C_{k'}} G_{jj', 2\sigma^2}}} \quad (2.59)$$

Se debe notar que el numerador de (2.59) evalúa el kernel gaussiano en pares de patrones que pertenecen a *clusters* distintos. Por el contrario, cada sumatoria en el denominador considera sólo patrones que pertenecen al mismo *cluster*. Tomando esto en consideración, esta fórmula puede ser fácilmente extendida para considerar múltiples *clusters*. Para esto en [9] se introducen las funciones de pertenencia $m(\mathbf{x}) : \mathbb{R}^d \rightarrow \{0, 1\}^c$, donde a cada patrón se le asigna un vector $m(\mathbf{x}) = \{m_k\}^T$, $k \in \{1, \dots, c\}$. Cada componente de m indica si el patrón pertenece a cierto *cluster*:

$$m_k(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \in C_k \\ 0 & \mathbf{x} \notin C_k \end{cases} \quad (2.60)$$

Utilizando las funciones de pertenencia m , la divergencia considerando todos los *clusters* queda:

$$D_{CS}(p_1, \dots, p_c) = -\ln \frac{\frac{1}{2} \sum_{i,j=1}^n (1 - m^T(\mathbf{x}_i) m(\mathbf{x}_j)) G_{ij, 2\sigma^2}}{\sqrt{\prod_{k=1}^c \sum_{i,j=1}^n m_k(\mathbf{x}_i) m_k(\mathbf{x}_j) G_{ij, 2\sigma^2}}} \quad (2.61)$$

Considerando que el logaritmo es una función monótona creciente, éste puede ser omitido para optimizar (2.61). Se define la función de costos de Cauchy-Schwartz como:

$$J_{CS}(p_1, \dots, p_c) = \frac{\frac{1}{2} \sum_{i,j=1}^n (1 - m^T(\mathbf{x}_i) m(\mathbf{x}_j)) G_{ij, 2\sigma^2}}{\sqrt{\prod_{k=1}^c \sum_{i,j=1}^n m_k(\mathbf{x}_i) m_k(\mathbf{x}_j) G_{ij, 2\sigma^2}}} \quad (2.62)$$

con lo que maximizar (2.61) es equivalente a minimizar (2.62).

Analizando (2.62) se puede notar que el denominador, al considerar sólo pares de patrones en el mismo *cluster*, equivale a la multiplicación de los potenciales de información de cada uno de los *cluster*. Recordando que

$$H_2(\mathbf{x}) = -\ln \frac{1}{n^2} V(\mathbf{x}), \quad (2.63)$$

minimizar J_{CS} implica minimizar la suma de las entropías de cada uno de los *clusters*, que es exactamente lo que se buscaba al proponer (2.43) como posible medida a optimizar en un problema de *clustering*. Por otro lado, el numerador de (2.62) consiste en una especie de potencial de información cruzado entre los *cluster*, por lo que minimizar J_{CS} implicaría maximizar las entropías cruzadas entre los pares de *clusters*, tal como se propone en (2.40).

En definitiva, la función de costos propuesta por Jenssen et. al [9] parece una excelente medida para optimizar en problemas de *clustering*, que es calculable en la práctica directamente desde los datos. Expresado de esta manera, el problema de *clustering* se transforma en un problema de optimización, consistente en determinar los valores de las funciones m para cada uno de los patrones de manera de minimizar (2.62).

En [9] se propone modelar el problema como un problema de optimización con restricciones. Las funciones de pertenencia son extendidas al rango continuo $[0, 1]$ y luego se ocupa el método de los multiplicadores de Lagrange y descenso por gradiente para obtener los valores de m . Durante el desarrollo de esta tesis, el mismo método fue publicado en [10]. Dicho trabajo consiste en una versión extendida del anterior y no se detectan cambios evidentes en el método. La función de costos de Cauchy-Schwartz es similar a la función CEF (por *Cluster Evaluation Function*) utilizada por Gockay y Príncipe [7, 13], quienes demostraron su utilidad para problemas de *clustering*. Sin embargo, en ambas publicaciones se utiliza un método de optimización especialmente diseñado para la función de costos propuesta, que es altamente sensible a la inicialización y se estanca fácilmente en mínimos locales. La función CEF es idéntica a (2.62), sólo que no considera el denominador. El aporte de Jenssen por sobre el desarrollo de Gockay y Príncipe consiste principalmente en la forma de adaptar la función de costos para que sea derivable, a través de la transformación de las funciones de pertenencia en variables continuas, además del desarrollo matemático para calcular el gradiente. Sin embargo, el método de Jenssen también sufre de estancamiento en mínimos locales, como es de esperarse para un método basado en descenso por gradiente. Este problema se trata de resolver con un proceso denominado de *annealing*, que consiste en variar el tamaño del kernel. Este

proceso se explica en mayor detalle en la sección siguiente.

2.7.4. Estimación del tamaño del kernel

Se debe notar que para el desarrollo de (2.59) se utilizó el mismo valor de σ^2 para cada estimador de Parzen. Esto significa que la estimación nunca representará fielmente cada una de las distribuciones, independiente del valor de σ^2 utilizado.

Jenssen et. al [9] sugieren usar una estimación del tamaño óptimo del kernel considerando todos los datos. Esta estimación viene dada por la regla de Silverman [102]:

$$\sigma_{\text{opt}} = \sigma_X \left(\frac{4}{n(2d+1)} \right)^{\frac{1}{d+4}} \quad (2.64)$$

donde $\sigma_X^2 = \frac{1}{d} \sum_{i=1}^d \sigma_{ii}^2$, y σ_{ii}^2 son las varianzas para dimensión de los datos. Al utilizar esta estimación para el tamaño del kernel, se está asumiendo que los datos provienen de una sola distribución normal, cuya varianza es σ_{opt}^2 . Esto está en evidente contradicción con el supuesto de que cada clase posee su propia distribución. A pesar de esto, la regla de Silverman cumple su cometido, en el sentido de que provee un valor aproximado para el tamaño del kernel que es útil para distintos conjuntos de datos. La figura 2.14 muestra el resultado de aplicar el método de Parzen para estimar la función de densidad de probabilidad, utilizando distintos valores para el tamaño del kernel. El conjunto de datos para el cual se estima la densidad de probabilidad se muestra en la figura 2.14(a). Los datos fueron generados usando dos distribuciones gaussianas con la misma matriz de covarianza $\Sigma = \sigma^2 I$. Se aprecia cómo cambia la estimación de la distribución, desde una estimación más local a una más global, a medida que se aumenta el valor de σ . Se puede decir que utilizando $\sigma = \sigma_{\text{opt}}$ se obtiene una estimación intermedia, que no es ni muy fina ni muy gruesa.

El método de *clustering* propuesto por Jenssen et al. [9, 10] presenta una importante sensibilidad al valor escogido para σ . Para solucionar esto, se propone un esquema denominado de *annealing*, que consiste en ajustar dinámicamente el valor de σ a través de las iteraciones. Normalmente, el tamaño del kernel varía linealmente desde $2\sigma_{\text{opt}}$ hasta $\frac{1}{2}\sigma_{\text{opt}}$, con lo que se busca que el método comience buscando una solución gruesa al problema, mientras pasa paulatinamente a un ajuste más fino. En esta tesis, se ocupa la regla de Silverman para obtener un valor de σ_{opt} , el cual se mantiene fijo.

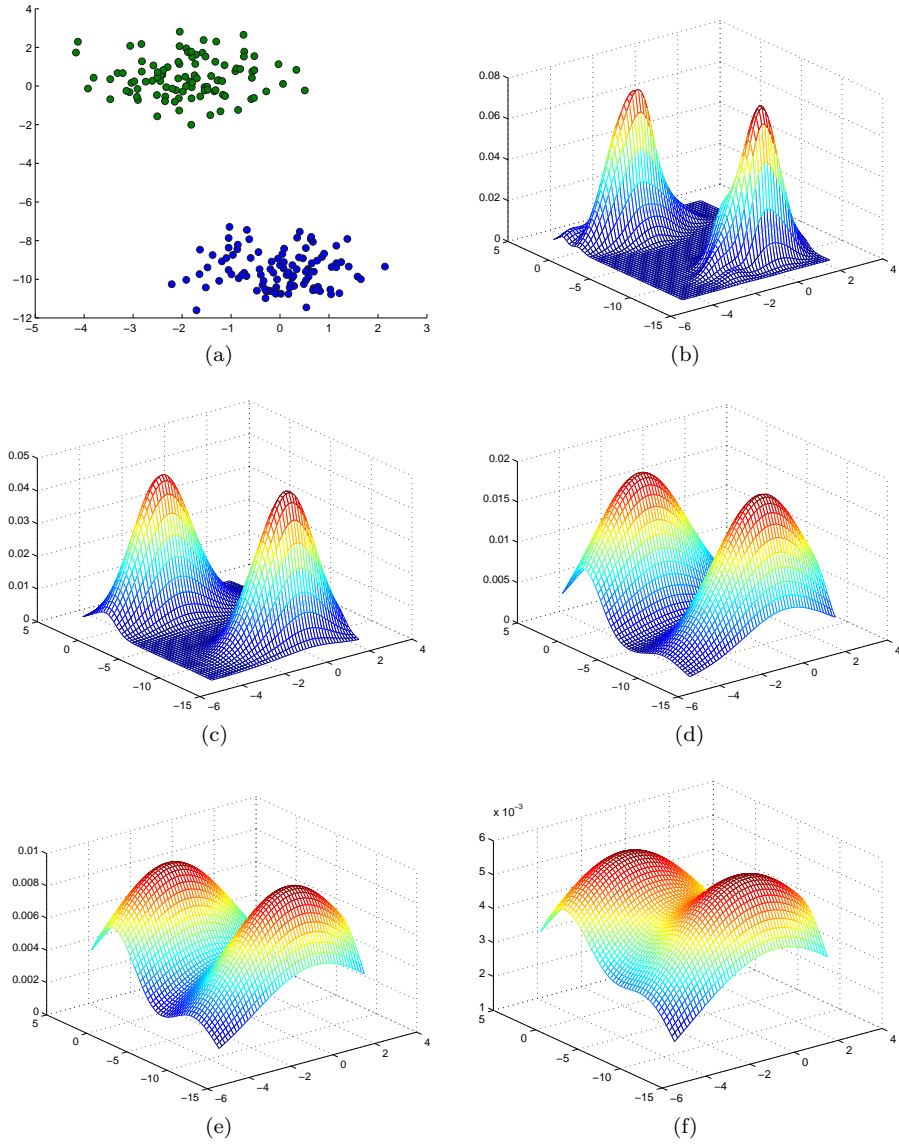


Figura 2.14: Estimaciones por ventana de Parzen de $p(\mathbf{x})$ utilizando distintos tamaños de kernel: (a) conjunto de datos con el cual se estima la pdf; (b) estimación de $p(\mathbf{x})$ con $\sigma^2 = \frac{1}{2}\sigma_{\text{opt}}^2$; (c) $\sigma^2 = \sigma_{\text{opt}}^2$; (d) $\sigma^2 = 2\sigma_{\text{opt}}^2$; (e) $\sigma^2 = 3\sigma_{\text{opt}}^2$; (f) $\sigma^2 = 4\sigma_{\text{opt}}^2$

Capítulo 3

Metodología

3.1. Introducción al nuevo método

En esta tesis se presenta un método para diseñar clasificadores no supervisados utilizando programación genética y teoría de la información. El uso de estas dos técnicas permite obtener clasificadores que separen los patrones de manera no paramétrica, es decir, sin asumir ninguna distribución *a priori* de los datos. Así se pueden detectar *clusters* con formas complejas y no sólo hiperesféricas. El esquema propuesto se puede resumir en los siguientes conceptos:

- Estructura multiárbol. Los individuos que evolucionan están constituidos por múltiples árboles. Cada árbol dentro de un individuo está asociado a uno de los *clusters* que se deben formar
- Interpretación probabilística. La salida de cada árbol dentro de un individuo se interpreta como la probabilidad de pertenencia al *cluster* que el árbol representa
- Medida de *fitness* basada en teoría de la información, que permite captar la verdadera estructura de los datos
- Selección adecuada de conjunto de terminales, funciones base, condiciones de término y demás parámetros que controlan la evolución

En conjunto, estos conceptos permiten obtener un método de *clustering* avanzado, que logra detectar *clusters* con cualquier forma y en múltiples dimensiones. En las secciones siguientes se analizan en

detalle cada uno de los componentes del esquema propuesto.

3.2. Representación multiárbol

En el diseño de clasificadores supervisados con PG se ha mostrado que el uso de individuos multiárbol es preferible a las técnicas de mapeo unidimensional como SRS ó DRS vistas en la sección (2.4.1). Algunas de las ventajas que presenta el enfoque multiárbol son:

- A diferencia de [21], se necesita una sola ejecución de PG para obtener una solución completa al problema.
- Comparado con SRS y DRS, los árboles resultantes son más simples, ya que cada árbol debe reconocer sólo los patrones pertenecientes a una sola clase. Esto, a su vez, facilita la convergencia.
- Presenta la posibilidad (al menos para el caso supervisado) de introducir nuevos operadores genéticos que ayuden a dirigir el proceso de evolución.
- La comprensión de los resultados es mucho más clara, ya que se trata de un conjunto de reglas separadas, en vez de una gran función que mapea las distintas clases en segmentos del espacio de salida.

Algunas de estas ventajas han sido demostradas en [28], donde además se obtienen desempeños que en la mayoría de los casos superan a otras técnicas no basadas en PG ampliamente usadas para clasificación.

Tomando en cuenta los buenos resultados del enfoque multiárbol para clasificación supervisada, en esta tesis se ha decidido utilizar el mismo enfoque como base para la aplicación de PG a *clustering*.

En el esquema propuesto cada individuo está formado por c árboles, cada uno de ellos asociado con uno de los *clusters* que se debe formar. Para formalizar la propuesta, se introduce la siguiente notación:

- Los individuos multiárbol serán representados por la letra I . El individuo i -ésimo de la población se representa por I_i .
- Los árboles se representarán por la letra T . Para representar cada árbol de un individuo I_i se usará la notación T_k^i , con $k = \{1, \dots, c\}$.

- Ya que los árboles representan funciones, se puede abusar de la notación y ocupar $T(\mathbf{x})$ para referirse a la salida del árbol T al evaluarlo en el patrón \mathbf{x} . Asimismo, la salida del individuo I se representa por $I(\mathbf{x})$.

La salida de un individuo I_i consiste en un vector que agrupa las salidas de todos los árboles del individuo, es decir, $I_i(\mathbf{x}) = [T_1^i(\mathbf{x}), \dots, T_c^i(\mathbf{x})]^T$. La salida de los individuos pasa a ser multidimensional (a diferencia de la salida unidimensional de PG clásica), con lo que se logra un mayor poder de representación, ya que se pueden evolucionar programas con múltiples salidas.

Junto con la nueva representación usada para los individuos, es necesario definir la interpretación que se dará a las salidas de éstos y los operadores genéticos que modificarán sus estructuras.

3.3. Interpretación probabilística

Los enfoques de PG multiárbol presentados hasta ahora en la literatura consideran árboles con salidas binarias, que indican si cada patrón presentado pertenece o no a la clase que el árbol representa. En la introducción se mostró que el objetivo final en el diseño de un clasificador era la obtención de un sistema que entregue de forma precisa la probabilidad de pertenencia de cada patrón a cada una de las clases. En definitiva, al entrenar un clasificador se busca un conjunto de funciones de probabilidad a posteriori $P(C_k|\mathbf{x})$, que representen fielmente a los datos. Es preferible, entonces que el sistema evolucione directamente estas funciones de probabilidad de pertenencia en vez de un conjunto de reglas binarias.

Con este fin, en esta tesis se introduce una interpretación probabilística para la salida de los árboles. Para un individuo dado, la salida de cada uno de sus árboles se interpreta como $T_k(\mathbf{x}) = P(C_k|\mathbf{x})$. Dado que la salida de un árbol puede ser cualquier valor real, no cumple con las dos propiedades básicas de las probabilidades:

1. $P(C_k|\mathbf{x}) \in [0, 1] \forall k = 1, \dots, c$.
2. $\sum_{k=1}^c P(C_k|\mathbf{x}) = 1$.

Sin embargo, estas propiedades se pueden forzar aplicando transformaciones a las salidas de los árboles, las que comúnmente se denominan funciones de activación, por su uso a la salida de cada neurona en redes neuronales [2]. En esta tesis se estudian dos funciones de activación:

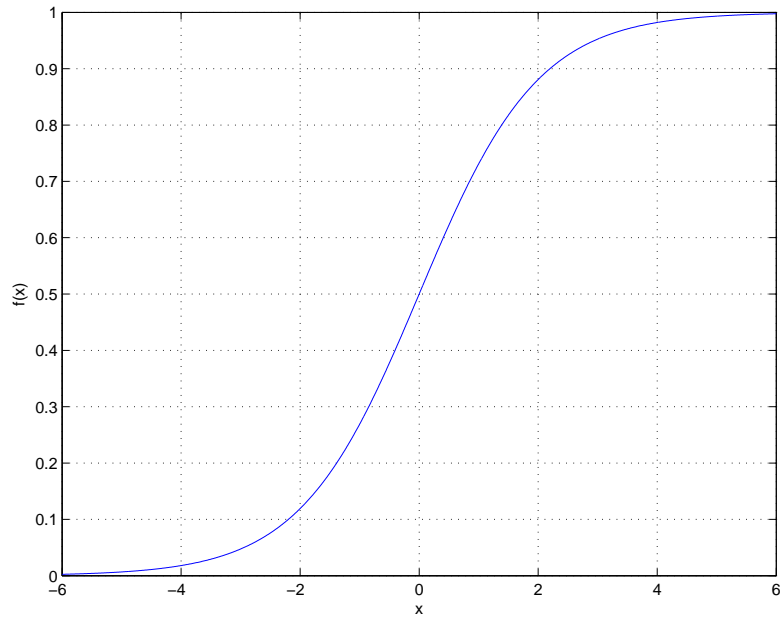


Figura 3.1: Función sigmoide logística

Máximo

Se obtiene el valor máximo para la salida de los c árboles, dado por

$$\hat{k} = \underset{k}{\operatorname{argmax}} T_k(\mathbf{x}) \quad (3.1)$$

Luego, la salida que corresponda al máximo se cambia por 1, dejando las demás en 0, es decir,

$$\tilde{T}_k(\mathbf{x}) = \begin{cases} 1 & k = \hat{k} \\ 0 & k \neq \hat{k} \end{cases} \quad (3.2)$$

Sigmoide logística

Esta función es de la forma $f(x) = \frac{1}{1+e^{-x}}$ y su representación gráfica se puede ver en la figura 3.1.

La salida transformada de un árbol queda:

$$\tilde{T}_k(\mathbf{x}) = \frac{1}{1 + e^{-T_k(\mathbf{x})}} \quad (3.3)$$

Luego de aplicar la función de activación, para cumplir la segunda propiedad de las probabilidades, se normaliza dividiendo por la suma de todas las salidas del individuo. Finalmente, es posible interpretar la salida de un árbol como la probabilidad de pertenencia a la clase dada por:

$$P(C_k|\mathbf{x}) = \frac{\tilde{T}_k(\mathbf{x})}{\sum_{k'=1}^c \tilde{T}_{k'}(\mathbf{x})} \quad (3.4)$$

Las dos funciones de activación buscan transformar la salida de los árboles en probabilidades, con un grado de continuidad variable. Mientras que la función máximo es completamente abrupta, la sigmoide logística es más suave.

Resumiendo, la salida de un individuo está dada por:

$$I_i(\mathbf{x}) = [P_i(C_1|\mathbf{x}), \dots, P_i(C_c|\mathbf{x})]^T \quad (3.5)$$

Es decir, se obtiene un conjunto de probabilidades de pertenencia a cada clase. La figura 3.2 muestra un esquema explicativo de los pasos involucrados en la interpretación probabilística. Para tomar una decisión con respecto a qué *cluster* pertenece un patrón \mathbf{x}_i , se debe aplicar la regla de Bayes, que consiste en elegir la clase $C_{\hat{k}}$ de manera que

$$\hat{k} = \underset{k}{\operatorname{argmax}} P(C_k|\mathbf{x}_i) \quad (3.6)$$

De esta forma se evita tener que definir reglas heurísticas para resolver conflictos cuando dos árboles se atribuyen la pertenencia del mismo ejemplo. Usando esta interpretación probabilística, se eliminan de plano estos conflictos.

Por otro lado, una de las grandes ventajas de utilizar PG para *clustering* es que el resultado del método son programas que pueden ser ejecutados posteriormente. En particular, en este caso, usando la interpretación probabilística se puede aplicar el teorema de Bayes sobre la salida de los árboles:

$$p(\mathbf{x}|C_k) = \frac{P(C_k|\mathbf{x})p(\mathbf{x})}{P(C_k)} \quad (3.7)$$

donde $p(\mathbf{x})$ se obtiene mediante el método de ventana de Parzen descrito en (2.47). La probabilidad a priori, $P(C_k)$, se asume uniforme para todas las clases (es decir, $P(C_k) = \frac{1}{c} \forall k$), ya que

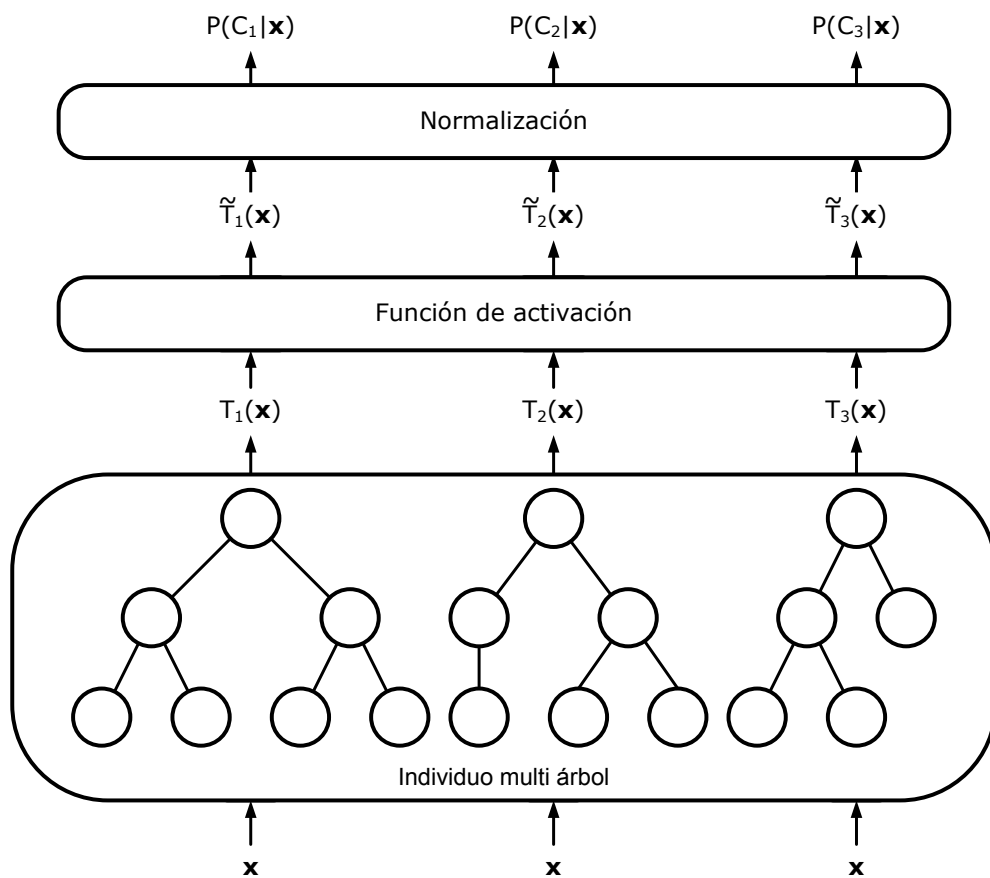


Figura 3.2: Diagrama explicativo de la interpretación probabilística para un individuo con tres árboles. Se debe recordar que \mathbf{x} es un vector de d dimensiones

corresponde a la mejor estimación posible en ausencia de información adicional [1]. Así se puede transformar la salida de un árbol en una estimación de la probabilidad condicional para cada clase, $p(\mathbf{x}|C_k)$. Dado que el árbol puede ser fácilmente evaluado en nuevos datos que no hayan sido presentados durante la evolución, se puede obtener esta probabilidad condicional para cualquier punto en el espacio de entrada. Es decir, el método propuesto obtiene una función que estima el modelo generador de los datos para cada una de las clases.

3.4. Medida de *fitness*

En la sección 2.7.3 se describe la divergencia de Cauchy-Schwartz (D_{CS}) entre dos distribuciones de probabilidad, cuya expresión matemática se muestra en (2.56). También se detalla cómo Jenssen et. al [9] adaptaron esta divergencia para aplicarla a problemas de *clustering*, creando la función de costos de Cauchy-Schwartz (J_{CS}) presentada en (2.62) mediante la introducción de las funciones de pertenencia m . Según el análisis visto en 2.7.3, J_{CS} cumple a cabalidad con las propiedades que debiera tener una función de costos para problemas de *clustering*. Debido a esto, en esta tesis se propone utilizar J_{CS} como medida de *fitness* para los individuos de PG.

Al utilizar funciones de pertenencia estrictas ($m_k(\mathbf{x}) \in \{0, 1\}$), la optimización de (2.62) presenta soluciones degeneradas que deben ser evitadas para obtener un *clustering* correcto. Por ejemplo, si todos los patrones son asignados a un mismo *cluster*, el numerador de (2.62) sería exactamente cero, convirtiéndose en un óptimo indeseado del problema. Para solucionar esto, en [9] las funciones de pertenencia son relajadas, es decir, extendidas al rango continuo $m(\mathbf{x}) \in [0, 1]$. Esto es compatible con la interpretación probabilística descrita en 3.3 para la salida de los árboles, por lo que las funciones de pertenencia pueden ser asociadas a la probabilidad a posteriori obtenida de la salida de un individuo. Así, al asignar el *fitness* a un individuo, las funciones de pertenencia en (2.62) pueden ser reemplazadas por la salida modificada de los árboles, imponiendo $m_k(\mathbf{x}) = P(C_k|\mathbf{x})$. Recordando que la salida de un individuo está dada por $I(\mathbf{x}) = [P(C_1|\mathbf{x}), \dots, P(C_c|\mathbf{x})]^T$, el *fitness* de un individuo I queda:

$$F(I) = \frac{\frac{1}{2} \sum_{i,j=1}^n (1 - I^T(\mathbf{x}_i) I(\mathbf{x}_j)) G_{ij, 2\sigma^2}}{\sqrt{\prod_{k=1}^c \sum_{i,j=1}^n P(C_k|\mathbf{x}_i) P(C_k|\mathbf{x}_j) G_{ij, 2\sigma^2}}} \quad (3.8)$$

el cual debe ser minimizado.

Se debe recordar que el valor utilizado para el tamaño del kernel es el que se obtiene al aplicar la regla de Silvermann descrita en la sección 2.7.4.

3.4.1. Detalles de implementación

A pesar de usar alguna de las funciones de activación descritas, pueden darse casos donde la pertenencia de algún patrón a todos los *clusters* sea igual o muy cercana a 0. Esto puede llevar a encontrar valores erróneos de *fitness* al normalizar dividiendo por valores muy pequeños, o encontrar un óptimo indeseado al asignar todos los patrones al mismo *cluster*. Para evitar esto, antes de la normalización mostrada en la ecuación (3.4), se suma un valor pequeño, ε , al valor de pertenencia para todos los *clusters*, preocupándose de que la probabilidad no exceda de 1. Es decir, en (3.4) se reemplaza $\tilde{T}_k(\mathbf{x})$ por:

$$\tilde{T}_k'(\mathbf{x}) = \max \{ \tilde{T}_k(\mathbf{x}) + \varepsilon, 1 \} \quad (3.9)$$

En todos los experimentos presentados acá se utiliza $\varepsilon = 0,05$.

Otro detalle a considerar en el cálculo del *fitness*, es que los valores de la evaluación del kernel gaussiano entre los pares de patrones permanecen constantes. Los únicos valores que cambian son los de las funciones de pertenencia, que se obtienen a través de la evaluación del individuo al que se calcula su *fitness*. Por lo tanto, es posible precalcular todos los valores $G_{ij,2\sigma^2}$ en una matriz G de tamaño $n \times n$ y almacenarlos en memoria durante toda la evolución, acelerando considerablemente el cálculo del *fitness*.

3.5. Operadores genéticos

La forma más simple de adaptar el operador de crossover para individuos multiárbol es que éste opere entre árboles seleccionados al azar de cada uno de los individuos. Es decir, si los padres con los cuales se va a operar son los individuos I_{i_1} y I_{i_2} , se deben seleccionar k_1 , k_2 al azar y realizar crossover normal entre los árboles $T_{k_1}^{i_1}$ y $T_{k_2}^{i_2}$. Los árboles resultantes reemplazarán a los originales en sus respectivas posiciones en cada uno de los individuos padre.

Asimismo, para un individuo I_{i_1} seleccionado para ser mutado, se selecciona k_1 al azar y se aplica la mutación estándar al árbol $T_{k_1}^{i_1}$.

Estos operadores se denominan “no guiados” ya que hacen una selección aleatoria de los árboles donde operan, sin ser guiados por información adicional. Estos siguen el mismo enfoque del crossover original, que intercambia ramas seleccionadas al azar entre dos árboles.

El concepto de operadores guiados presentado por Muni et al. [28] y revisado en la sección 2.4.3 no puede extenderse a un problema de *clustering*, ya que durante la ejecución del método, como no está disponible la información de las clases de los patrones, no es posible determinar a qué clase efectivamente representa cada árbol. Por lo tanto, sería incorrecto combinar siempre los árboles que se encuentran en la misma posición para distintos individuos. En esta tesis se experimentó con operadores que trataran de combinar árboles que representaran al mismo *cluster*, buscando la correspondencia entre los árboles de dos individuos distintos. Una posibilidad estudiada fue la de seleccionar el árbol del primer individuo al azar y luego seleccionar del segundo individuo el árbol que presentara las salidas más parecidas (según distancia euclidiana) al ya seleccionado. También se estudió aplicar el concepto de *unfitness* para poder seleccionar con mayor probabilidad a los peores árboles dentro de un individuo para modificarlos. La medida de *unfitness* para un árbol se definió como el denominador de (3.8), considerando sólo los patrones asignados al *cluster* representado por el árbol. Es decir, se estima la entropía del *cluster* definido por el árbol; a mayor entropía, mayor *unfitness*. Analizadas desde el punto de vista del número de generaciones requeridas para encontrar la solución correcta, ninguna de estas propuestas resultó beneficiosa, sólo logrando aumentar el costo computacional del método. Por esta razón, las variantes mencionadas no se consideraron para los experimentos presentados en esta tesis.

3.6. Conjunto de terminales y funciones base

El conjunto de funciones base utilizado en esta tesis es $F = \{+, -, \times, /, (\cdot)^2\}$. El conjunto de terminales consiste en $T = \{v_l, R\}$, donde v_l , con $l = \{1, \dots, d\}$, son las variables que describen los datos y R son constantes aleatorias efímeras dentro del rango $[0, 100]$.

3.7. Parámetros que controlan la evolución

En la tabla 3.1 se muestran los principales parámetros utilizados en las simulaciones. El método de selección escogido es selección por torneo.

Tabla 3.1: Valores de parámetros de PG comunes a todas las simulaciones

| Parámetro | Valor |
|------------------------------|-------|
| Máximo de generaciones | 200 |
| Tamaño Población | 1000 |
| Largo máximo árboles | 8 |
| Tamaño torneo | 5 |
| Probabilidad de crossover | 0,85 |
| Probabilidad de mutación | 0,05 |
| Probabilidad de Reproducción | 0,1 |

El criterio de detención utilizado es el número máximo de generaciones y el método para designar el resultado es elegir al mejor individuo encontrado durante toda la evolución.

3.8. Reducción de la complejidad

La complejidad de la función de *fitness* presentada en (3.4) es $O(n^2)$, con n el número de patrones. Esto significa que el número de operaciones necesarias para calcular el *fitness* aumenta con el cuadrado de n . Si además se considera que debe ser calculada para cada individuo en la población, una vez por generación, se concluye rápidamente que el método propuesto puede ser bastante costoso computacionalmente. A continuación se describen dos técnicas estudiadas para reducir la complejidad de la medida de *fitness*, que pretenden acelerar la ejecución del método sin afectar demasiado su desempeño. Estas dos técnicas de reducción de complejidad se pueden aplicar de forma independiente o ambas simultáneamente. Su efecto en los tiempos de cómputos y en el desempeño del método se analiza en detalle más adelante.

3.8.1. Muestreo estocástico

Esta técnica consiste en considerar sólo un subconjunto de los patrones para el cálculo del *fitness*. Previo a cada generación, se seleccionan aleatoriamente M patrones, los cuales se usarán para estimar la divergencia de Cauchy-Schwartz. Es importante destacar que el mismo subconjunto debe ser usado para la evaluación de *todos* los individuos en la misma generación, comparándolos así en igualdad de condiciones. Usando este *fitness* aproximado, se selecciona el mejor individuo de la generación, el cual será candidato a convertirse en el mejor individuo hasta ahora. Pero se

debe considerar que los individuos fueron evaluados usando subconjuntos de datos distintos, por lo que para hacer una comparación justa, se debe evaluar nuevamente el candidato usando todos los patrones para verificar si su *fitness* supera al del mejor individuo hasta ahora. Así se evita quedarse con un individuo subóptimo, que sólo supera al mejor hasta ahora en un subconjunto de los datos.

Se debe recordar que para obtener la fórmula de la función de *fitness* (3.8), para cada patrón se estima la densidad de probabilidad calculando las distancias a todos los demás patrones y es por esto que las sumatorias en (3.8) son dobles. El muestreo estocástico no cambia la forma de estimar la densidad de probabilidad, por lo que igualmente para cada patrón se evalúa el kernel gaussiano con respecto a todos los demás patrones. Es decir, sólo se hace un muestreo de los patrones utilizados en la primera sumatoria:

$$F_M(I) = \frac{\frac{1}{2} \sum_i^M \sum_{j=1}^n (1 - I^T(\mathbf{x}_i) I(\mathbf{x}_j)) G_{ij,2\sigma^2}}{\sqrt{\prod_{k=1}^c \sum_i^M \sum_{j=1}^n P(C_k|\mathbf{x}_i) P(C_k|\mathbf{x}_j) G_{ij,2\sigma^2}}} \quad (3.10)$$

Con esto, el número de operaciones se reduce de n^2 a $M \cdot n$. Normalmente, se tendrá que M es una fracción de n , por ejemplo, $M = 0,2n$. Así, la medida de *fitness* sigue siendo de orden cuadrático, ya que $O(Mn) = O(0,2n^2) = O(n^2)$. No obstante, esta técnica puede acelerar considerablemente los cálculos.

3.8.2. Estimación por vecindad

Esta técnica concierne al cálculo de las segundas sumatorias en (3.8). Dado que el kernel gaussiano decae rápidamente al aumentar la distancia entre dos patrones, no es necesario incluir los pares de patrones más lejanos entre sí en el cálculo del *fitness*, ya que su contribución es casi nula. Por lo tanto, esta técnica consiste en considerar una vecindad para estimar la densidad de probabilidad en cada punto. Para cada patrón \mathbf{x}_i , se define una vecindad

$$\Omega(\mathbf{x}_i) = \{\mathbf{x} / |\mathbf{x} - \mathbf{x}_i| < r_\Omega\} \quad (3.11)$$

Dada esta definición, la función de *fitness* se puede expresar como:

$$F_\Omega(I) = \frac{\frac{1}{2} \sum_i^m \sum_{\mathbf{x}_j \in \Omega(\mathbf{x}_i)} (1 - I^T(\mathbf{x}_i) I(\mathbf{x}_j)) G_{ij,2\sigma^2}}{\sqrt{\prod_{k=1}^c \sum_i^m \sum_{\mathbf{x}_j \in \Omega(\mathbf{x}_i)} P(C_k|\mathbf{x}_i) P(C_k|\mathbf{x}_j) G_{ij,2\sigma^2}}} \quad (3.12)$$

Una forma de calcular un radio r_Ω óptimo para la vecindad es usar la técnica del radio pareto de Ultsch [103]. Sin embargo, en general con esta técnica se obtuvieron vecindades demasiado pequeñas, que afectaban negativamente los resultados. Por lo tanto, se optó por una regla más simple que definiera r_Ω como una función del tamaño del kernel utilizando, que, recordando, corresponde al tamaño óptimo según la regla de Silvermann vista en la sección 2.7.4. En todos los experimentos de estimación por vecindad mostrados en esta tesis, se utilizó $r_\Omega = 3,5\sigma_{\text{opt}}^2$.

3.9. Experimentos

En esta sección se detallan una serie de experimentos diseñados para comprobar la eficacia del método propuesto en forma global, además de experimentos específicos para analizar ciertos conceptos de forma aislada.

Los experimentos principales consisten en evaluar el método propuesto mediante simulaciones computacionales en distintas bases de datos benchmark o artificiales. Los distintos experimentos han sido diseñados con los siguientes objetivos:

- Mostrar que el método propuesto es capaz de realizar un *clustering* adecuado en conjuntos con distintas formas, número de patrones y dimensiones.
- Comparar el desempeño del método con otros métodos conocidos, en particular con el algoritmo *k-means* [45] y el método original de Jenssen [9].
- Mostrar que el método es capaz de estimar el modelo generador de los datos y usarlo para clasificar nuevos patrones.

Para este fin se han diseñado los experimentos de *clustering*, de generalización y de aprendizaje de la densidad de probabilidad condicional que se detallan más adelante. Además, se realizan experimentos para comprobar la eficacia de las técnicas de reducción de la complejidad.

Para poder evaluar de manera cuantitativa, todos los experimentos se realizan con bases que han sido previamente etiquetadas por un experto. Además, en todas las pruebas, el número de *clusters*, c , se fijó igual al número real de clases presentes en la base.

Para todos los experimentos con PG se utilizó la herramienta GPalta [104], desarrollada en conjunto con esta tesis. Para los experimentos con el método de Jenssen, se utilizó el código implementado por él mismo, que a la fecha se puede descargar de [105]. Para los experimentos con

Tabla 3.2: Parámetros utilizados en los experimentos para el método de Jenssen, obtenidos de [10]

| Parámetro | Valor |
|---|--------------------------|
| Tamaño inicial del kernel | $2\sigma_{\text{opt}}$ |
| Tamaño final del kernel | $0,5\sigma_{\text{opt}}$ |
| Iteraciones de <i>annealing</i> | 200 |
| Porcentaje de datos para muestreo estocástico | 20 % |

k-means se utilizó la implementación incluida en el *toolbox* de estadísticas de *Matlab*. Los experimentos se ejecutaron en un computador de escritorio con procesador *Athlon* 64 de 2 GHz de frecuencia de reloj y 1 GB de memoria RAM, bajo ambiente *Linux* (distribución *Ubuntu*) de 32 bits.

3.9.1. *Clustering*

En este experimento se evalúa la tarea fundamental para la que fue diseñado el método propuesto. El experimento consiste en realizar un *clustering* de distintas bases de datos. Para cada base, se ejecuta el método usando todos los patrones disponibles. El individuo ganador se evalúa en cada patrón, asignando una etiqueta de acuerdo al máximo de las funciones de pertenencia representadas por el individuo (ver sección 3.3). Las etiquetas así obtenidas se comparan con las etiquetas reales de acuerdo a los indicadores definidos más adelante en la sección 3.9.5. Debido a que PG es un proceso estocástico, es probable que distintas ejecuciones del algoritmo conlleven a soluciones distintas. Para tomar en cuenta las variaciones en el resultado, el método es ejecutado 40 veces.

El algoritmo *k-means* y el método de Jenssen son ejecutados en las mismas condiciones y sus resultados son comparados con el método propuesto. El método de Jenssen se ejecuta utilizando los parámetros descritos en [10], que se reproducen en la tabla 3.2. Para el caso de PG, también se comparan las dos funciones de activación descritas en la sección 3.3.

3.9.2. Generalización

En este experimento se intenta evaluar si el método propuesto es capaz de aprender la distribución de cada uno de las clases presentes en los datos. Con este propósito, el método es ejecutado usando sólo un subconjunto de los datos, denominado conjunto de entrenamiento, y evaluado en los patrones restantes, que forman un conjunto de validación. El objetivo es que el método sea capaz

de aprender el modelo generador de los datos usando sólo el conjunto de entrenamiento y luego aplicar este modelo para clasificar correctamente los patrones del conjunto de validación.

Para no sesgar los resultados ocupando siempre los mismos conjuntos de entrenamiento y validación, se realiza un procedimiento muy común en reconocimiento de patrones, denominado validación cruzada (tenfold cross validation) [2, 3]. Este procedimiento consiste en dividir aleatoriamente la base en diez subconjuntos y ejecutar diez veces el método a evaluar. En cada ejecución, se selecciona uno de los subconjuntos como conjunto de validación, dejándolo fuera del entrenamiento. Los restantes nueve subconjuntos forman el conjunto de entrenamiento, con el cual se realiza la evolución. Luego de cada ejecución el individuo ganador se evalúa tanto en los patrones del conjunto de entrenamiento como en los del de validación. A cada patrón se le asigna una etiqueta, las cuales se comparan con las etiquetas reales. En ambos conjuntos se calcula por separado el porcentaje de clasificaciones correctas. El resultado final corresponde a los promedios de clasificaciones a través de las 10 ejecuciones para ambos conjuntos. Un método con buena capacidad de generalización debiera mantener resultados similares en ambos conjuntos. Para considerar distintas particiones aleatorias de los datos, la validación cruzada se repite 6 veces (se eligen 3 particiones y cada una se utiliza dos veces), lo que da un total de 60 ejecuciones de PG para cada base de datos.

Cabe notar que en esta tarea (generalización), no se pueden comparar los resultados con los otros dos métodos (*k-means* y Jenssen), ya que éstos no contemplan la generación de funciones de pertenencia que permitan clasificar nuevos patrones.

3.9.3. Aprendizaje de la densidad de probabilidad condicional

El objetivo de este experimento es mostrar que el método propuesto es capaz de estimar la densidad de probabilidad condicional para cada clase de los datos. Con este fin, el método se entrena usando un conjunto de datos de dos dimensiones especialmente diseñada para evaluar fácilmente el resultado de forma gráfica. La base consta de cuatro *clusters* fácilmente separables, cada uno con 100 patrones. Los *clusters* fueron generados artificialmente usando distribuciones gaussianas. En la figura 3.3 se muestra el conjunto de datos utilizado para este experimento.

El experimento consiste en ejecutar PG y luego evaluar el individuo ganador en una grilla equiespaciada de puntos dentro del rango de los datos de entrenamiento. Esto permite obtener una representación gráfica de las funciones de probabilidad *a posteriori*, $P(C_k|\mathbf{x})$, obtenidas a través

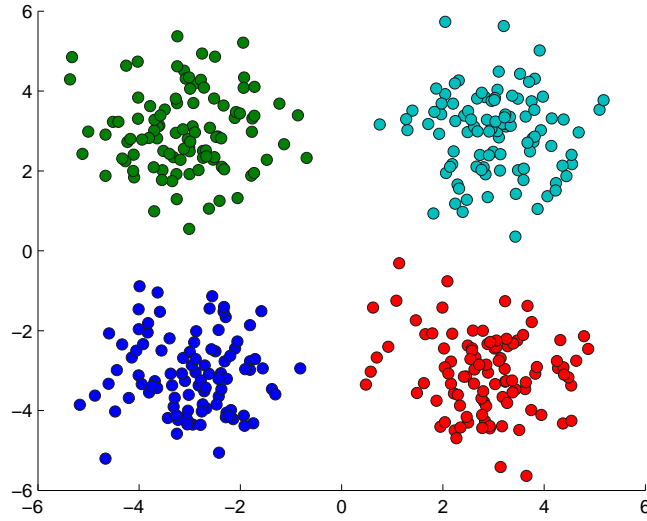


Figura 3.3: Conjunto de datos simple con cuatro *clusters*, utilizado en el experimento de aprendizaje de la densidad de probabilidad condicional

de las salidas de los árboles en todo el espacio de dos dimensiones. Luego, aplicando el teorema de Bayes sobre estas probabilidades y la estimación por ventana de Parzen de $p(\mathbf{x})$ para cada punto en la grilla, las salidas de los árboles se transforman en el conjunto de funciones de densidad de probabilidad condicional, $p(\mathbf{x}|C_k)$, para cada uno de los *clusters*. Se espera que la representación gráfica de las pdf condicionales muestre una sola distribución gaussiana para cada clase.

3.9.4. Bases de datos

Para evaluar el desempeño del método propuesto, éste fue probado en una serie de bases de datos de distintas características, incluyendo bases de datos reales comúnmente usadas como *benchmark* y bases artificiales especialmente diseñadas para *clustering*. Las bases *benchmark* utilizadas en esta tesis son Iris, Wisconsin Breast Cancer (WBC), Wine y Pendigits. Todas ellas pueden ser obtenidas desde el repositorio de bases de datos de la UCI [106], aunque se debe tener en cuenta que para Iris existen pequeñas variaciones en distintas versiones de la base [107]. En total se utilizaron 12 conjuntos que contienen desde 150 a 4096 patrones y de 2 a 16 dimensiones.

Iris

Este conjunto corresponde a la conocida base Iris recolectada por Anderson [108], pero publicada en extensión por primera vez por Fisher [109]. La base contiene 150 patrones en cuatro dimensiones

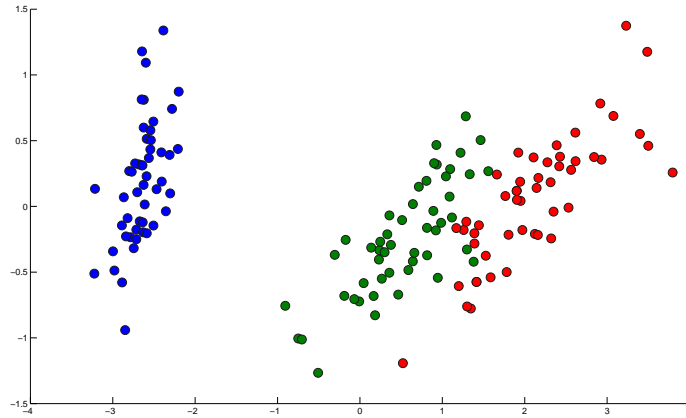


Figura 3.4: Proyección en dos dimensiones de la base Iris mediante PCA [44]

medidos en flores de tres especies distintas (Iris Setosa, Iris Versicolour e Iris Virginica). Las cuatro características medidas son el ancho y largo de los sépalos y pétalos de cada flor. Cada clase está representada por 50 patrones. Esta base es una de las más difundidas en el ámbito de reconocimiento de patrones y ha sido ampliamente utilizada como *benchmark* en numerosas publicaciones sobre clasificación y *clustering*. La base Iris es considerada un problema de fácil solución, ya que los patrones se agrupan de forma natural en tres *clusters*, uno de ellos (clase 1) completamente separado de los otros dos, mientras que las clases 2 y 3 se traslapan un poco. En la figura 3.4 se puede ver una proyección muy simple a dos dimensiones utilizando análisis de componentes principales (PCA, por sus siglas en inglés) [44]. Los resultados típicos de reclasificación publicados usando algoritmos de *clustering* en esta base reportan entre 14 y 17 errores, aunque también es común que algunos métodos encuentren sólo dos *clusters* [52].

Wisconsin Breast Cancer (WBC)

Esta base contiene 699 instancias de 9 atributos medidos en pacientes posiblemente con cáncer de mamas. Los atributos tienen valores discretos entre 1 y 10. Cada instancia está etiquetada como benigna (65,5 %) o maligna (34,5 %). Dieciséis instancias fueron removidas ya que contienen valores incompletos.

Wine

Esta base corresponde a mediciones sobre propiedades químicas de tres tipos de vinos. Los vinos fueron cultivados en la misma región de Italia, pero provienen de diferentes cepas. La base contiene

178 instancias, cada una con 13 atributos continuos, que corresponden a intensidad del color, nivel de alcohol, magnesio, etc. Las clases contienen 71, 59 y 48 patrones, respectivamente.

Pendigits

Esta base corresponde a dígitos manuscritos digitalizados con un tablero sensible a la presión. Los datos originales (Pendigits-orig) consisten en las coordenadas (x, y) donde el dispositivo detecta presión, ordenados través del tiempo (es decir, el trazo dibujado por la persona). Los datos normalizados (ocupados acá) corresponden a un muestro equiespaciado de los trazos para obtener sólo ocho pares de coordenadas representativas, con lo que cada muestra contiene 16 atributos. Todos los atributos son enteros entre 0 y 100. La base está separada en 7494 patrones de entrenamiento y 3498 de prueba. Con el fin de comparar los resultados con [10], sólo se ocupan los patrones correspondientes a los dígitos 0, 1 y 2 del conjunto de prueba, con 363, 364 y 364 muestras respectivamente.

Fundamental Clustering Problem Suite (FCPS)

Esta base contiene una serie de conjuntos de datos artificiales diseñados por Ultsch [110] que tratan de representar distintos los problemas que puede enfrentar un algoritmo de *clustering*. Cada conjunto fue diseñado para evaluar escenarios específicos donde cierto tipo de algoritmos de *clustering* suelen fallar. Además, todos los conjuntos son visualizables (es decir, son de dimensión 2 ó 3), lo que permite evaluar gráficamente los resultados. La tabla 3.3 muestra las principales características de los 8 conjuntos de datos tomados de esta base, mientras la figura 3.5 muestra una representación gráfica de ellos. La base FCPS contiene dos conjuntos que no fueron usados en esta tesis. Éstos son el conjunto denominado “GolfBall”, que no fue utilizado ya que los datos no forman ningún *cluster* (equivalentemente se podría decir que forman sólo uno) y el conjunto “Target”, que fue descartado porque algunos de los *cluster* etiquetados están formados por sólo 3 patrones. Ver [110] para más detalles sobre la base FCPS.

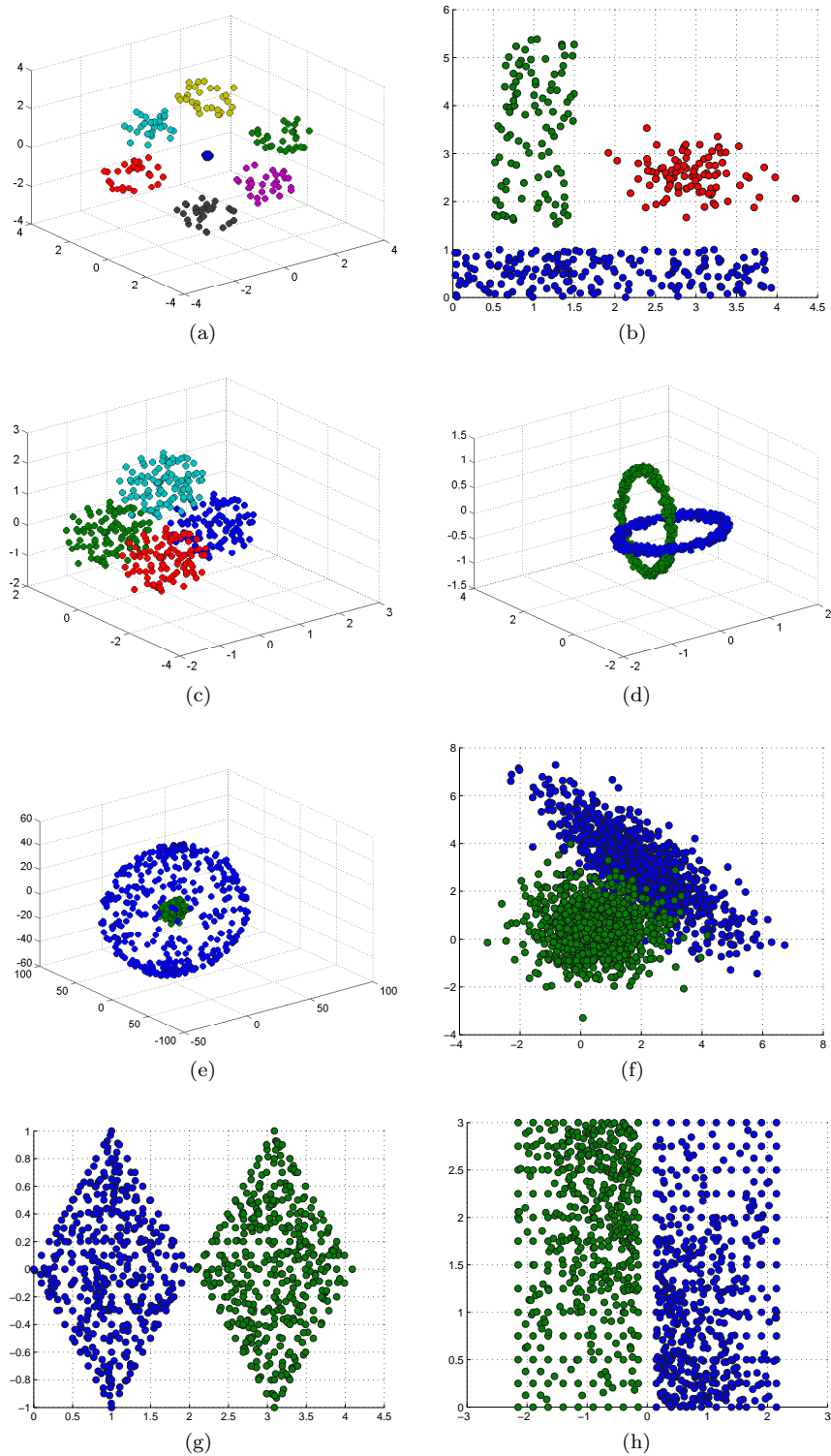


Figura 3.5: Conjuntos de datos de la base “Fundamental Clustering Problem Suite” (FCPS) [110]: (a) Hepta; (b) Lsun; (c) Tetra; (d) Chainlink; (e) Atom; (f) EngyTime; (g) TwoDiamonds; (h) WingNut

Tabla 3.3: Resumen de Propiedades de Conjuntos de Datos utilizados de la Base FCPS

| Conjunto | Tamaño | Dimensiones | Clases | Principal dificultad |
|-------------|--------|-------------|--------|--|
| Hepta | 212 | 3 | 7 | Diferentes densidades para los <i>clusters</i> |
| Lsun | 400 | 2 | 3 | Distintas varianzas en los <i>clusters</i> (<i>clusters</i> alargados vs. esféricos) |
| Tetra | 400 | 3 | 4 | <i>Clusters</i> densos y muy juntos |
| Chainlink | 1000 | 3 | 2 | <i>Clusters</i> con estructura compleja y no linealmente separables |
| Atom | 800 | 3 | 2 | <i>Clusters</i> no linealmente separables. Distintas varianzas en cada <i>cluster</i> |
| EngyTime | 4096 | 2 | 2 | <i>Clusters</i> definidos por densidad |
| TwoDiamonds | 800 | 2 | 2 | <i>Clusters</i> que se tocan en una esquina |
| WingNut | 1070 | 2 | 2 | Mayores densidades en los bordes de los <i>clusters</i> |

3.9.5. Indicadores de desempeño

Tal como se describe en la sección 2.2.2, existen muchas formas de evaluar el desempeño de un método de *clustering*. Si existen etiquetas previamente asignadas por un experto, lo más apropiado es utilizar el criterio externo [3]. Este criterio consiste en comparar la partición obtenida por el método de *clustering* con la partición “real”, determinada por las etiquetas disponibles. De los índices de validez externa, el más utilizado es el índice de Rand [54], especialmente en su versión ajustada [55]. El índice ajustado de Rand entrega un número entre 0 y 1 que indica la probabilidad de que la partición obtenida corresponda a la correcta.

Por otro lado, si el número de *clusters* detectados es igual al número de clases presentes en los datos, es posible traducir la pertenencia a un *cluster* en una etiqueta de clase y calcular la razón de clasificaciones correctas. Este proceso de traducción es necesario porque no es posible saber de antemano cómo se relacionan los *clusters* con la clases presentes en los datos. Es decir, durante la ejecución del algoritmo, no se sabe con qué clase está asociado cada uno de los *clusters*. Intuitivamente, una forma de realizar la traducción es asociar un *cluster* con la clase correspondiente a la mayoría de los patrones seleccionados dentro del *cluster*. De forma inversa, se puede asociar un *cluster* con una clase si es que la mayoría de los patrones de la clase fueron seleccionados en ese *cluster*. Estos criterios son complementarios y de preferencia deben aplicarse ambos simultáneamente. Este proceso funciona bien cuando las diferencias entre las particiones son relativamente pequeñas,

pudiendo producirse conflictos de forma contraria. La mayoría de las publicaciones ocupan esta interpretación al especificar el porcentaje de patrones correcta o incorrectamente clasificados por un algoritmo, aunque el proceso de traducción generalmente se pasa por alto. Ver, por ejemplo [93]. En algunas publicaciones se habla de porcentaje de reclasificación si se cuentan los aciertos o error de resubstitución si se cuentan los errores.

En esta tesis se calcula tanto la razón de clasificaciones correctas como el índice de Rand ajustado. El primer índice entrega una medida intuitiva para evaluar el resultado del método y es posible calcularlo dado que en todos los experimentos se especifica un número de *clusters* igual al número de clases. El segundo índice, aunque menos intuitivo, es útil para realizar comparaciones con otros resultados publicados en la literatura.

3.9.6. Reducción de complejidad

Estas pruebas consisten en ejecutar nuevamente los experimentos de *clustering*, pero agregando los métodos de reducción de complejidad descritos en la sección 3.8. Se comparan los casos en que no se aplica reducción de complejidad versus aplicar estimación por vecindad, muestreo estocástico y ambos simultáneamente. Para cada caso se miden los indicadores de desempeño, con la intención de evaluar si las modificaciones afectan el resultado final.

Además, se realizan mediciones del tiempo que toma evaluar el *fitness* de un individuo considerando las distintas variantes. Sólo se toma en cuenta la evaluación del *fitness* en vez de medir el tiempo que toma la evolución completa, ya que este último es un proceso altamente aleatorio.

Capítulo 4

Resultados

4.1. *Clustering*

En la tabla 4.1 se muestran los resultados del experimento de *clustering* utilizando PG con la función de activación máximo. En la tabla 4.2 se muestran los resultados para la misma prueba, pero usando la función de activación sigmoide logística. Para comparar el desempeño del método propuesto, se realizan los mismos experimentos utilizando *k-means* y el método de Jenssen [9, 10]. Los resultados se muestran en las tablas 4.3 y 4.4, respectivamente. Para todos los métodos, se especifica *a priori* el número de *clusters* a encontrar, que es igual al número de clases presentes en cada conjunto. Las tablas muestran el porcentaje de clasificación obtenido según el procedimiento descrito en la sección 3.9.5 y el índice de Rand ajustado. Para ambos indicadores, se muestra el promedio, desviación estándar y mejor caso considerando 40 ejecuciones en cada conjunto de datos.

A continuación se realiza un breve análisis comparativo de los resultados obtenidos en cada una de las bases con los tres métodos evaluados.

Tabla 4.1: Resultados experimento de *clustering* usando PG y función de activación máximo

| Conjunto de datos | Clasificación (%) | | | Índice Rand ajustado | | |
|-------------------|--------------------|------------|-------|----------------------|------------|-------|
| | Promedio | Desviación | Mejor | Promedio | Desviación | Mejor |
| Iris | 90,07 | 0,20 | 90,67 | 0,75 | 0 | 0,76 |
| WBC | 96,66 | 0,14 | 97,07 | 0,87 | 0,01 | 0,89 |
| Wine | 94,45 | 1,21 | 97,75 | 0,85 | 0,03 | 0,93 |
| Pentest3 | 82,09 | 2,11 | 85,52 | 0,6 | 0,04 | 0,67 |
| EngyTime | 96 | 0,02 | 96,04 | 0,85 | 0 | 0,85 |
| Tetra | 100 | 0 | 100 | 1 | 0 | 1 |
| Hepta | 100 | 0 | 100 | 1 | 0 | 1 |
| Lsun | 96,69 | 8,26 | 100 | 0,93 | 0,18 | 1 |
| TwoDiamonds | 100 | 0 | 100 | 1 | 0 | 1 |
| WingNut | 96,88 | 0,22 | 98,23 | 0,88 | 0,01 | 0,93 |
| Chainlink | 99,97 | 0,09 | 100 | 1 | 0 | 1 |
| Atom | 100 | 0 | 100 | 1 | 0 | 1 |

Tabla 4.2: Resultados experimento de *clustering* usando PG y función de activación sigmoide logística

| Conjunto de datos | Clasificación (%) | | | Índice Rand ajustado | | |
|-------------------|--------------------|------------|-------|----------------------|------------|-------|
| | Promedio | Desviación | Mejor | Promedio | Desviación | Mejor |
| Iris | 92,13 | 2,41 | 98 | 0,79 | 0,06 | 0,94 |
| WBC | 69,76 | 28,06 | 97,21 | 0,45 | 0,41 | 0,89 |
| Wine | 86,05 | 5,69 | 94,94 | 0,64 | 0,13 | 0,85 |
| Pentest3 | 81,3 | 3,48 | 85,43 | 0,59 | 0,05 | 0,67 |
| EngyTime | 95,95 | 0,32 | 96,29 | 0,84 | 0,01 | 0,86 |
| Tetra | 99,58 | 1,08 | 100 | 0,99 | 0,03 | 1 |
| Hepta | 99,63 | 2,24 | 100 | 1 | 0,02 | 1 |
| Lsun | 90,74 | 18,26 | 100 | 0,88 | 0,23 | 1 |
| TwoDiamonds | 100 | 0 | 100 | 1 | 0 | 1 |
| WingNut | 96,91 | 0,21 | 97,64 | 0,88 | 0,01 | 0,91 |
| Chainlink | 79,97 | 3,35 | 100 | 0,36 | 0,11 | 1 |
| Atom | 100 | 0 | 100 | 1 | 0 | 1 |

Tabla 4.3: Resultados experimento de *clustering* usando *k-means*

| Conjunto de datos | Clasificación (%) | | | Índice Rand ajustado | | |
|-------------------|-------------------|------------|-------|----------------------|------------|-------|
| | Promedio | Desviación | Mejor | Promedio | Desviación | Mejor |
| Iris | 79,98 | 15,37 | 89,33 | 0,65 | 0,13 | 0,73 |
| WBC | 96,04 | 0 | 96,04 | 0,85 | 0 | 0,85 |
| Wine | 95,56 | 6,75 | 96,63 | 0,88 | 0,09 | 0,9 |
| Pentest3 | 71,47 | 12,96 | 83,23 | 0,51 | 0,07 | 0,59 |
| EngyTime | 95,14 | 0 | 95,14 | 0,82 | 0 | 0,82 |
| Tetra | 100 | 0 | 100 | 1 | 0 | 1 |
| Hepta | 78,16 | 16,11 | 100 | 0,77 | 0,18 | 1 |
| Lsun | 76,16 | 0,24 | 76,5 | 0,43 | 0 | 0,44 |
| TwoDiamonds | 100 | 0 | 100 | 1 | 0 | 1 |
| WingNut | 96,36 | 0 | 96,36 | 0,86 | 0 | 0,86 |
| Chainlink | 65,56 | 0,3 | 65,9 | 0,1 | 0 | 0,1 |
| Atom | 70,58 | 1,42 | 72,13 | 0,17 | 0,02 | 0,2 |

 Tabla 4.4: Resultados experimento de *clustering* usando el método de Janssen con parámetros utilizados en [10]

| Conjunto de datos | Clasificación (%) | | | Índice Rand ajustado | | |
|-------------------|-------------------|------------|-------|----------------------|------------|-------|
| | Promedio | Desviación | Mejor | Promedio | Desviación | Mejor |
| Iris | 90,17 | 0,29 | 90,67 | 0,75 | 0,01 | 0,76 |
| WBC | 39,33 | 4,9 | 69,5 | 0,04 | 0,02 | 0,15 |
| Wine | 92,01 | 2,01 | 96,63 | 0,77 | 0,05 | 0,9 |
| Pentest3 | 78,06 | 7,16 | 85,43 | 0,55 | 0,08 | 0,67 |
| EngyTime | 96,07 | 0,07 | 96,24 | 0,85 | 0 | 0,86 |
| Tetra | 100 | 0 | 100 | 1 | 0 | 1 |
| Hepta | 100 | 0 | 100 | 1 | 0 | 1 |
| Lsun | 64,64 | 11,95 | 94,75 | 0,45 | 0,12 | 0,85 |
| TwoDiamonds | 100 | 0 | 100 | 1 | 0 | 1 |
| WingNut | 98,89 | 1,12 | 99,9 | 0,96 | 0,04 | 1 |
| Chainlink | 60,56 | 14,83 | 96,4 | 0,13 | 0,26 | 0,86 |
| Atom | 81,83 | 22,61 | 100 | 0,6 | 0,49 | 1 |

Iris

En esta base, PG logra un desempeño casi idéntico al método de Jenssen al usar función de activación máximo, mientras que al usar sigmoide logística lo supera ampliamente. La clasificación promedio obtenida por *k-means* es baja debido a que en varias ejecuciones este método encuentra sólo dos *clusters*.

WBC

En este conjunto el desempeño obtenido por PG con función de activación máximo es muy similar al de *k-means*, siendo ambos muy superiores al método de Jenssen, que nunca se acerca siquiera a la partición ideal. Cabe mencionar que en [9] se menciona un desempeño de 94,5 % de clasificaciones correctas. Haciendo un ajuste manual de los parámetros, sólo fue posible obtener un 90,05 % de clasificación promedio, con una desviación de 10,57 % y un máximo de 96,19 % si se fija el tamaño del kernel inicial en $5\sigma_{\text{opt}}$. Al usar sigmoide logística, el desempeño de PG baja considerablemente. Analizando los valores de *fitness* obtenidos por el mejor individuo de cada ejecución, se observa que los programas que generan una buena clasificación (sobre 96 %) tienen siempre un mejor valor de *fitness* que aquellos que obtienen una mala clasificación. Es decir, la función de *fitness* asigna correctamente un mejor valor a mejores clasificaciones. La baja clasificación promedio se explica porque un resultado aceptable se obtiene sólo en 19 de las 40 las pruebas, por lo que se concluye que con esta función de activación el método se estanca reiteradamente en un mínimo local. La tabla 4.5 muestra algunos ejemplos de porcentajes de clasificación obtenidos en este conjunto con PG y sigmoide logística, y sus respectivos valores de *fitness*. Usando la función de activación máximo, en cambio, siempre se obtiene una buena clasificación. Una posible explicación para este fenómeno es que al utilizar el máximo, deja de importar el valor absoluto de probabilidad que se asigne para cada *cluster*, importando sólo cual de ellos presenta el mayor valor. Con esto se reduce considerablemente el espacio de búsqueda, ya que posiblemente existen muchas configuraciones de programas que generan la misma clasificación, pero cuyos valores exactos de probabilidad no les permiten obtener un buen *fitness*. Esta propiedad se repite en otros conjuntos y además tiene el efecto de reducir la cantidad de generaciones necesarias para que el método converja.

Tabla 4.5: Ejemplos de distintos porcentajes de clasificación y sus respectivos valores de la función de *fitness*, obtenidos utilizando función de activación sigmoide logística en el conjunto WBC. Se debe recordar que el *fitness* corresponde a la función de costos de Cauchy-Schwartz, que debe ser minimizada, es decir un menor *fitness* corresponde a un mejor resultado para PG.

| Clasificación (%) | Valor de <i>fitness</i> (J_{CS}) |
|-------------------|--------------------------------------|
| 96,92 | 0,5315 |
| 96,77 | 0,5314 |
| 97,21 | 0,5314 |
| 38,27 | 0,5732 |
| 74,49 | 0,5714 |
| 38,42 | 0,5730 |

Wine

En este conjunto, los resultados de todos los métodos son muy similares. Dadas las desviaciones estándar obtenidas, no es posible decir que alguno sea superior con significancia estadística. Cabe notar que en estos experimentos los datos se encuentran normalizados, ya que esta base presenta rangos de valores muy distintos para cada dimensión. A pesar de que el método de Jenssen presenta resultados similares a los demás métodos, no se pudo reproducir el promedio de 97.2 % reportado en [10].

Pentest3

En este conjunto el desempeño de PG es claramente superior a los otros dos métodos, ya sea usando máximo o sigmoide logística como función de activación. Sólo considerando el mejor resultado, el método de Jenssen se acerca a la clasificación obtenida por PG. Al igual que para el conjunto Wine, no se pudo reproducir el resultado reportado en [10], donde se menciona un porcentaje de clasificación promedio de 84,4 %. Ajustando manualmente el tamaño del kernel inicial a $5\sigma_{\text{opt}}$ se obtiene una clasificación promedio de 82,52 %.

FCPS

Esta base está especialmente diseñada para poner a prueba algoritmos de *clustering*, utilizando configuraciones donde ciertos métodos conocidos suelen fallar. La base tiene la ventaja de que todos los conjuntos son de dos o tres dimensiones y por lo tanto, se pueden visualizar y el resultado de cada

método puede ser analizado gráficamente. De acuerdo a la complejidad de los *clusters* presentes en los conjuntos de datos, esta base se puede separar en tres grupos:

- Los conjuntos EngyTime, Tetra y Hepta son bastante simples, ya que los *clusters* son todos elipsoidales (gaussianos) y linealmente separables (excepto EngyTime, donde existe traslape entre las dos clases, pero igualmente los *clusters* son en gran medida separables). Para estos conjuntos, todos los métodos funcionan igualmente bien, encontrando siempre la partición óptima, a excepción de *k-means*, que falla reiteradamente en el conjunto Hepta. Un ejemplo de clasificación incorrecta obtenida por *k-means* se muestra en la figura 4.2(a), donde se aprecia que el algoritmo une el *cluster* central con uno de los laterales (formando el *cluster* amarillo), mientras que separa otro *cluster* en dos (*clusters* verde y negro).
- Los conjuntos Lsun, TwoDiamonds y WingNut son linealmente separables, pero la forma de los *clusters* es más compleja que en los conjuntos anteriores. En Lsun, sólo uno de los *clusters* es gaussiano, mientras que los otros dos tienen una distribución uniforme en una región rectangular. El mal desempeño de *k-means* en este conjunto es esperable, ya que está diseñado para encontrar *clusters* gaussianos. La figura (4.2)(b) muestra un resultado típico de *k-means* en este conjunto. PG, en cambio, encuentra la partición óptima en la mayoría de los casos, aunque con algunas deficiencias. Con función de activación máximo, en 5 de las 40 pruebas se obtiene un resultado erróneo similar al que se muestra en la figura 4.1(a). Al contrario del caso de WBC, si se analizan los resultados de cada ejecución, se observa que la configuración indeseada corresponde a un mejor valor de *fitness* que la partición ideal. En este caso, entonces, el error se debe a una deficiencia en la función de *fitness*, ya que asigna un puntaje mejor a una solución que está lejos de ser óptima. Al usar sigmoide logística, en 10 de las 40 pruebas se obtiene un resultado similar al de la figura 4.1(b). En este caso, se repite la situación del conjunto WBC, donde el algoritmo converge a un mínimo local, ya que los valores de *fitness* para los 10 resultados erróneos son mayores que para los otros 30. Si se observa con detenimiento ambas figuras (4.1(a) y 4.1(b)), se apreciará que las configuraciones erróneas no son equivalentes. El método de Jenssen tiene un bajo desempeño en este conjunto. El resultado típico corresponde al que se muestra en la figura 4.3(a), que es equivalente al de la figura 4.1(b), obtenido por PG con sigmoide logística. En algunos casos se obtiene el resultado mostrado en la figura 4.3(b), que es equivalente al de la figura 4.1(a), obtenido por

PG con máximo. Es decir, en este conjunto el método de Jenssen converge casi siempre a un óptimo local, en algunas ocasiones encuentra el óptimo global indeseado, pero nunca encuentra el resultado correcto. En el conjunto WingNut, existe una mayor densidad de patrones en los bordes de los *clusters*, lo que hace que todos los métodos analizados se confundan, aunque el método de Jenssen lo hace en menor medida. Un resultado típico de PG para este conjunto se muestra en la figura 4.1(c). En el conjunto TwoDiamonds, ninguno de los métodos presenta problemas. Este conjunto está más bien pensado para evidenciar las deficiencias de métodos jerárquicos como el de enlace simple y todos los algoritmos particionales estudiados acá funcionan correctamente.

- Por último, los conjuntos ChainLink y Atom son los más complejos, ya que no tienen una distribución gaussiana ni son linealmente separables. Debido a esto, en ambos conjuntos *k-means* no es capaz de separar correctamente los *clusters*, ya que sólo puede formar fronteras lineales. Ejemplos de resultados de *k-means* para ChainLink y Atom se muestran en las figuras 4.2(c) y 4.2(d), respectivamente. PG siempre separa correctamente los dos anillos si se utiliza la función de activación máximo. Al utilizar sigmoide logística, en cambio, el método converge a un mínimo local como el que se muestra en la figura 4.1(d). Sólo en una de las 40 pruebas se obtuvo la partición correcta, que corresponde a un mejor valor de *fitness* que las otras 39 ejecuciones. Algo similar sucede con el método de Jenssen. Sólo en una pocas de las 40 ejecuciones se obtiene un resultado cercano al óptimo (alrededor de 96%). En las figuras 4.3(c) y 4.3(d) se muestran resultados típicos del método de Jenssen en los conjuntos Atom y ChainLink, respectivamente.

4.2. Generalización

En la tabla 4.6 se muestran los resultados para el experimento de generalización. Dado que los demás métodos no contemplan de forma intrínseca la capacidad de generalización, sólo se incluyen los resultados para PG. La función de activación utilizada en este experimento es máximo, dado que con ella se obtienen los mejores resultados en general para el experimento de *clustering*, si se consideran todos los conjuntos de datos. Además, dado que se deben ejecutar 60 evoluciones de PG para cada base, se incluyen las dos técnicas de reducción de complejidad descritas en la sección

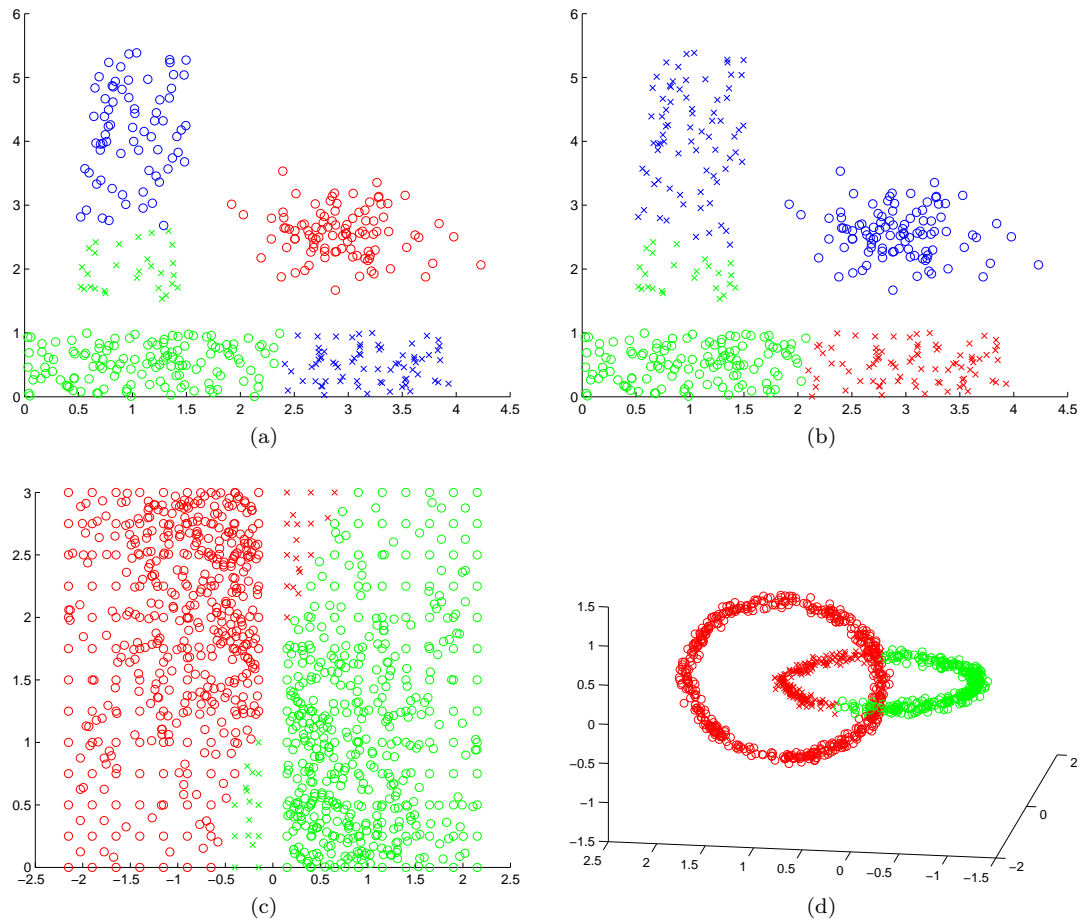


Figura 4.1: Ejemplos de clasificación subóptima obtenida por PG: (a) Lsun con función de activación máximo; (b) Lsun con función de activación sigmoide logística; (c) WingNut con función de activación máximo; (d) ChainLink con función de activación sigmoide logística. Los colores identifican la clase asignada. Un círculo señala una clasificación correcta y una cruz, incorrecta

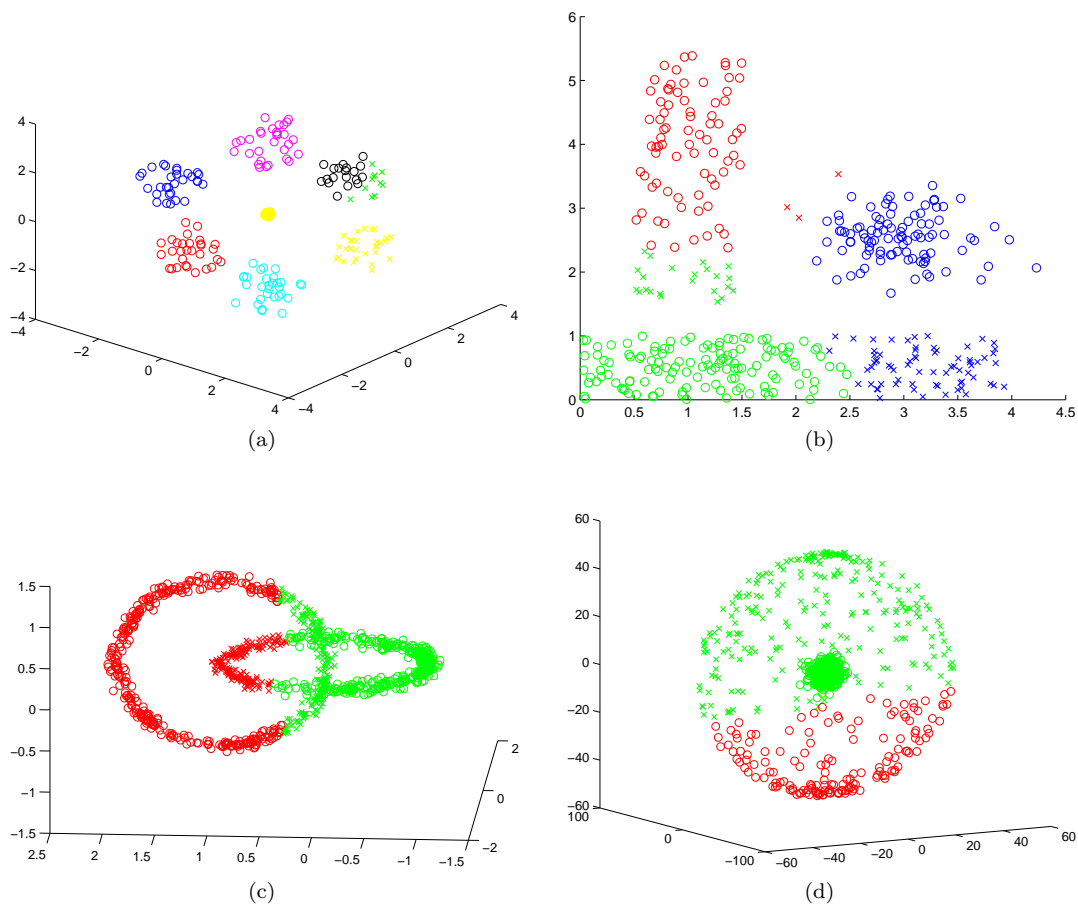


Figura 4.2: Ejemplo de clasificación subóptima obtenida por k -means: (a) Hepta; (b) Lsun; (c) ChainLink; (d) Atom

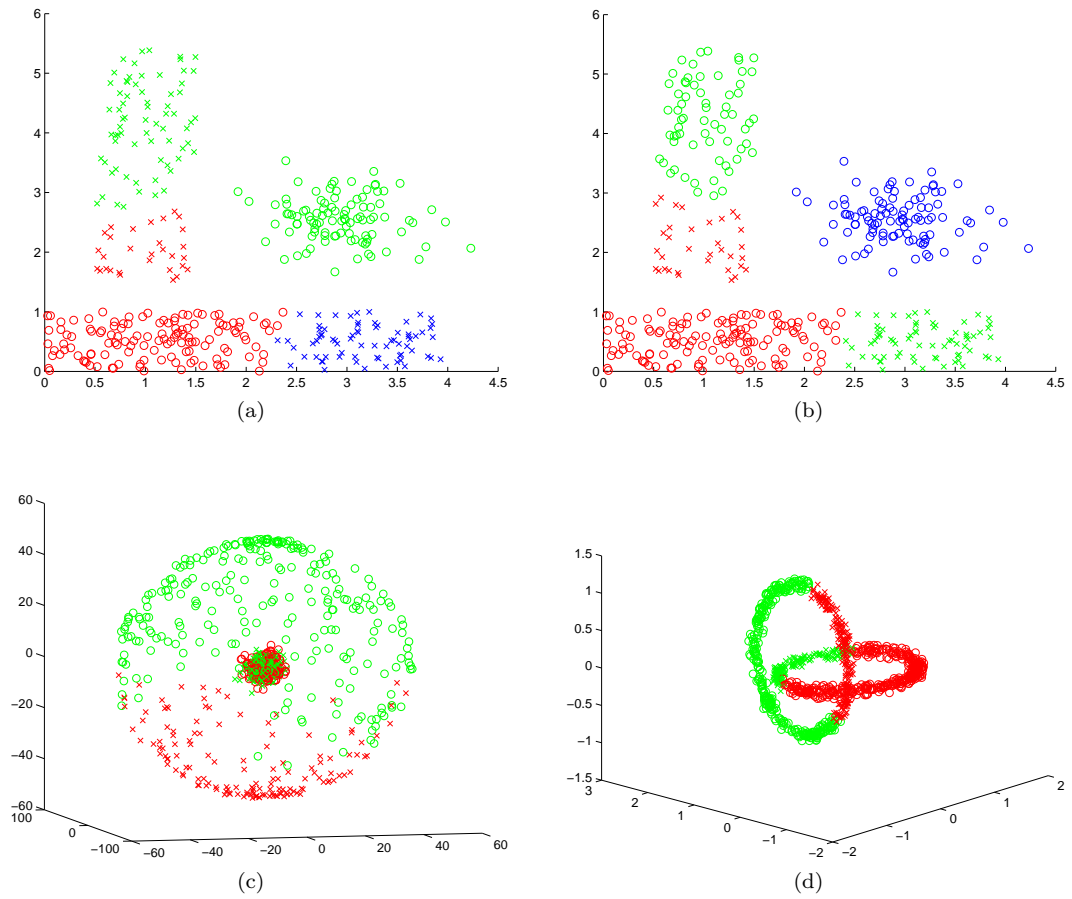


Figura 4.3: Ejemplos de clasificación subóptima obtenida por el método de Jenssen: (a) resultado típico para conjunto Lsun; (b) óptimo indeseado para conjunto Lsun; (c) resultado típico para conjunto Atom; (d) resultado típico para conjunto ChainLink

Tabla 4.6: Resultados experimento de generalización

| Conjunto de datos | Clasificación entrenamiento (%) | | Clasificación validación (%) | |
|-------------------|---------------------------------|------------|------------------------------|------------|
| | Promedio | Desviación | Promedio | Desviación |
| Iris | 90,81 | 2,33 | 88,44 | 7,85 |
| WBC | 96,64 | 0,36 | 96,65 | 2,32 |
| Wine | 95,33 | 1,53 | 90,52 | 7,21 |
| Pentest3 | 83,27 | 1,79 | 83,41 | 4,27 |
| EngyTime | 96,04 | 0,19 | 96,01 | 1,11 |
| Tetra | 100 | 0 | 99,42 | 1,41 |
| Hepta | 100 | 0 | 99,92 | 0,61 |
| Lsun | 92,41 | 11,41 | 92,75 | 11,72 |
| TwoDiamonds | 100 | 0 | 99,92 | 0,31 |
| WingNut | 96,89 | 0,3 | 96,96 | 1,44 |
| Chainlink | 99,43 | 2,98 | 98,97 | 3,26 |
| Atom | 100 | 0 | 99,81 | 0,56 |

3.8, las cuales no afectan significativamente los resultados, pero sí aceleran los cálculos, según se muestra más adelante. Los resultados de la tabla 4.6 muestran que el método propuesto obtiene porcentajes de clasificación similares para los conjuntos de entrenamiento y de validación. Como es de esperar, existe un descenso en el desempeño al evaluar en el conjunto de validación, pero en la mayoría de las bases éste no es significativo. A través de estos resultados se puede concluir que el método presenta una buena capacidad de generalización.

4.3. Aprendizaje de la densidad de probabilidad condicional

En la figura 4.4 se muestra la estimación por ventana de Parzen de $p(\mathbf{x})$ para todos los puntos de la grilla. Se obtienen cuatro distribuciones gaussianas centradas en los centros reales de cada *cluster*, lo que sugiere que el método de Parzen logra estimar correctamente la pdf para el conjunto completo de datos.

La figura 4.5 muestra el resultado típico de ejecutar PG en el conjunto de datos utilizado para este experimento (figura 3.3). Cada una de las cuatro subfiguras muestra la salida normalizada

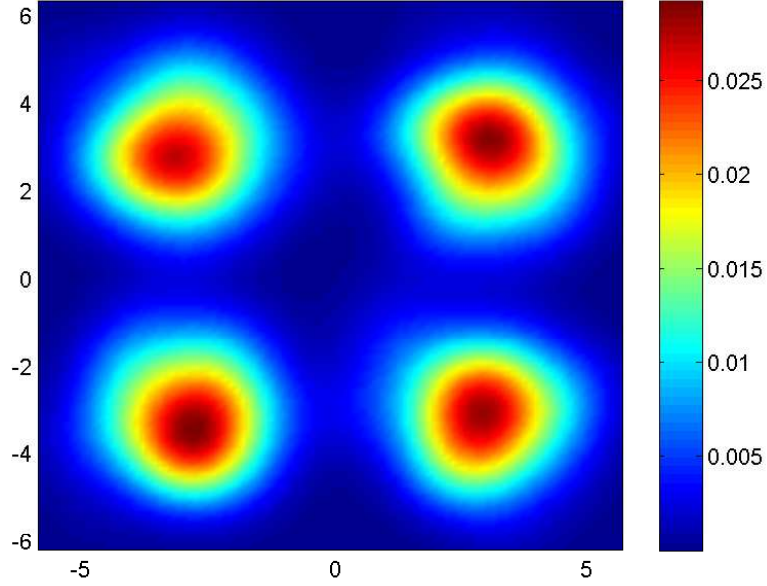


Figura 4.4: Estimación de $p(\mathbf{x})$ mediante ventana de Parzen

de uno de los árboles del individuo ganador. Estas salidas se interpretan como la probabilidad a posteriori de pertenencia a la clase que cada árbol representa, es decir, $P(C_k|\mathbf{x})$. Se aprecia que el método separa apropiadamente el espacio de entrada en cuatro regiones, con un *cluster* en cada una. Finalmente, la figura 4.6 muestra el resultado de combinar la salidas normalizadas con la estimación de $p(\mathbf{x})$ a través del teorema de Bayes, mostrado en (3.7). Se observa que el método es capaz de separar las cuatro distribuciones gaussianas, logrando una estimación de las funciones de densidad de probabilidad condicional que claramente se asemeja a la distribución de los datos para cada clase.

4.4. Reducción de complejidad

En esta sección se muestran los resultados de los experimentos de reducción de complejidad. En la tablas 4.8 y 4.9 se presentan los casos correspondientes a la estimación por muestreo estocástico y por vecindad, respectivamente. Para la tabla 4.10 se utilizaron ambas técnicas simultáneamente. Las tablas muestran el desempeño en términos de clasificación, pero se agrega también la medición del tiempo que demora el método en evaluar una generación (1000 individuos). Para facilitar las

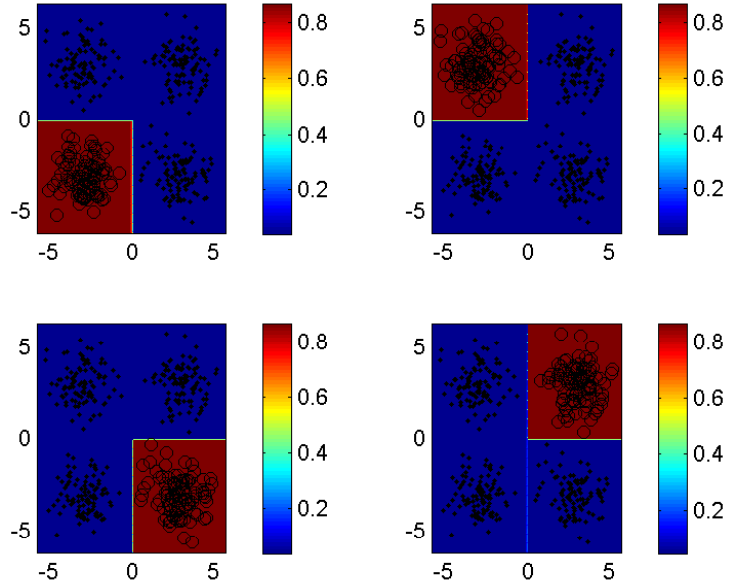


Figura 4.5: $P(C_k|\mathbf{x})$ dada por la salida normalizada de los árboles

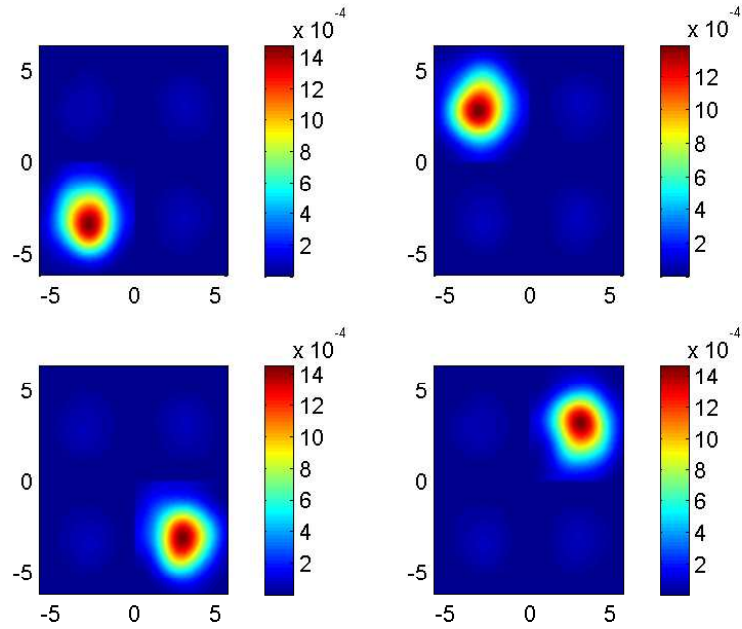


Figura 4.6: Estimación de $p(\mathbf{x}|C_k)$ obtenida usando el teorema de Bayes

Tabla 4.7: Resultados experimento de *clustering* usando PG sin técnicas de reducción de complejidad. Por ser esta la prueba base, todos los tiempos relativos son 1

| Conjunto de datos | Clasificación (%) | | | Tiempo relativo |
|-------------------|-------------------|------------|-------|-----------------|
| | Promedio | Desviación | Mejor | |
| Iris | 90,07 | 0,20 | 90,67 | 1 |
| WBC | 96,66 | 0,14 | 97,07 | 1 |
| Wine | 94,45 | 1,21 | 97,75 | 1 |
| Pentest3 | 82,09 | 2,11 | 85,52 | 1 |
| EngyTime | 96 | 0,02 | 96,04 | 1 |
| Tetra | 100 | 0 | 100 | 1 |
| Hepta | 100 | 0 | 100 | 1 |
| Lsun | 96,69 | 8,26 | 100 | 1 |
| TwoDiamonds | 100 | 0 | 100 | 1 |
| WingNut | 96,88 | 0,22 | 98,23 | 1 |
| Chainlink | 99,97 | 0,09 | 100 | 1 |
| Atom | 100 | 0 | 100 | 1 |

comparaciones, la tabla 4.7 repite los resultados de la tabla 4.1 (es decir, experimento de *clustering* con PG y función de activación máximo, pero sin usar técnicas de reducción de complejidad). Esta prueba se consideró como la prueba “base”. Todos los tiempos presentados en esta sección se muestran de forma relativa a los tiempos medidos en la prueba base. Esto significa que si una prueba muestra un tiempo de 0,5 para cierto conjunto, entonces dicha prueba demoró la mitad del tiempo que tomó la prueba base en el mismo conjunto. Con esto se logra independizar las medidas de las características del computador donde se efectuaron las pruebas (todas se ejecutaron en el mismo computador). Para todas las pruebas, se escogió la función de activación máximo, ya que es la que presenta mejores resultados en los experimentos de *clustering*, si se consideran todos los conjuntos de datos.

Tabla 4.8: Resultados experimento de *clustering* usando PG y muestreo estocástico

| Conjunto de datos | Clasificación (%) | | | Tiempo relativo |
|-------------------|-------------------|------------|-------|-----------------|
| | Promedio | Desviación | Mejor | |
| Iris | 90,15 | 0,38 | 91,33 | 0,32 |
| WBC | 96,98 | 0,2 | 97,21 | 0,22 |
| Wine | 95,67 | 1,27 | 97,75 | 0,29 |
| Pentest3 | 82,74 | 1,79 | 85,33 | 0,2 |
| EngyTime | 96,08 | 0,07 | 96,19 | 0,2 |
| Tetra | 100 | 0 | 100 | 0,23 |
| Hepta | 100 | 0 | 100 | 0,26 |
| Lsun | 96,7 | 8,39 | 100 | 0,22 |
| TwoDiamonds | 100 | 0 | 100 | 0,2 |
| WingNut | 96,86 | 0,17 | 97,44 | 0,21 |
| Chainlink | 99,95 | 0,16 | 100 | 0,21 |
| Atom | 100 | 0 | 100 | 0,22 |

 Tabla 4.9: Resultados experimento de *clustering* usando PG y estimación por vecindad

| Conjunto de datos | Clasificación (%) | | | Tiempo relativo |
|-------------------|-------------------|------------|-------|-----------------|
| | Promedio | Desviación | Mejor | |
| Iris | 90,7 | 0,3 | 91,33 | 0,51 |
| WBC | 96,32 | 0,16 | 96,77 | 0,42 |
| Wine | 94,26 | 0,96 | 96,63 | 0,45 |
| Pentest3 | 82,41 | 2,01 | 85,52 | 0,28 |
| EngyTime | 96,06 | 0,05 | 96,12 | 0,25 |
| Tetra | 100 | 0 | 100 | 0,23 |
| Hepta | 100 | 0 | 100 | 0,24 |
| Lsun | 89,64 | 11,84 | 100 | 0,31 |
| TwoDiamonds | 100 | 0 | 100 | 0,38 |
| WingNut | 96,9 | 0,11 | 97,44 | 0,34 |
| Chainlink | 99,97 | 0,09 | 100 | 0,18 |
| Atom | 100 | 0 | 100 | 0,35 |

Tabla 4.10: Resultados experimento de *clustering* usando PG, muestreo estocástico y vecindad

| Conjunto de datos | Clasificación (%) | | | Tiempo relativo |
|-------------------|-------------------|------------|-------|-----------------|
| | Promedio | Desviación | Mejor | |
| Iris | 90,18 | 0,43 | 91,33 | 0,23 |
| WBC | 95,99 | 3,8 | 96,92 | 0,1 |
| Wine | 92,68 | 2,11 | 96,07 | 0,2 |
| Pentest3 | 82,41 | 2,26 | 85,43 | 0,07 |
| EngyTime | 96,09 | 0,09 | 96,31 | 0,05 |
| Tetra | 100 | 0 | 100 | 0,08 |
| Hepta | 100 | 0 | 100 | 0,11 |
| Lsun | 96,06 | 9,03 | 100 | 0,1 |
| TwoDiamonds | 100 | 0 | 100 | 0,08 |
| WingNut | 96,95 | 0,26 | 98,43 | 0,08 |
| Chainlink | 99,96 | 0,1 | 100 | 0,05 |
| Atom | 100 | 0 | 100 | 0,08 |

Los resultados de la tabla 4.8 muestran que el tiempo necesario para evaluar mil individuos disminuye hasta alrededor de un 20 %-30 % del tiempo base si se utiliza muestreo estocástico. En los conjuntos con más patrones (EngyTime y Pentest3) se cumple el mínimo esperado de un 20 % (que es exactamente la cantidad de patrones considerados para el cálculo). Por otro lado, el porcentaje de clasificación prácticamente no se ve afectado en ninguno de los conjuntos de datos.

Al utilizar la vecindad en torno a cada patrón para estimar la densidad (tabla 4.9), los tiempos de cálculo se pueden reducir desde la mitad (conjunto Iris) hasta un 18 % del tiempo base (conjunto ChainLink). Igual que en el caso de muestreo estocástico, la clasificación se ve afectada muy levemente, excepto en el conjunto Lsun. En este conjunto, se aprecia una baja considerable del desempeño (de 96,69 % a 89,64 %) y se debe a que aumenta el número de casos en que el método converge a una solución errónea.

Finalmente, al utilizar ambas técnicas en forma conjunta (tabla 4.10) se obtienen prácticamente los mismos resultados que en la prueba base, sin la baja de desempeño observada previamente en el conjunto Lsun. El tiempo necesario para evaluar los individuos disminuye hasta tomar sólo un 5 % de la prueba base.

Capítulo 5

Análisis y discusiones

5.1. Desempeño del método propuesto

De acuerdo a los resultados presentados en la sección 4.1, el método propuesto es capaz de separar correctamente los *clusters* en todos los conjuntos de datos en los que fue probado, especialmente cuando se utiliza la función de activación máximo. El método fue ejecutado utilizando conjuntos de datos reales de 4 a 16 dimensiones y artificiales de 2 y 3 dimensiones. Los conjuntos reales no son visualizables, por su alta dimensionalidad, por lo que no se puede analizar su complejidad gráficamente. Sin embargo, considerando el desempeño aceptable del algoritmo *k-means* en estos conjuntos, se puede argumentar que todos ellos poseen una estructura relativamente simple, con *clusters* de tipo hipersféricos. En todos estos conjuntos, PG logra siempre un desempeño igual o superior a *k-means*. Al compararlo con el método de Jenssen, PG también alcanza una mejor clasificación en estos conjuntos. Sólo haciendo un ajuste manual de los parámetros del método de Jenssen, como el tamaño inicial del kernel y la cantidad de iteraciones de annealing se logra un desempeño similar al de PG.

Los conjuntos de datos artificiales, provenientes de la base FCPS, fueron diseñados para simular varias situaciones en que distintos algoritmos de *clustering* suelen fallar. En todos los conjuntos de esta base, el método propuesto logra separar correctamente los *clusters* si se utiliza la función de activación máximo. De especial interés son los conjuntos ChainLink y Atom, con estructuras complejas y no linealmente separables. En ambos casos, tanto *k-means* como el método de Jenssen

fallan consistentemente, mientras PG no presenta problemas para separar correctamente los dos *clusters*.

El método propuesto considera dos posibles funciones de activación: máximo y sigmoide logística. La elección de esta función determina cuán abrupta es la transformación de la salida de los árboles en una probabilidad de pertenencia a un *cluster*. Al utilizar la función de activación sigmoide logística, PG presenta problemas de convergencia que le impiden obtener un resultado óptimo de forma consistente en algunos conjuntos (WBC, Lsun, y ChainLink). Estos problemas se solucionan en su totalidad al utilizar la función de activación máximo. Una explicación posible para este resultado tienen que ver con el tamaño del espacio de búsqueda de PG, que corresponde a todos los posibles programas que se pueden generar con el conjunto de terminales y funciones base disponibles. Al usar el máximo de los valores de probabilidad para cada *cluster*, aquellos programas que asignen los patrones a los mismos *clusters*, aunque los valores exactos de probabilidad sean distintos, generan la misma solución y pasan a ser equivalentes en términos de *fitness*. Por lo tanto, sólo importa la partición generada por el programa y no los valores exactos de las funciones de pertenencia. De esta manera, se puede decir que el espacio de búsqueda se reduce, ya que muchos programas son considerados iguales, aunque su estructura sea distinta. Otra forma de verlo es que aumenta la probabilidad de encontrar un individuo óptimo entre todos los individuos posibles a formar. Por lo mismo, en todos los casos, la función de activación máximo reduce considerablemente el número de iteraciones necesarias para encontrar una solución al problema.

La representación utilizada para los individuos (programas que representan funciones de pertenencia) permite explorar el espacio de posibles particiones de manera eficiente. Dado que los programas corresponden a funciones aplicadas directamente sobre los datos, todas las soluciones tendrán forzosamente cierta estructura, es decir, formarán *clusters* con regiones definidas y no pueden separar los datos de manera totalmente aleatoria. En cambio, al usar una tira binaria como los métodos basados en algoritmos genéticos descritos en 2.6.1, es totalmente posible (y de hecho, lo más probable) que las soluciones iniciales separen los datos de manera aleatoria, sin ninguna estructura definida. En definitiva, esto implica que el número de posibles particiones generadas por GP es menor que una implementación con GA, lo que también redundará en un espacio de búsqueda más pequeño. Además, recordando que se quiere minimizar la entropía de la partición generada, y ésta será menor de acuerdo al nivel de estructura que tengan los datos en cada *cluster*, se puede

decir que las soluciones en PG comienzan con un nivel de entropía mucho menor, ya que el nivel de aleatoriedad de las particiones es menor. Es de esperar que un algoritmo genético encuentre eventualmente las mismas soluciones que PG, pero tomando un tiempo mucho mayor en converger.

Comparado con el método de Jenssen, PG presenta un desempeño superior. Considerando que ambos métodos utilizan la misma función de costos a minimizar, se puede concluir que el método propuesto en su conjunto (uso de una técnica evolutiva como PG, estructura de los individuos, interpretación probabilística, etc.) ayuda a superar los problemas de convergencia que presenta el método de Jenssen, derivados de su enfoque basado en descenso por gradiente.

5.2. Generalización

Una gran ventaja del método propuesto es su capacidad inherente de generalización. Al usar PG, las soluciones obtenidas corresponden a programas, que pueden ser evaluados usando datos no vistos con anterioridad. Así, se obtienen no sólo una partición del conjunto de datos estudiados, sino un programa clasificador de nuevos datos. En la sección 4.2 se muestra que el método propuesto es capaz de mantener un buen desempeño al clasificar nuevos patrones no vistos durante la etapa de entrenamiento. Esto permite utilizar el método de *clustering* propuesto para la tarea de predicción basada en grupos, descrita en la sección 2.2. Ninguno de los otros métodos estudiados en esta tesis (y en general, casi ningún algoritmo de *clustering*) considera intrínsecamente esta capacidad de generalización.

Otra característica importante del método es su capacidad de estimar la función de densidad de probabilidad condicional de cada clase de los datos. Gracias a su capacidad de generalización y a la interpretación probabilística, el programa resultante de una ejecución del método puede evaluarse en todo el espacio que rodea a los datos de entrenamiento. La salida del programa se puede transformar luego en la probabilidad condicional de cada clase por separado, en cualquier punto del espacio que se desee. Los resultados mostrados en la sección 4.3 indican que al menos para conjuntos simples la estimación se asemeja a las distribuciones reales.

5.3. Función de *fitness*

Para cualquier algoritmo evolutivo, la función de *fitness*, junto con la representación de las soluciones, son los elementos más críticos de especificar correctamente si se quiere resolver cualquier problema de mediana complejidad. Para una aplicación de *clustering*, esto es quizás más cierto que en otros problemas, ya que se trata de un problema no supervisado y ni siquiera existe una definición clara del concepto de *cluster*. La función objetivo definirá el tipo de *clusters* que se pueden obtener.

En esta tesis se describe por qué no es efectivo utilizar un *fitness* basado en centroides o en estadísticos de segundo orden de los datos: esto llevaría a encontrar *clusters* solo de tipo hiperesféricos. Para encontrar agrupaciones de datos con formas más complejas, es necesario utilizar herramientas más avanzadas. La teoría de la información entrega herramientas que permiten capturar el nivel de estructura presente en particiones de los datos, independiente de su forma. La función objetivo de Cauchy-Schwartz propuesta por Jenssen y basada en los desarrollos de Príncipe et al. permite estimar la entropía de los *clusters* generados de forma simple, utilizando directamente los datos disponibles. De acuerdo a los resultados obtenidos en esta tesis, se puede concluir que esta medida es apropiada para medir la calidad de una partición en conjuntos de datos con estructuras muy disímiles, desde las más simples a las más complejas y en varias dimensiones.

En esta tesis se presentó un desarrollo teórico que permite fundamentar la elección de la función de *fitness*. Aún así, pueden quedar dudas sobre la pérdida de generalidad que implica usar un kernel isotrópico (con matriz de covarianza diagonal y el mismo σ para todas las dimensiones) en la estimación de la densidad de probabilidad, lo que simplifica el desarrollo de la función objetivo. Los experimentos realizados en esta tesis muestran que esta simplificación no afecta la capacidad de la función de *fitness* de diferenciar entre una “buena” y una “mala” partición. Más aún, el uso de la regla de Silvermann para la selección del tamaño del kernel, aunque corresponda a una estimación gruesa del tamaño óptimo para calzar los datos con una sola distribución gaussiana, es suficiente para obtener buenos resultados con el método propuesto. A diferencia del método de Jenssen, PG no presenta una sensibilidad importante con respecto al valor de σ , lo que permite ejecutar el método en todos los conjuntos sin necesidad de ajuste. Además, el método de Jenssen necesita un esquema de annealing, que consiste en disminuir gradualmente el valor de σ para hacer un ajuste cada vez más fino de la solución, lo que a su vez requiere ajustar más parámetros. Con PG, el valor de σ se mantiene constante durante toda la evolución.

5.4. Costo computacional

A pesar de las ventajas mencionadas hasta ahora de PG con respecto al método de Jenssen, una gran desventaja es su alto costo computacional. Al basarse en descenso por gradiente, el método de Jenssen considera sólo una solución de forma simultánea, la cual se va modificando en el tiempo. Así, una iteración requiere evaluar sólo una vez la función de *fitness* (en realidad se evalúa el gradiente, pero el costo es similar, $O(n^2)$). En PG, en cambio y como en cualquier método evolutivo, se consideran simultáneamente muchas soluciones. En una iteración (generación) se evalúa a toda la población de individuos, calculando para cada uno la función de *fitness*. Es por esto que ambos métodos deben compararse con respecto a la cantidad de evaluaciones de la función de *fitness* y no de iteraciones del método.

Las técnicas para reducir el costo computacional propuestas en la sección 3.8 logran disminuir considerablemente el tiempo de ejecución del método, sin afectar mayormente el resultado. Sin embargo, PG sigue siendo más costoso que el método de Jenssen (siendo ambos mucho más costosos que *k-means*). Le corresponde al usuario decidir si las ventajas que ofrece el método propuesto superan a las desventajas de su alto costo computacional.

5.5. Complejidad

En esta tesis se ha privilegiado el desempeño del método por sobre el tamaño de los programas resultantes. Como consecuencia, normalmente se obtienen programas muy grandes que son imposibles de analizar manualmente. Esto impide materializar una de las posibles ventajas de utilizar PG para *clustering*, que consiste en la posibilidad de obtener mayor información acerca de los datos a través del análisis de las reglas presentes en los programas generados por el proceso evolutivo. Para trabajos futuros, podrían considerarse algunas de las técnicas mencionadas en la sección 2.4.4.

5.6. Número de *clusters*

Al igual que la mayoría de los algoritmos de *clustering* encontrados en la literatura, el método propuesto necesita que el usuario ingrese el número “correcto” de *clusters* existentes en los datos previo a la ejecución. Esta no es una limitación menor, ya que las técnicas de *clustering* suelen usarse como un método de análisis preliminar de los datos, es decir, para obtener información

acerca de los datos cuando no se conoce nada de ellos. En este caso, no es posible asumir que el usuario conoce el número correcto de *clusters a priori*. A pesar de que existen técnicas para estimar este número, como se describe en la sección 2.2.2, éstas están basadas en su mayoría en aplicar métodos de *clustering*, lo que hace recordar el problema de la definición circular mencionado en la introducción: se ocupa un método de *clustering* para estimar el número de *clusters*, para así poder aplicar un método de *clustering*. Una mejora importante para el método propuesto sería eliminar el requisito de conocer el número de *clusters* de antemano.

Capítulo 6

Conclusiones

En esta tesis se ha desarrollado un nuevo método de *clustering* basado en programación genética y teoría de la información. El método comprende una representación multiárbol de los individuos de PG, una interpretación probabilística para la salida de los árboles y una función de *fitness* basada en teoría de la información que mide la entropía de la partición generada. En su conjunto, estas características permiten al método propuesto encontrar *clusters* sin restricción de forma, en conjuntos multidimensionales y sin asumir un modelo *a priori* de los datos.

Visto desde el ámbito de la computación evolutiva, el método propuesto es el primero en la literatura que logra separar cualquier tipo de *clusters*, no sólo hiperesféricos, ya que la función de *fitness* está basada en la entropía de los datos de cada *cluster*, en vez de estadísticos de segundo orden. Desde el punto de vista del *clustering* con teoría de la información, esta tesis propone un método de optimización evolutivo, más apropiado que el descenso por gradiente y otros métodos clásicos encontrados en la literatura. De esta forma se logra evitar considerablemente mínimos locales y así sacar real partido a los poderosos conceptos de teoría de la información.

Otra propiedad importante a destacar es la capacidad de generalización del método. En general los métodos de *clustering* se preocupan de encontrar una partición adecuada de los datos, pero dejan de lado la posibilidad de generalizar el conocimiento adquirido para clasificar nuevos patrones. El método propuesto considera intrínsecamente esta posibilidad, ya que como resultado entrega programas clasificadores, que pueden ser evaluados utilizando datos nuevos, manteniendo un desempeño similar a la fase de entrenamiento.

6.1. Recomendaciones para trabajo futuro

El mayor desafío pendiente para mejorar el método propuesto es resolver el problema del número de *clusters*. Si se logra diseñar un esquema que permita obtener a la vez la mejor partición y el número “correcto” de *clusters*, se mejorará enormemente la aplicabilidad del método, especialmente en problemas reales sobre los cuales se posee casi nula información previa acerca de los datos

Otra tarea importante es continuar reduciendo el costo computacional. A pesar de las ventajas que presenta, el método propuesto es altamante costoso, limitando su aplicabilidad en conjuntos de datos masivos, los cuales abundan en la actualidad. Cualquier mejora en este ámbito es bienvenida, especialmente si al mismo tiempo se mejora la interpretabilidad de los árboles resultantes, permitiendo su análisis para tareas de data mining. Una variable de PG no explorada en esta tesis, pero que afecta la velocidad, son las condiciones de término. Sería de utilidad contar con un método de detención temprana de la evolución, detectando automáticamente cuando se está cerca de la solución óptima, en vez de usar un número fijo de generaciones.

Por último, sería de interés evaluar el método propuesto en conjuntos de datos reales de mayor complejidad (por ejemplo, de muy alta dimensionalidad) y además compararlo con métodos de *clustering* basados en otras técnicas evolutivas, como algortimos genéticos y PSO.

Bibliografía

- [1] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press, 2002.
- [2] C. M. Bishop, *Neural networks for pattern recognition*. Oxford, UK: Oxford University Press, 1996.
- [3] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. San Diego, CA, USA: Academic Press, 1999.
- [4] A. D. Gordon, *Classification*, 2nd ed., ser. Monographs in Applied Statistics and Probability. Chapman & Hall / CRC, 1999.
- [5] A. Jain, R. Duin, and J. Mao, “Statistical pattern recognition: a review,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, Jan. 2000.
- [6] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.
- [7] E. Gokcay and J. Principe, “Information theoretic clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 158–171, Feb. 2002.
- [8] H. Li, K. Zhang, and T. Jiang, “Minimum entropy clustering and applications to gene expression analysis,” in *Proc. IEEE Computational Systems Bioinformatics Conference CSB 2004*, 16–19 Aug. 2004, pp. 142–151.
- [9] R. Jenssen, D. Erdogmus, K. E. Hild II, J. C. Príncipe, and T. Eltoft, “Optimizing the Cauchy-Schwarz PDF distance for information theoretic, non-parametric clustering,” in *EMMCVPR*, 2005, pp. 34–45.

- [10] —, “Information cut for clustering using a gradient descent approach,” *Pattern Recognition*, vol. 40, no. 3, pp. 796–806, March 2007.
- [11] C. E. Shannon, “A mathematical theory of communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, July, October 1948.
- [12] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed. New York, NY, USA: John Wiley & Sons, 2006.
- [13] E. Gokcay and J. Principe, “A new clustering evaluation function using Renyi’s information potential,” in *Acoustics, Speech, and Signal Processing, 2000. ICASSP ’00. Proceedings. 2000 IEEE International Conference on*, vol. 6, 5–9 June 2000, pp. 3490–3493.
- [14] J. C. Príncipe, D. Xu, and J. Fisher, “Information theoretic learning,” in *Unsupervised Adaptive Filtering*, S. Haykin, Ed. New York: John Wiley & Sons, 2000.
- [15] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.
- [16] J. Koza, I. Bennett, F.H., D. Andre, and M. Keane, “Four problems for which a computer program evolved by genetic programming is competitive with human performance,” in *Proc. IEEE International Conference on Evolutionary Computation*, 20–22 May 1996, pp. 1–10.
- [17] J. Koza, I. Bennett, F.H., J. Lohn, F. Dunlap, M. Keane, and D. Andre, “Automated synthesis of computational circuits using genetic programming,” in *Proc. IEEE International Conference on Evolutionary Computation*, 13–16 April 1997, pp. 447–452.
- [18] J. Koza, “Darwinian invention and problem solving by means of genetic programming,” in *Proc. IEEE International Conference on Systems, Man, and Cybernetics IEEE SMC ’99*, vol. 3, 12–15 Oct. 1999, pp. 604–609.
- [19] J. Koza, M. Keane, and M. Streeter, “What’s AI done for me lately? Genetic programming’s human-competitive results,” *IEEE Intelligent Systems*, vol. 18, no. 3, pp. 25–31, May–Jun 2003.
- [20] M. Streeter, M. Keane, and J. Koza, “Automatic synthesis using genetic programming of both

- the topology and sizing for five post-2000 patented analog and mixed analog-digital circuits,” in *Proc. Southwest Symposium on Mixed-Signal Design*, 23–25 Feb. 2003, pp. 5–10.
- [21] J. Kishore, L. Patnaik, V. Mani, and V. Agrawal, “Application of genetic programming for multiclass pattern classification,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 242–258, Sept. 2000.
 - [22] M. Zhang and V. Ciesielski, “Genetic programming for multiple class object detection,” in *AI ’99: Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*. London, UK: Springer-Verlag, 1999, pp. 180–192.
 - [23] T. Loveard and V. Ciesielski, “Representing classification problems in genetic programming,” in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2, 27–30 May 2001, pp. 1070–1077.
 - [24] M. Zhang, V. B. Ciesielski, and P. Andreae, “A domain-independent window approach to multiclass object detection using genetic programming,” *EURASIP Journal on Applied Signal Processing*, vol. 2003, no. 8, pp. 841–859, 2003.
 - [25] W. Smart and M. Zhang, “Classification strategies for image classification in genetic programming,” in *Proceeding of Image and Vision Computing Conference*, D. Bailey, Ed., Palmerston North, New Zealand, 2003, pp. 402–407.
 - [26] M. Zhang and W. D. Smart, “Multiclass object classification using genetic programming,” in *EvoWorkshops*, ser. Lecture Notes in Computer Science, G. R. Raidl, S. Cagnoni, J. Branke, D. Corne, R. Drechsler, Y. Jin, C. G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. D. Smith, and G. Squillero, Eds., vol. 3005. Springer, 2004, pp. 369–378.
 - [27] M. Zhang and W. Smart, “Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification,” *Pattern Recogn. Lett.*, vol. 27, no. 11, pp. 1266–1274, 2006.
 - [28] D. Muni, N. Pal, and J. Das, “A novel approach to design classifiers using genetic programming,” *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 183–196, April 2004.

- [29] —, “Genetic programming for simultaneous feature selection and classifier design,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 36, no. 1, pp. 106–117, Feb. 2006.
- [30] L. Cordelia, C. De Stefano, F. Fontanella, and A. Marcelli, “Genetic programming for generating prototypes in classification problems,” in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 2, 2-5 Sept. 2005, pp. 1149–1155.
- [31] Y. Lin and B. Bhanu, “Evolutionary feature synthesis for object recognition,” *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, vol. 35, no. 2, pp. 156–171, May 2005.
- [32] X. Tan, B. Bhanu, and Y. Lin, “Fingerprint classification based on learned features,” *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, vol. 35, no. 3, pp. 287–300, Aug. 2005.
- [33] H. Guo, L. Jack, and A. Nandi, “Feature generation using genetic programming with application to fault classification,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 35, no. 1, pp. 89–99, Feb. 2005.
- [34] L. Zhang, L. Jack, and A. Nandi, “Extending genetic programming for multi-class classification by combining k-nearest neighbor,” in *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, vol. 5, 18-23 March 2005, pp. 349–352.
- [35] H. Guo and A. K. Nandi, “Breast cancer diagnosis using genetic programming generated feature,” *Pattern Recognition*, vol. 39, no. 5, pp. 980–987, May 2006.
- [36] H. Firpi, E. Goodman, and J. Echaz, “Genetic programming artificial features with applications to epileptic seizure prediction,” in *Proc. 27th Annual International Conference of the Engineering in Medicine and Biology Society IEEE-EMBS 2005*, 01–04 Sept. 2005, pp. 4510–4513.
- [37] I. De Falco, E. Tarantino, A. Della Cioppa, and F. Gagliardi, “A novel grammar-based genetic programming approach to clustering,” in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. New York, NY, USA: ACM Press, 2005, pp. 928–932.

- [38] N. Boric and P. A. Estevez, "Genetic programming-based clustering using an information theoretic fitness measure," in *Proc. IEEE Congress on Evolutionary Computation CEC 2007*, 2007, pp. 31–38.
- [39] V. Estivill-Castro, "Why so many clustering algorithms: a position paper," *SIGKDD Explorations Newsletter*, vol. 4, no. 1, pp. 65–75, 2002.
- [40] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.
- [41] J. Kleinberg, "An impossibility theorem for clustering," in *Proc. of the 16th conference on Neural Information Processing Systems*, vol. 15, 2002, pp. 446–453.
- [42] R. Xu and I. Wunsch, D., "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.
- [43] K. Fukunaga, *Introduction to statistical pattern recognition*, 2nd ed. San Diego, CA, USA: Academic Press Professional, Inc., 1990.
- [44] A. R. Webb, *Statistical Pattern Recognition, 2nd Edition*. John Wiley & Sons, October 2002.
- [45] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, 1967, pp. 281–297.
- [46] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer-Verlag, 1995.
- [47] T. Martinetz and K. Schulten, "A neural gas network learns topologies," in *Artificial Neural Networks*, T. Kohonen, K. Makisara, O. Simula, and J. Kangas, Eds. Amsterdam: North-Holland, 1991, pp. 397–402.
- [48] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. 28, no. 1, pp. 84–95, Jan 1980.
- [49] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, pp. 1–38, 1977.
- [50] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1981.

- [51] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," *IEEE Transactions on Fuzzy Systems*, vol. 1, no. 2, pp. 98–110, May 1993.
- [52] N. Pal, K. Pal, J. Keller, and J. Bezdek, "A possibilistic fuzzy c-means clustering algorithm," *IEEE Transactions on Fuzzy Systems*, vol. 13, no. 4, pp. 517–530, Aug. 2005.
- [53] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On clustering validation techniques," *Journal of Intelligent Information Systems*, vol. 17, no. 2, pp. 107–145, 2001.
- [54] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, pp. 622–626, 1971.
- [55] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, pp. 193–218, 1985.
- [56] M. Meila, "Comparing clusterings—an information based distance," *Journal of Multivariate Analysis*, vol. 98, no. 5, pp. 873–895, May 2007.
- [57] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 224–227, 1979.
- [58] J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32–57, 1973.
- [59] J. Bezdek and N. Pal, "Some new indexes of cluster validity," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 28, no. 3, pp. 301–315, June 1998.
- [60] S. Bandyopadhyay and U. Maulik, "Nonparametric genetic clustering: comparison of validity indices," *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, vol. 31, no. 1, pp. 120–125, Feb. 2001.
- [61] M. K. Pakhira, S. Bandyopadhyay, and U. Maulik, "Validity index for crisp and fuzzy clusters," *Pattern Recognition*, vol. 37, no. 3, pp. 487–501, March 2004.
- [62] U. Maulik and S. Bandyopadhyay, "Genetic algorithm-based clustering technique," *Pattern Recognition*, vol. 33, pp. 1455–1465, September 2000.

- [63] W. Sheng, S. Swift, L. Zhang, and X. Liu, “A weighted sum validity function for clustering with a hybrid niching genetic algorithm,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 35, no. 6, pp. 1156–1167, Dec. 2005.
- [64] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd ed. Cambridge, MA, USA: MIT Press, 1992.
- [65] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
- [66] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.
- [67] S. Sette and L. Boullart, “Genetic programming: principles and applications,” *Engineering Applications of Artificial Intelligence*, vol. 14, no. 6, pp. 727–736, December 2001.
- [68] J. Koza, “Genetically breeding populations of computer programs to solve problems in artificial intelligence,” in *Proc. 2nd International IEEE Conference on Tools for Artificial Intelligence*, 6–9 Nov. 1990, pp. 819–827.
- [69] M.-J. Willis, H. Hiden, P. Marenbach, B. McKay, and G. Montague, “Genetic programming: an introduction and survey of applications,” in *Proc. Genetic Algorithms In Engineering Systems: Innovations And Applications Second International Conference On (Conf. Publ. No. 446)*, 2–4 Sept. 1997, pp. 314–319.
- [70] A. McIntyre and M. Heywood, “Toward co-evolutionary training of a multi-class classifier,” in *Proc. IEEE Congress on Evolutionary Computation*, vol. 3, 2–5 Sept. 2005, pp. 2130–2137.
- [71] M. Cavaretta and K. Chellapilla, “Data mining using genetic programming: the implications of parsimony on generalization error,” in *Proc. Congress on Evolutionary Computation CEC 99*, vol. 2, 1999, pp. 1330–1337.
- [72] P. Domingos, “Occam’s two razors: The sharp and the blunt,” in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1998.
- [73] S. Silva and E. Costa, “Comparing tree depth limits and resource-limited gp,” in *Proc. IEEE Congress on Evolutionary Computation*, vol. 1, 2–5 Sept. 2005, pp. 920–927.

- [74] S. Bleuler, M. Brack, L. Thiele, and E. Zitzler, “Multiobjective genetic programming: reducing bloat using SPEA2,” in *Proc. Congress on Evolutionary Computation*, vol. 1, 2001, pp. 536–543.
- [75] D. Parrott, X. Li, and V. Ciesielski, “Multi-objective techniques in genetic programming for evolving classifiers,” in *Proc. IEEE Congress on Evolutionary Computation*, vol. 2, 2–5 Sept. 2005, pp. 1141–1148.
- [76] A. Garcia-Almanza and E. Tsang, “Simplifying decision trees learned by genetic programming,” in *Proc. CEC 2006. Evolutionary Computation IEEE Congress on*, 16–21 July 2006, pp. 2142–2148.
- [77] P. Wong and M. Zhang, “Algebraic simplification of GP programs during evolution,” in *GEC-CO ’06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM Press, 2006, pp. 927–934.
- [78] —, “Effects of program simplification on simple building blocks in genetic programming,” in *Proc. IEEE Congress on Evolutionary Computation CEC 2007*, 2007, pp. 1570–1577.
- [79] L. Liu, H. Cai, M. Ying, and J. Le, “RLGP: An efficient method to avoid code bloat on genetic programming,” in *Proc. International Conference on Mechatronics and Automation ICMA 2007*, 5–8 Aug. 2007, pp. 2945–2950.
- [80] H. Xie, M. Zhang, and P. Andreae, “An analysis of depth of crossover points in tree-based genetic programming,” in *Proc. IEEE Congress on Evolutionary Computation CEC 2007*, 2007, pp. 4561–4568.
- [81] A. A. Freitas, “A genetic programming framework for two data mining tasks: Classification and generalized rule induction,” in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, Eds. Stanford University, CA, USA: Morgan Kaufmann, 1997, pp. 96–101.
- [82] C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas, “Discovering comprehensible classification rules by using genetic programming: a case study in a medical domain,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben,

- M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 2. Orlando, Florida, USA: Morgan Kaufmann, 13-17 Jul. 1999, pp. 953–958.
- [83] I. De Falco, A. Della Cioppa, and E. Tarantino, “Discovering interesting classification rules with genetic programming,” *Applied Soft Computing*, vol. 1, no. 4, pp. 257–269, May 2002.
- [84] K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng, “Mining multiple comprehensible classification rules using genetic programming,” in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, Eds. IEEE Press, 2002, pp. 1302–1307.
- [85] C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz, “A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets,” *Artificial Intelligence in Medicine*, vol. 30, no. 1, pp. 27–48, January 2004.
- [86] A. Tsakonas, G. Dounias, J. Jantzen, H. Axer, B. Bjerregaard, and D. G. von Keyserlingk, “Evolving rule-based systems in two medical domains using genetic programming,” *Artificial Intelligence in Medicine*, vol. 32, no. 3, pp. 195–216, November 2004.
- [87] G. Folino, C. Pizzuti, and G. Spezzano, “GP ensembles for large-scale data classification,” *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 604–616, Oct. 2006.
- [88] R. Curry, P. Lichodziejewski, and M. I. Heywood, “Scaling genetic programming to large datasets using hierarchical dynamic subset selection,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 37, no. 4, pp. 1065–1073, Aug. 2007.
- [89] D. Song, M. Heywood, and A. Zincir-Heywood, “Training genetic programming on half a million patterns: an example from anomaly detection,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 225–239, June 2005.
- [90] S. Cagnoni, F. Bergenti, M. Mordonini, and G. Adorni, “Evolving binary classifiers through parallel computation of multiple fitness cases,” *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 35, no. 3, pp. 548–555, June 2005.
- [91] S. Harding and W. Banzhaf, “Fast genetic programming and artificial developmental systems on GPUs,” in *Proc. 21st International Symposium on High Performance Computing Systems and Applications HPCS 2007*, May 2007, pp. 2–2.

- [92] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. San Francisco, USA: Morgan Kaufmann Publishers, 2001.
- [93] L. Hall, I. Ozyurt, and J. Bezdek, "Clustering with a genetically optimized approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 103–112, July 1999.
- [94] H. Pan, J. Zhu, and D. Han, "Genetic algorithms applied to multi-class clustering for gene expression data," *Genomics, Proteomics, Bioinformatics*, vol. 1, no. 4, pp. 279–287, 2003.
- [95] M. Laszlo and S. Mukherjee, "A genetic algorithm using hyper-quadtrees for low-dimensional k-means clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 533–543, April 2006.
- [96] J. Handl and J. Knowles, "An evolutionary approach to multiobjective clustering," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 1, pp. 56–76, Feb. 2007.
- [97] D. van der Merwe and A. Engelbrecht, "Data clustering using particle swarm optimization," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 1, 8–12 Dec. 2003, pp. 215–220.
- [98] X. Xiao, E. Dow, R. Eberhart, Z. Miled, and R. Oppelt, "Gene clustering using self-organizing maps and particle swarm optimization," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, 22–26 April 2003, p. 10pp.
- [99] A. Rényi, "On measures of entropy and information," in *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*, Neyman, Ed., vol. 1. Berkeley: University of California Press, 1961, pp. 547–561.
- [100] D. Xu and J. Principe, "Learning from examples with quadratic mutual information," in *Proc. IEEE Signal Processing Society Workshop Neural Networks for Signal Processing VIII*, 31 Aug.–2 Sept. 1998, pp. 155–164.
- [101] E. Parzen, "On estimation of a probability density function and mode," *Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.
- [102] B. W. Silverman, *Density estimation for statistics and data analysis*. New York, NY, USA: Chapman & Hall / CRC, 1986.

- [103] A. Ultsch, "Maps for the visualization of high-dimensional data spaces," in *Proc. Workshop on Self-Organizing Maps*, Kyushu, Japan, 2003, pp. 225–230.
- [104] N. Boric, "GPalta genetic programming toolbox," 2005. [Online]. Available: <http://hayabusa.die.uchile.cl/~nboric/gpalta>
- [105] S. Rao, "Information theoretic clustering demo using Cauchy-Schwartz PDF measure for non-parametric clustering." [Online]. Available: <http://itl.cnel.ufl.edu/Clustering.htm>
- [106] D. Newman, S. Hettich, C. Blake, and C. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: <http://www.ics.uci.edu/~mlearn/MLRepository.html>
- [107] J. Bezdek, J. Keller, R. Krishnapuram, L. Kuncheva, and N. Pal, "Will the real iris data please stand up?" *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 3, pp. 368–369, June 1999.
- [108] E. Anderson, "The irises of the gaspe peninsula," *Bulletin of the American Iris Society*, vol. 59, pp. 2–5, 1935.
- [109] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals Eugen.*, vol. 7, no. 2, pp. 179–188, 1936.
- [110] A. Ultsch, "Clustering with SOM: U*C," in *Proc. Workshop on Self-Organizing Maps*, Paris, France, 2005, pp. 75–82.

Apéndice A

**Publicación en IEEE Congress on
Evolutionary Computation (CEC
2007)**

Genetic Programming-based Clustering Using an Information Theoretic Fitness Measure

Neven Boric and Pablo A. Estévez

Abstract—A clustering method based on multitree genetic programming and an information theoretic fitness is proposed. A probabilistic interpretation is given to the output of trees that does not require a conflict resolution phase. The method can cluster data with irregular shapes, estimate the underlying models of the data for each class and use those models to classify unseen patterns. The proposed scheme is tested on several real and artificial data sets, outperforming k-means algorithm in all of them.

I. INTRODUCTION

Clustering consists in partitioning a data set into subgroups such that pairs of patterns belonging to the same group are more *similar* than those belonging to different groups. This definition, as noted by several authors, is circular because the similarity measure defines the optimal clustering [1].

Although some researchers have suggested that “the objective of cluster analysis is simply to find a convenient and valid organization of the data, not to establish rules for separating future data into categories” [2], it is often desired to use the resulting clustering to predict the labels of unseen data, a process denoted in [1] as prediction based on groups.

Given the variety of research fields where clustering is applied, a multitude of approaches for clustering can be found in the literature. Clustering algorithms can be classified as hierarchical and partitional [3], depending on the form of the generated clustering. A hierarchical algorithm produces a nested sequence of partitions, whereas a partitional algorithm produces a single partition. Most partitional clustering algorithms try to minimize the distance between the data and the cluster prototypes, which is equivalent to minimizing the cluster variance. Such algorithms can detect clusters of hyperspherical shape only.

Information theory has been used by several researchers as a basis for clustering, as it provides means to gather higher order statistics in order to reveal the structure of the data [4]–[6]. Gokcay and Principe [4], [7] presented a clustering evaluation function based on Renyi’s entropy. Li et. al [5] used k-nearest neighbors to estimate Havrda-Charvat cluster entropy. Both approaches use ad-hoc iterative algorithms for optimization. In [6] the recently developed Cauchy-Schwartz divergence between probability density functions [8] is optimized through gradient descent. None of these methods can be used for clustering unseen data.

Genetic Programming (GP) is an evolutionary technique related to genetic algorithms, where the structures evolved

are computer programs, usually represented as trees [9]. GP has proven to be a powerful framework to deal with supervised classification problems. The free nature of GP trees can be exploited to generate programs that separate regions with any kind of shape. Another advantage, shared with other evolutionary techniques, is the capability of avoiding local minima, contrary to gradient descent methods such as backpropagation. GP has been used to solve binary supervised classification problems since its inception [9]. The extension to multiclass problems is not direct, but several approaches have been proposed [10]–[14]. In [10] the multiclass problem is modeled as a set of binary class problems. In [11] the output space of the trees is segmented and each segment is mapped to a class. The best approach so far is the one proposed by Muni et al. [12], [13], where multitree individuals consisting of c trees are evolved for a c class problem. Each tree encodes a rule that is activated when a pattern is said to belong to that class. If multiple rules are activated for the same pattern, a conflict resolution phase is required.

Given the success of GP in supervised classification problems, it seems appropriate to apply GP for clustering. On this matter, the authors are aware of only the work of De Falco et al. [15], which shows the feasibility of using GP for clustering. Their approach is similar to [12] in that it also uses multitree individuals encoding binary rules. Therefore, it requires a conflict resolution phase too. The method allegedly finds the optimal number of clusters by using a variable number of trees per individual. However, instead of specifying c , the user needs to specify another parameter that directly influences the number of clusters found. Cluster variance is used as fitness, constraining clusters to be hyperspherical, and thus, diminishing the main advantage of using GP. Other evolutionary computational techniques applied to clustering include genetic algorithms [16]–[18] and particle swarm optimization [19]. All of them use fitness measures based on cluster variance.

In this paper, a clustering algorithm based on multitree GP is presented. By using an information theoretic fitness measure, the evolved trees are capable of separating clusters of any shape, not only hyperspherical. Unlike other multitree GP approaches that generate a set of binary rules, in our approach the interpretation given to trees’ output is probabilistic and thus no conflict resolution phase is required. Compared to purely information theoretic approaches and other evolutionary techniques, the main advantage of the proposed method is that the output of the algorithm is not only the optimal clustering, but also the program (set of

The authors are with Computational Intelligence Laboratory, Department of Electrical Engineering, Universidad de Chile (e-mail: nboric@ing.uchile.cl; pestevez@ing.uchile.cl).

functions) that generates such clustering. As a consequence the evolved programs can easily be applied to cluster unseen data.

II. GENETIC PROGRAMMING

GP is an evolutionary technique to automatically generate computer programs, based on the Darwinian principle of survival of the fittest [9]. The individuals undergoing evolution are computer programs, encoded as hierarchical trees composed of functions and terminals. Each individual is a possible solution to an optimization problem. A GP simulation starts with a population of program trees generated at random. All trees are evaluated and a fitness value is assigned to each one. Trees with better fitness have a higher probability of being selected as parents for the next generation. Genetic operators such as crossover and mutation are applied to the selected trees. The offspring form the next generation of individuals. The process is repeated until satisfying some stopping criterion.

Koza [9] specified five preparatory steps for applying GP, all of which are problem dependent:

- determining the set of terminals,
- determining the set of functions,
- determining the fitness measure,
- determining the parameters and variables for controlling the run, and
- determining the method of designating a result and the criterion for terminating a run.

For applying GP to multicategory classifier design, whether supervised or unsupervised, some further steps are necessary. As the output of a tree is a scalar, it is necessary to adapt the system so that a label is obtained when evaluating an individual. Several possibilities for solving this issue have been explored in the literature, as mentioned in the introduction. In this paper we focus on the multitree approach, as it has shown the best results so far for supervised classification, and extend this approach to clustering. The following sections present in detail our choice for a fitness measure based on information theory and the probabilistic interpretation of the trees' output.

III. INFORMATION THEORETIC FITNESS MEASURE

A. Background

In pattern recognition, it is often assumed that the patterns corresponding to each of the classes come from a probability density function (pdf) representing the generating model of the data [1], i.e. there exist class conditional pdfs $p(x|C_k)$, $k \in \{1, \dots, c\}$ such that the samples for each C_k are drawn from these pdfs. Of course, if the pdfs were known, the classification problem would be automatically solved by using Bayes decision rule [1]. Thus many supervised classification methods try to estimate the pdfs from data samples in a way that the class labels are maintained.

In unsupervised classification, the C_k labels are unknown, so usually the problem becomes to determine the optimal

labels by minimizing a certain cost function. Most clustering algorithms use a cost function based on second order statistics of the data such as cluster variance.

Information theory provides the foundations to build more complex cost functions that include higher order statistics, and are able to better capture the real structure of the data. The basis of information theory is given by Shannon's entropy. Given a random variable x , its entropy is defined as

$$H(x) = - \int p(x) \ln p(x) dx$$

and it measures the amount of uncertainty present in x or, equivalently, its *randomness*. If the entropy is low, the variable is *less random*. Based on this property, one could minimize the entropy of the patterns belonging to the same cluster, as a lower entropy means that the patterns are more *alike*. The quantity that measures the entropy of all clusters together is the total conditional entropy of x given the clustering C :

$$H(x|C) = - \sum_{k=1}^c P(C_k) \int p(x|C_k) \ln p(x|C_k) dx \quad (1)$$

where $P(C_k)$ is the *a priori* probability that any given pattern belongs to C_k .

Eq. (1) is minimized when all patterns within each cluster are very similar. Following a similar reasoning, another option is to maximize some kind of inter cluster cross entropy to ensure that patterns belonging to different clusters are dissimilar. One such measure is the Kullback-Leibler divergence between pdfs $p(x)$ and $q(x)$:

$$D_{KL}(p, q) = p(x) \int \ln \frac{p(x)}{q(x)} dx \quad (2)$$

Intuitively, both (1) and (2) seem good candidates for creating a fitness function for GP, as they would drive individuals to cluster together similar patterns and separate dissimilar ones. However, these measures are very difficult to estimate directly from data without imposing assumptions about the pdfs [8], and thus some alternative measures of entropy are needed.

B. Cauchy-Schwartz Divergence

Principe et al. [8] devised a way to estimate entropy directly from data. In this section we briefly review it to justify the choice of our fitness function. Instead of Shannon's entropy, they use Renyi's entropy, given by:

$$H_\alpha(x) = \frac{1}{1-\alpha} \ln \int p^\alpha(x) dx \quad (3)$$

Given a data set $X = \{x_1, \dots, x_n\}$, $x_i \in \mathbb{R}^d$, a Parzen window with a Gaussian kernel can be used to estimate $p(x)$:

$$\tilde{p}(x) = \frac{1}{n} \sum_{i=1}^n G(x - x_i, \sigma^2) \quad (4)$$

where

$$G(x - x_i, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp\left\{-\frac{\|x - x_i\|^2}{2\sigma^2}\right\}$$

The σ^2 parameter is called the kernel size and must be specified *a priori*. Replacing (4) into (3), and using $\alpha = 2$, it yields:

$$H_2(x) \approx -\ln \int \frac{1}{n} \sum_{i=1}^n G(x - x_i, \sigma^2) \frac{1}{n} \sum_{j=1}^n G(x - x_j, \sigma^2) dx$$

which can be calculated in exact form by exchanging the integral with the sum operators and applying the convolution theorem for gaussians [8]. The final result is

$$H_2(x) \approx -\ln \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n G_{ij, 2\sigma^2}$$

where

$$G_{ij, \sigma^2} \equiv G(x_i - x_j, \sigma^2)$$

The quantity

$$V(x) = \sum_{i=1}^n \sum_{j=1}^n G_{ij, 2\sigma^2} \quad (5)$$

is called the Information Potential (IP) [8]. Maximizing $V(x)$ is equivalent to minimizing the entropy of x , therefore it could be used to approximate (1) as a cost function for clustering problems.

Principe et al. [8] also proposed divergence measures for comparing two pdfs that can be estimated in the same way as (5). One of these measures is the Cauchy-Schwartz (CS) divergence, defined as:

$$D_{CS}(p, q) = -\ln \frac{\int p(x) q(x) dx}{\sqrt{\int p^2(x) dx \int q^2(x) dx}}$$

Jenssen et al. [6] used the CS divergence as a cost function for clustering in the following way. Assume the problem consists of separating the data sets in two clusters, then it makes sense to maximize $D_{CS}(p_1, p_2)$, where $p_1 = p(x|C_1)$, $p_2 = p(x|C_2)$. Using again the Parzen window to estimate both pdfs, D_{CS} can be rewritten as

$$D_{CS}(p, q) \approx -\ln \frac{\sum_{x_i \in C_1} \sum_{x_j \in C_2} G_{ij, 2\sigma^2}}{\sqrt{\sum_{x_i, x_{i'} \in C_1} G_{ii', 2\sigma^2} \sum_{x_j, x_{j'} \in C_2} G_{jj', 2\sigma^2}}} \quad (6)$$

Notice that the numerator of (6) evaluates the gaussian kernel for pairs of patterns belonging to different clusters. Instead, each summation term in the denominator includes only pairs of patterns that are on the same cluster.

To extend (6) for multiple clusters, a crisp membership function of the form $m(x) : \mathbb{R}^d \rightarrow \{0, 1\}^c$ is introduced, where every pattern is assigned a vector $m(x) = \{m_k\}^T$,

$k \in \{1, \dots, c\}$. Each component of m indicates if the pattern belongs to a certain cluster:

$$m_k(x) = \begin{cases} 1 & x \in C_k \\ 0 & x \notin C_k \end{cases}$$

Using m and omitting the logarithm, the Cauchy-Schwartz cost function is defined as

$$J_{CS}(p_1, \dots, p_c) = \frac{\frac{1}{2} \sum_{i,j=1}^n (1 - m^T(x_i) m(x_j)) G_{ij, 2\sigma^2}}{\sqrt{\prod_{k=1}^c \sum_{i,j=1}^n m_k(x_i) m_k(x_j) G_{ij, 2\sigma^2}}} \quad (7)$$

The denominator of (7) consists of the multiplication of the information potentials of the patterns belonging to each cluster. Recalling that $D_{CS} = -\ln(J_{CS})$, minimizing J_{CS} implies minimizing the sum of “within cluster” entropies, exactly as in (1). The numerator, on the other hand forms a “between cluster” cross information potential, and thus minimizing J_{CS} implies maximizing the inter cluster cross entropy, similar to (2).

Expressed in terms of minimizing (7), the clustering problem becomes an optimization problem over the set of $m_k(x)$ functions.

C. Determination of σ

Notice that in the derivation of (6), the kernel size σ for each Parzen estimator was chosen the same. This implies that the pdf estimation will not truthfully represent every underlying pdf, no matter the choice of σ . However, Jenssen et al. suggested to estimate the optimal kernel size for the whole data set by using Silverman’s rule of thumb:

$$\sigma_{\text{opt}} = \sigma_X \left(\frac{4}{n(2d+1)} \right)^{\frac{1}{d+4}} \quad (8)$$

where $\sigma_X^2 = \frac{1}{d} \sum_{i=1}^d \sigma_{ii}^2$, and σ_{ii}^2 are the variances for each dimension of the data. This choice of σ is sufficient for most cases shown below.

IV. GP-BASED CLUSTERING ALGORITHM

The proposed approach uses GP to obtain the set of membership functions that minimize (7), which is used as a fitness measure. The aim is to produce clustering solutions that assign highly similar patterns to the same cluster and dissimilar patterns to different clusters without imposing any shape constraints to the resulting partitions. The main drawback of J_{CS} as a fitness measure is that its calculation is $O(N^2)$.

A. Multitree Representation

Similar to [12], in our system an individual is formed by c trees, each of them associated with one of the clusters that are to be formed. The i th individual of the population is denoted as I_i and each of its trees as T_k^i . The output of I_i is a vector combining the outputs of all its trees, $I_i(x) = [T_1^i(x), \dots, T_c^i(x)]^T$. As in most clustering algorithms, the numbers of clusters c is specified beforehand by the user. Fig. 1 shows an example of an individual in our system.

B. Probabilistic Interpretation

The optimization problem described in the previous section has degenerated solutions that must be avoided in order to achieve a proper clustering. For instance, if all the patterns were assigned to a single cluster, the numerator of (7) would be exactly zero, yielding an unwanted optimum. To overcome this problem the membership functions are extended to the continuous range $[0, 1]$. If the membership functions are further restricted to sum to one such that

$$\sum_{k=1}^c m_k(x_i) = 1, \forall i = 1, \dots, n$$

they can be regarded as a set of *a posteriori* probabilities $m_k(x) = P(C_k|x)$. Our GP based approach tries to obtain these normalized membership functions. Notice that by using GP, the functions themselves are evolved, instead of just their values for the given data set. Using Bayes theorem the class conditional pdfs $p(x|C_k)$ can be obtained:

$$p(x|C_k) = \frac{P(C_k|x)p(x)}{P(C_k)} \quad (9)$$

These pdfs approximate the real class conditionals only to the extent of minimizing J_{CS} , given the (often limited) availability of data samples.

The output of a GP tree is unrestricted in \mathbb{R} . In order to interpret the trees' output as posterior probabilities, some transformations must be made. First, the logistic sigmoid function is applied

$$\tilde{T}_k(x) = \frac{1}{1 + e^{-T_k(x)}}$$

to insure that $\tilde{T}_k(x) \in [0, 1]$. Then, $\tilde{T}_k(x)$ is normalized with respect to all the outputs of the trees in the same individual, i.e.

$$P(C_k|x) = \frac{\tilde{T}_k(x)}{\sum_{k'=1}^c \tilde{T}_{k'}(x)}$$

In summary, the output of an individual in our approach is a vector

$$I_i(x) = [P_i(C_1|x), \dots, P_i(C_c|x)]^T$$

where the components correspond to the trees' output transformed such that they are all in $[0, 1]$ and sum to 1.

To decide to which cluster a pattern belongs to, the membership functions must be converted back to crisp values. The Bayes rule is to assign pattern x to class $C_{\hat{k}}$ so that

$$\hat{k} = \underset{k}{\operatorname{argmax}} P(C_k|x) \quad (10)$$

In this way, a label can be assigned to each pattern avoiding a conflict resolution phase.

C. Terminal and Function Sets

The function set used in this work is $F = \{+, -, \times, /, (\cdot)^2, \sqrt{\cdot}, \sin, \cos\}$. The terminal set corresponds to $T = \{\text{feature variables}, R\}$, where R are random constants in $[0, 100]$.

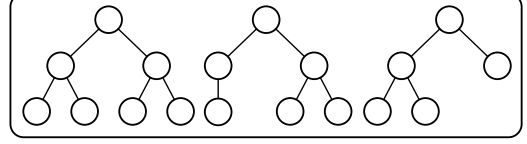


Fig. 1. An example of a multitree individual with $c = 3$

D. Genetic Operators

When using multitree GP, it is necessary to specify new genetic operators that take into account the structure of individuals. Muni et al. [12] introduced an operator that performs crossover between trees that are in the same position in their respective multitree individuals (i.e. for individuals I_{i_1} and I_{i_2} , perform crossover between trees $T_{k_1}^{i_1}$ and $T_{k_2}^{i_2}$). In this way only trees representing the same class are mixed together. However, this operator loses its appeal for clustering tasks because clusters are indistinguishable between each other during evolution. As the class labels are not available during the GP run, it would be misleading to use a crossover operation that always mixes trees that are in the same position in their respective individuals. In our approach we randomly select one tree from two parents and then perform the usual crossover operation between them (i.e. for individuals I_{i_1} and I_{i_2} , select k_1, k_2 at random and perform crossover between trees $T_{k_1}^{i_1}$ and $T_{k_2}^{i_2}$). This has the disadvantage that the evolution is less directed and more random than in [12]. On the other hand, this approach follows the rationale of the original crossover operation, that randomly selects subtrees to be exchanged without further care for *directing* the evolution.

V. EXPERIMENTAL RESULTS

A. Cluster Evaluation

There exist several ways to measure the performance of a clustering algorithm. If previous knowledge about a data set in the form of class labels is available, the external criterion is applicable [1]. This criterion consists in comparing the partition obtained using the clustering algorithm with the actual partition (existing labels). Several indices have been developed for this purpose, such as the (adjusted) Rand index [20], [21]. If the number of clusters c is equal to the number of predefined classes of the data, it is possible to translate the cluster labels into class labels and calculate a rate of correct classification. A translation process is necessary because it is not possible to know beforehand how clusters and class labels are related to each other. Intuitively, this is solved by associating a cluster with the class of the majority of patterns in that cluster. Although it is usually left unnoticed, most publications implicitly use this interpretation when stating how many patterns were correctly classified or mislabeled by an algorithm. See for instance [16].

B. Experiment Design

To evaluate the performance of the proposed method, three kinds of experiments were carried out. In all of them, the

number of clusters c was set equal to the number of actual classes.

1) *Clustering*: This is a simple clustering experiment. For each data set, the GP is run using all available samples. The resulting membership functions are evaluated on each pattern and a label is assigned using (10). The obtained labels are compared with the actual class labels and a classification score is obtained. The process is repeated 10 times to deal with the random nature of GP. The GP results are compared with those of the popular k-means algorithm [22].

2) *Generalization*: Here we try to test whether our approach is capable of learning the distribution of each cluster. For this aim, the GP is trained with only a subset of the data and tested on the remaining patterns. A tenfold cross validation procedure was used as follows: first, randomly split the data set into ten subsets. For each GP run, one of the subsets is left out as the validation set. The remaining nine subsets form the training set. The evolution is run using only the training set. The resulting membership functions are evaluated on the validation set and labels are assigned and compared with the actual labels. The process is repeated 10 times (until every subset has been used as a validation set) and the classification scores are averaged.

3) *Learning Conditional pdf*: On this experiment we intend to show if our approach is capable of estimating the pdf for the data of each class. With this aim, the GP is trained on a two dimensional data set specially designed for graphical evaluation. It consists of four easily separable gaussian clusters, with 100 patterns each. All patterns are used during the training phase. The resulting trees are evaluated on a grid of equidistant points within the range of the data set. This procedure allows us to obtain a graphical representation of the set of *a posteriori* pdfs, $P(C_i|x)$, obtained by our GP approach. Finally, applying Bayes theorem for every point in the grid, the set of class conditional pdfs $p(x|C_i)$ for each cluster is obtained. The graphical representation of the class conditionals should show a single gaussian for each cluster.

C. Data Sets

The clustering and generalization tests were performed on two databases commonly used for benchmarking purposes and on an artificial problem suite specially designed for clustering applications. This gives a total of 10 data sets ranging from 2 to 9 dimensions and from 150 to 4096 patterns.

1) *Iris*: This data set corresponds to the well known Iris data set by Fisher [23]. It contains 150 patterns in four dimensions measured on Iris flowers of three different species. Each class is represented by 50 patterns.

2) *Wisconsin Breast Cancer (WBC)*: It consists of 699 instances of 9 attributes measured on cancer patients. Each instance is labeled either as benign (65.5%) or malignant (34.5%). Sixteen instances were removed as they contain missing values. WBC was obtained from UCI online repository [24].

TABLE I
SUMMARY OF PROPERTIES OF FCPS DATA SETS

| Data set | Size | Dimensions | Classes |
|-------------|------|------------|---------|
| Hepta | 212 | 3 | 7 |
| Lsun | 400 | 2 | 3 |
| Tetra | 400 | 3 | 4 |
| Chainlink | 1000 | 3 | 2 |
| Atom | 800 | 3 | 2 |
| EngyTime | 4096 | 2 | 2 |
| TwoDiamonds | 800 | 2 | 2 |
| WingNut | 1070 | 2 | 2 |

TABLE II
VALUES OF GP PARAMETERS COMMON TO ALL EXPERIMENTS

| Parameter | Value |
|-----------------------------|-------|
| Max. Generations | 200 |
| Population Size | 1000 |
| Max. Tree Depth | 8 |
| Tournament Size | 5 |
| Probability of Crossover | 0.85 |
| Probability of Mutation | 0.05 |
| Probability of Reproduction | 0.1 |

3) *Fundamental Clustering Problem Suite (FCPS)*: This is a suite of artificial data sets designed by Ultsch [25] that tries to represent all the problems that a clustering algorithm may face. Each data set was designed to test specific scenarios where some clustering algorithms fail. All the data sets are of dimension less than or equal to 3, allowing one to visually assess the performance of a clustering algorithm. Table I shows the main characteristics of 8 data sets taken from this problem suite. Fig. 2 shows a graphical representation of the data sets. FCPS contains two more data sets that were not used in this work, namely “GolfBall” and “Target”. The former was not used since it contains no clusters, and the latter because some clusters are formed by only three patterns. See [25] for more details on FCPS.

D. Results

The experiments were performed using GPAlta toolbox [26]. Table II shows the value of the parameters used for all GP runs.

The results for the clustering experiment are shown in Table III. It can be seen that the proposed method made a correct clustering of all the data sets except for the “ChainLink” data set, where there is a larger discrepancy between the best and average results. To explain the later result the fitness values for every GP run were analyzed. For all the data sets, the best fitness value matched the best classification score. This confirms that the cost function combined with σ estimation form a good fitness measure, as there are no global optima that lead to undesirable results. Consequently, we believe that the poor convergence on the “ChainLink” data set was caused by the complicated structure of the data, which makes it difficult to generate a set of functions that appropriately partitions it. Nevertheless, in all the data sets, our approach outperformed the popular k-means algorithm.

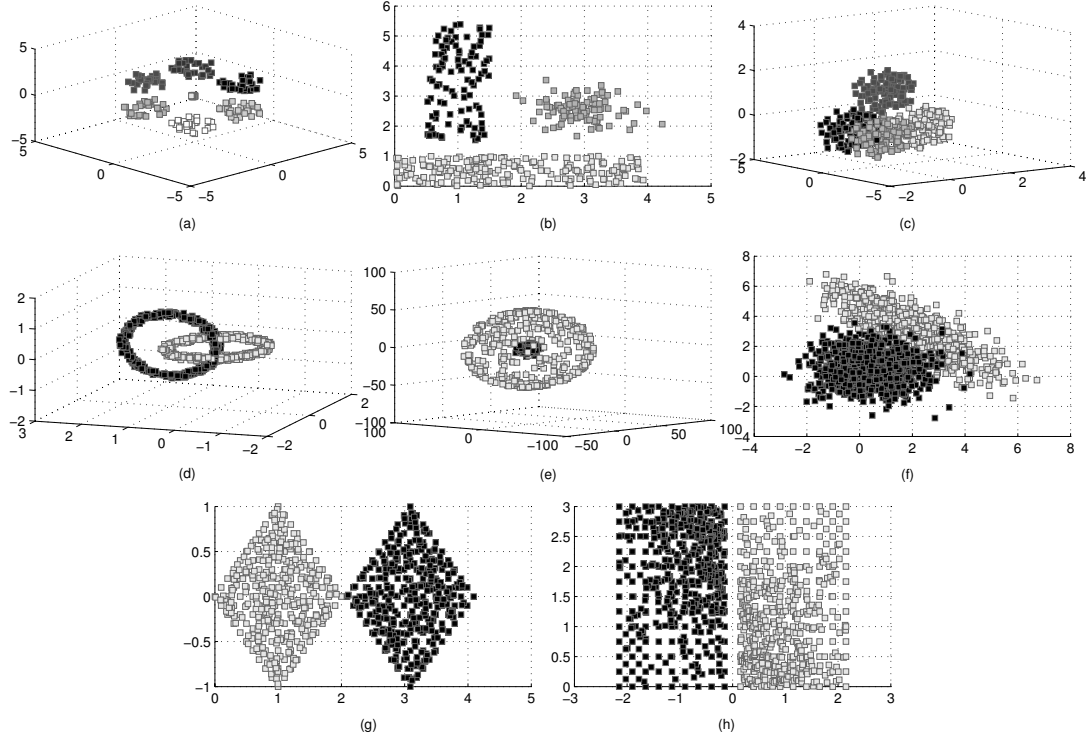


Fig. 2. Fundamental Clustering Problem Suite data sets [25]: (a) Hepta; (b) Lsun; (c) Tetra; (d) Chainlink; (e) Atom; (f) EngyTime; (g) TwoDiamonds; (h) WingNut

TABLE III
CLUSTERING RESULTS USING THE GP BASED APPROACH AND
K-MEANS

| Data sets | GP | | k-means | |
|-------------|---------|-------|---------|-------|
| | Average | Best | Average | Best |
| Iris | 93.00 | 97.33 | 89.33 | 89.33 |
| WBC | 96.86 | 97.21 | 96.04 | 96.04 |
| Hepta | 100 | 100 | 76.09 | 100 |
| Lsun | 99.90 | 100 | 76.20 | 76.50 |
| Tetra | 99.40 | 100 | 96.40 | 100 |
| Chainlink | 84.68 | 100 | 65.42 | 65.90 |
| Atom | 99.58 | 100 | 71.53 | 72.13 |
| EngyTime | 95.08 | 96.46 | 95.14 | 95.14 |
| TwoDiamonds | 99.89 | 100 | 100 | 100 |
| WingNut | 99.21 | 100 | 96.36 | 96.36 |

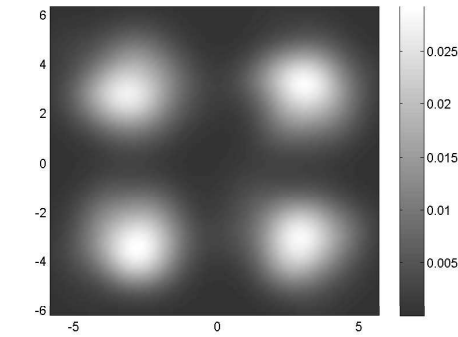
TABLE IV
GENERALIZATION RESULTS USING THE GP APPROACH

| Data sets | Classification |
|-------------|----------------|
| Iris | 92.67 |
| WBC | 90.50 |
| Hepta | 98.74 |
| Lsun | 92.75 |
| Tetra | 97.75 |
| Chainlink | 78.30 |
| Atom | 99.17 |
| EngyTime | 93.69 |
| TwoDiamonds | 99.88 |
| WingNut | 99.51 |

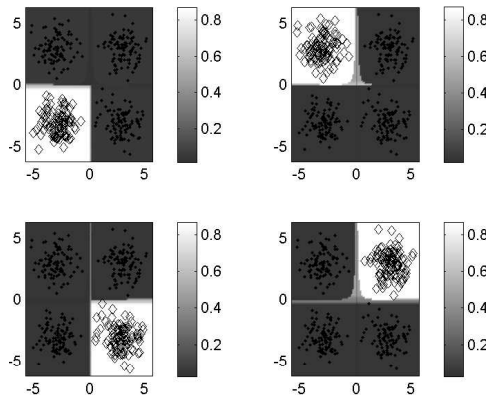
The results for the generalization experiment are shown on Table IV. On every case, our method was able to classify the patterns from the validation set with almost the same performance as in the clustering experiment. These results show that the proposed approach, although trained in an unsupervised way, can generalize well and the resulting trees can be easily applied to classify unseen data.

Fig. 3 shows a typical result for the learning conditional pdf experiment. Fig. 3(a) shows the Parzen window estimation of $p(x)$. The estimation yields four gaussians centered on the actual centers for each class, showing that the Parzen

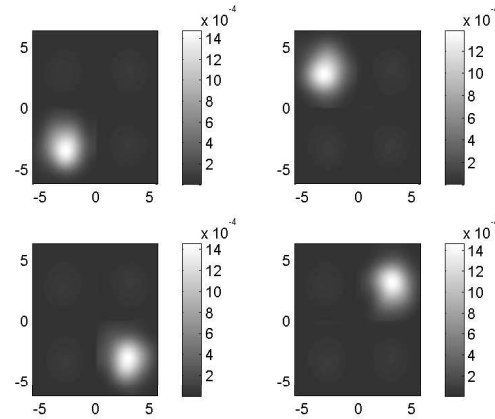
method is a good estimator for the pdf of the entire data set. Fig. 3(b) has four subfigures. Each of them shows the normalized output of one of the resulting trees. For a better interpretation, the data set is also represented on each subfigure, with the patterns associated to each cluster emphasized. It can be seen how the trees properly divide the input space. On Fig. 3(c) the output of the trees and the Parzen estimator are combined using (9). It can be observed that our method was capable of separating the four gaussians, providing a good estimation for the class conditional pdfs that clearly resemble the actual distribution of the data for each class.



(a)



(b)



(c)

Fig. 3. Typical results for learning conditional pdf experiment: (a) Parzen window $p(x)$ estimation; (b) $P(C_i|x)$ as given by the normalized trees' output; (c) Estimation of $p(x|C_i)$ obtained using Bayes theorem.

E. Computational Costs

In GP all individuals must be evaluated to get their fitness value in every generation. This could be very costly. The fitness measure used in this paper is $O(N^2)$, which adds more complexity. However, the method usually finds the best individual within the first 100 generations. Average execution times for a single GP run range from 5 minutes to 5 hours, depending on the size of the data set. In our opinion the benefits of the proposed method outweigh the added complexity, at least for the studied data sets. Stochastic subsampling as described in [6] was explored, but it was found to hinder the performance of the algorithm.

VI. CONCLUSIONS

We have extended the recent work on GP applied to classifier design into the field of clustering. By combining GP with an information theoretic fitness measure, our method is able to cluster data without any shape constraints. The probabilistic interpretation given to trees' output eliminates the need for a conflict resolution phase present in other GP applications. Although the same as well as similar cost functions have been optimized by other authors through different means, the main advantage of using GP is the ability to *learn* the membership functions themselves instead of simply partitioning the data. In the generalization experiment, it was shown that these functions can be applied to cluster unseen data with high accuracy. Furthermore, the system was shown to indirectly evolve the generating model for the data, as expressed by the conditional pdfs encoded in GP trees.

Further research efforts should be directed towards improving the convergence of the method and reducing its computational costs. It would also be of interest to perform a comparison between the proposed method and other modern evolutionary approaches, specially when applied on high dimensional data sets.

ACKNOWLEDGMENTS

This research was partially supported by FONDECYT 1050751. Neven Boric received a scholarship from CONICYT-CHILE for pursuing graduate studies.

REFERENCES

- [1] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. San Diego, CA, USA: Academic Press, 1999.
- [2] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.
- [3] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.
- [4] E. Gokcay and J. Principe, "Information theoretic clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 2, pp. 158–171, Feb. 2002.
- [5] H. Li, K. Zhang, and T. Jiang, "Minimum entropy clustering and applications to gene expression analysis," in *Computational Systems Bioinformatics Conference, 2004. CSB 2004. Proceedings. 2004 IEEE*, 16–19 Aug. 2004, pp. 142–151.
- [6] R. Jenssen, D. Erdogmus, K. E. Hild II, J. C. Principe, and T. Eltoft, "Optimizing the cauchy-schwarz pdf distance for information theoretic, non-parametric clustering," in *EMMCVPR, 2005*, pp. 34–45.
- [7] E. Gokcay and J. Principe, "A new clustering evaluation function using renyi's information potential," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, vol. 6, 5–9 June 2000, pp. 3490–3493.

- [8] J. Principe, D. Xu, and J. Fisher, "Information theoretic learning," in *Unsupervised Adaptive Filtering*, S. Haykin, Ed. New York: John Wiley & Sons, 2000.
- [9] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.
- [10] J. Kishore, L. Patnaik, V. Mani, and V. Agrawal, "Application of genetic programming for multiclass pattern classification," *IEEE Trans. Evol. Comput.*, vol. 4, no. 3, pp. 242–258, Sept. 2000.
- [11] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2, 27–30 May 2001, pp. 1070–1077vol.2.
- [12] D. Muni, N. Pal, and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 183–196, April 2004.
- [13] —, "Genetic programming for simultaneous feature selection and classifier design," *IEEE Trans. Syst., Man, Cybern. B*, vol. 36, no. 1, pp. 106–117, Feb. 2006.
- [14] L. Cordelia, C. De Stefano, F. Fontanella, and A. Marcelli, "Genetic programming for generating prototypes in classification problems," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 2, 2–5 Sept. 2005, pp. 1149–1155.
- [15] I. D. Falco, E. Tarantino, A. D. Cioppa, and F. Gagliardi, "A novel grammar-based genetic programming approach to clustering," in *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*. New York, NY, USA: ACM Press, 2005, pp. 928–932.
- [16] L. Hall, I. Ozyurt, and J. Bezdek, "Clustering with a genetically optimized approach," *Evolutionary Computation, IEEE Transactions on*, vol. 3, no. 2, pp. 103–112, July 1999.
- [17] S. Bandyopadhyay and U. Maulik, "Nonparametric genetic clustering: comparison of validity indices," *IEEE Trans. Syst., Man, Cybern. C*, vol. 31, no. 1, pp. 120–125, Feb. 2001.
- [18] W. Sheng, S. Swift, L. Zhang, and X. Liu, "A weighted sum validity function for clustering with a hybrid niching genetic algorithm," *IEEE Trans. Syst., Man, Cybern. B*, vol. 35, no. 6, pp. 1156–1167, Dec. 2005.
- [19] D. van der Merwe and A. Engelbrecht, "Data clustering using particle swarm optimization," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 1, 8–12 Dec. 2003, pp. 215–220.
- [20] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, pp. 622–626, 1971.
- [21] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, pp. 193–218, 1985.
- [22] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, 1967, pp. 281–297.
- [23] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals Eugen.*, vol. 7, no. 2, pp. 179–188, 1936.
- [24] D. Newman, S. Hettich, C. Blake, and C. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: <http://www.ics.uci.edu/~mllearn/MLRepository.html>
- [25] A. Ultsch, "Clustering with SOM: U*C," in *Proc. Workshop on Self-Organizing Maps*, Paris, France, 2005, pp. 75–82.
- [26] N. Boric, "GPAlta genetic programming toolbox," 2005. [Online]. Available: <http://hayabusa.die.uchile.cl/~nboric/gpalta>