Aprobación de Borrador de Tesis

Obtención de Conjuntos Aproximados

Mediante

Algoritmos Genéticos

Javier Trejos Zelaya, Ph.D.

Profesor Asesor (Visto Bueno)

Carlos González, Ph.D.

Director Maestría (Visto Bueno)

A Veracruz, por su comprensión,

a Viviana y Sebastián, mis fuentes de inspiración.

Todo lo que ocurre una vez
puede que no ocurra nunca ms.
Pero todo lo que ocurre dos veces
ocurrir ciertamente una tercera.
P. Coelho

Chapter 1

Conjuntos aproximados

Teniendo un conjunto de objetos sobre los que se ha medido una serie de características o atributos cualitativos, se llaman *conjuntos aproximados* a combinaciones de esos atributos que describan lo mejor posible al conjunto de objetos. En muchas ocasiones se tiene además una partición o clasificación previa de los objetos y lo que se pretende es dar una caracterización de las clases de esa partición mediante los atributos observados ([23], [33]). Este es un objetivo muy similar al de la discriminación en estadística ([7]).

1.1 Introducción

En términos generales, la teoría de conjuntos aproximados, propuesta por Pawlak ([22]), consiste en explorar las particiones asociadas a las relaciones de equivalencia generadas por cada subconjunto de los atributos observados, calculando un índice que indique la medida en que éstas coincidan con la clasificación conocida de los datos. Tal exploración tiene una complejidad computacional más elevada que la de los métodos clásicos de discriminación, pues el espacio de búsqueda de soluciones tiene una cardinalidad muy alta y en ocasiones, ante la dificultad de hacer un reco- rrido exhaustivo de todas las posibilidades, el usuario tiene que conformarse con una solución no óptima del criterio. Los algoritmos genéticos tienen la cualidad, en diversos problemas, de aproximar bastante bien las soluciones óptimas.

La importancia que tienen los conjuntos aproximados en el tratamiento estadístico de

datos es que permiten, por un lado, eliminar información superflua y, por tanto, la posible omisión de mediciones innecesarias que se hacen de algunos atributos definidos sobre los datos, con el correspondiente ahorro de tiempo y la disminución de costos. Por otro lado, las tablas de datos reducidas están más cerca de lo que puede ser una base de conocimiento para la generación de sistemas expertos.

Los conjuntos aproximados (CA) fueron introducidos por Pawlak ([22]) en los años ochenta. Su estudio se ubica dentro del campo de la inteligencia artificial que tiene que ver con la vaguedad de datos imprecisos, tratando de descubrir relaciones entre éstos ([23]). Desde el punto de vista matemático, el enfoque de conjuntos aproximados es bastante simple, pues sólo requiere del manejo de conjuntos finitos, cardinalidades y relaciones de equivalencia. Su fortaleza está en que constituye una herramienta para la simplificación de la información y para que los datos concretos se hagan más discernibles.

Esta teoría tiene mucho en común con lo que se realiza en algunos métodos estadísticos, especialmente con el *análisis discriminante* en el que, a grandes rasgos, se tiene cierta cantidad de variables medidas sobre una colección finita de objetos; una de esas variables (variable a explicar) depende de las otras (variables explicativas) y se calcula la medida en que tal dependencia ocurre y las que más aportan a tal dependencia. Pero el enfoque de conjuntos aproximados está especialmente justificado en el caso de que el conjunto de datos experimentales sea muy pequeño, en los que no tiene suficiente validez la aplicación de métodos estadísticos estándar ([32]).

1.2 Conceptos básicos

Como punto de partida, se tiene un conjunto $\Omega = \{\omega_1, \omega_2, ..., \omega_n\}$ de n observaciones o datos sobre los que se han medido p atributos cualitativos. Además, los datos están previamente clasificados en k clases o categorías Y_1, Y_2,...,Y_k, siendo Ω la unión disjunta de estas clases.

Si uno o más de los atributos es cuantitativo puede usarse un algoritmo conocido como

Algoritmo de Fisher para hacer una recodificación de tales atributos, de forma que todos los atributos sean cualitativos ([8]) y tenga sentido aplicar la teoría de conjuntos aproximados a la tabla resultante. Otro caso que puede darse es que en la tabla de datos haya datos faltantes; en [32] se proponen formas de analizar este problema, pero este caso no estuvo contemplado en la investigación de tesis realizada.

Dada la clase Y_j, ésta puede verse como uno de los conceptos que han servido para categorizar a los individuos; lo que se busca es un suconjunto de atributos (la menor cantidad posible) que caractericen totalmente a una buena aproximación Y_j' de Y_j. La aproximación Y_j' se construye vía una relación de equivalencia definida sobre Ω. Se pretende encontrar una relación definida por un subconjunto P del conjunto de atributos en la que haya la mayor cantidad posible de individuos de Y_j' relacionados con otros de Y_j.

Este tipo de aproximaciones de Y_j permite eliminar atributos y observaciones redundantes para Y_j en la caracterización de este conjunto. La eliminación de atributos permite obtener un modelo para Y_j al crear un conjunto mínimo de variables que expliquen suficientemente el concepto Y_j.

Un CA es un modelo matemático para la clasificación aproximada que hace uso de subconjuntos de Ω llamados *aproximación inferior* y *aproximación superior* que se calculan respecto a un subconjunto del conjunto de atributos.

Si $X = \{x^1, x^2, ..., x^p\}$ es el conjunto de atributos, $x^j(\omega_i)$ denota el valor del atributo x^j en la observación w_i. Dado $P \subseteq X$, $P \neq \emptyset$, se define la relación $\wp$:

$$\forall x^j \in P, \;\; \omega_i \wp \omega_h \iff x^j(\omega_i) = x^j(\omega_h).$$

Esta relación $\wp$ definida sobre Ω, e inducida por P, es llamada *relación de indiscernibilidad* pues cualesquiera dos objetos que se relacionen son indiscernibles respecto a las variables presentes en P. Es sencillo verificar que $\wp$ es una relación de equi- valencia y por tanto, produce una partición de Ω, $\Omega/\wp$, formada por sus clases de equivalencia $[\omega]_\wp$, $\forall \;\; \omega \in \Omega$.

Para cada una de las categorías Y_j se definen los conjuntos:

$$I(Y_j) = \{\omega \in \Omega : [\omega]_\wp \subseteq Y_j\}$$

$$S(Y_j) = \{\omega \in \Omega : [\omega]_\wp \cap Y_j \neq \emptyset\}.$$

$I(Y_j)$ y $S(Y_j)$ son, respectivamente, la *aproximación inferior* y *superior* de Y_j. Es claro que $I(Y_j) \subseteq S(Y_j)$. El conjunto $Fr(Y_j) = S(Y_j) - I(Y_j)$ es la *frontera* de la aproximación y es una región de duda.

1.2.1 Medidas de calidad de la aproximación

Para cada Y_j, y para todo $P \subseteq X$, con $P \neq \emptyset$, se define la *calidad de la aproximación del conjunto* Y_j como el índice

$$W_j = \frac{\text{card } I(Y_j)}{\text{card } S(Y_j)}.$$

También se define la *calidad de la aproximación de la clasificación* como el índice

$$W_C = \frac{\sum_{i=1}^{k} \text{card } I(Y_i)}{\sum_{i=1}^{k} \text{card } S(Y_i)}$$

y el siguiente índice sirve para medir la *exactitud de la aproximación de la clasificación*

$$W_e = \frac{\sum_{i=1}^{k} \text{card } I(Y_i)}{\text{card } \Omega}.$$

La *suma inferior* es $\underline{S} = \sum_{i=1}^{k} \text{card } I(Y_i)$

La *suma superior* es $\overline{S} = \sum_{i=1}^{k} \text{card } S(Y_i)$

De los tres índices (W_j, W_C y W_e)[1], que son los que usualmente aparecen en la literatura de conjuntos aproximados, el primero es específico para medir la calidad con que la partición

[1]Para no recargar la notación se han escrito de esta forma, pero debe recalcarse que todos dependen del subconjunto de atributos P, por lo que una notación más acorde con tal dependencia sería: $W_j(P)$, $W_C(P)$ y $W_e(P)$, respectivamente.

inducida por el conjunto P está aproximando a la categoría Y_j; en cambio, los dos últimos son globales. Puede observarse que al calcular cualquiera de estos índices, el resultado está en el intervalo $[0, 1]$ y, en cualquiera de los tres, un buen subconjunto P de atributos se ve reflejado en un valor del índice igual o cercano a uno; contrariamente, cuando el resutado del índice está cerca de cero, P da una aproximación muy pobre.

1.2.2 Ejemplo

Considérese la siguiente tabla de datos, que consiste en los valores de cuatro atributos x^1, x^2, x^3 y x^4, medidos en nueve individuos representados en las filas. La última columna se refiere a la categoría (Y_1 o Y_2) en que viene clasificado cada individuo.

Individuo	x^1	x^2	x^3	x^4	**Clase**
1	0	1	1	0	**1**
2	2	1	0	1	**1**
3	0	0	0	1	**1**
4	1	0	2	0	**1**
5	0	1	1	0	**1**
6	2	1	0	1	**2**
7	0	1	1	0	**2**
8	1	0	2	1	**2**
9	0	0	0	1	**2**

De acuerdo con la última columna de la tabla, las categorías que forman la clasificación del conjunto de datos son

$$Y_1 = \{1, 2, 3, 4, 5\},$$

$$Y_2 = \{6, 7, 8, 9\}.$$

Dado el subconjunto de atributos $P = \{x^2, x^4\}$, éste induce la siguiente partición sobre Ω:

$$\Omega/_{\wp} = \{\{1,5,7\},\{2,6\},\{3,8,9\},\{4\}\}$$

Al calcular los índices presentados anteriormente, se tienen los siguientes valores:

- $W_1 = \dfrac{1}{3+2+3+1} = \dfrac{1}{9}$
- $W_2 = \dfrac{0}{2+3+3} = 0$
- $W_C = \underline{S}/\overline{S} = \dfrac{1+0}{3+2+3+1+2+3+3} = \dfrac{1}{17}$
- $W_e = \underline{S}/9 = \dfrac{1}{9}$

Pueden interpretarse los índices calculados como sigue: La partición $\Omega/_{\wp}$ aproxima a la clase Y_1 con un índice de calidad $W_1 = \frac{1}{9}$, lo cual refleja que el subconjunto de atributos $\{x^2, x^4\}$ representan muy mal a la clase Y_1 y, de paso, que la región de duda es grande. Algo similar ocurre con la clase Y_2, pues $W_2 = 0$, por lo que ni un solo elemento de Y_2 queda bien clasificado usando esos dos atributos. Por otro lado, la calidad global $W_C = \dfrac{1}{17}$ es muy baja y es la proporción global que hay entre las aproximaciones inferiores y superiores, consideradas en forma global. Finalmente el índice de exactitud $W_e = \dfrac{1}{9}$ indica la proporción de individuos de Ω que quedan bien clasificados en Y_1 o en Y_2 mediante los dos atributos considerados.

1.2.3 Otros índices de calidad propuestos

Otros índices globales de calidad que podrían estudiarse y que dependen de un subconjunto P de atributos, son los siguientes:

1. El promedio ponderado de las calidades de aproximación de los conjuntos Y_j:

$$\widetilde{W} = \sum_{j=1}^{k} \frac{card(Y_j)}{card(\Omega)} W_j.$$

La escogencia de un subconjunto de atributos que haga alto este índice estaría dándole mayor peso a las clases con una cardinalidad alta.

2. La proporción de información parcial total dada por un conjunto de atributos

$$\widetilde{\widetilde{W}} = \frac{\sum_{[\omega]_\wp} \text{máx}\{card([\omega]_\wp \cap Y_j), \text{ j=1...k}\}}{card(\Omega)}$$

Este índice que proponemos es una adaptación del usado en [12] para la escogencia del mejor atributo que servirá para abrir en un nodo determinado, en el proceso de generación de un árbol de decisión. Así como está planteado, se diferencia del índice W_e, en que no es necesario que una clase de la partición inducida por P esté enteramente contenida en Y_j para entrar a contar en las sumas inferiores, sino que toma en cuenta también que algunas clases $[\omega]_\wp$ pueden estar "casi" contenidas en Y_j. En [12] se reporta que el índice en que nos basamos para formular $\widetilde{\widetilde{W}}$ tiene un mejor desempeño en la mayoría de experimentos de generación de árboles de decisión realizados, en contraste con otros algoritmos, por lo que presumimos que tiene sentido aplicarlo a la generación de conjuntos aproximados.

Por ejemplo:

Si $\Omega = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ es el conjunto de datos que viene particionado como

$$\{Y_1, Y_2, Y_3, Y_4, Y_5\} = \{\{1, 3\}, \{2, 5\}, \{4, 7, 9, 10\}, \{8\}, \{6\}\}$$

Si P es cierto conjunto de atributos que induce la siguiente partición sobre Ω:

$$\Omega/_\wp = \{\{1, 2, 3\}, \{4\}, \{5, 8\}, \{6\}, \{7\}, \{9, 10\}\}$$

Entonces:

- $W_1 = \dfrac{0}{3} = 0, \quad W_2 = \dfrac{0}{3+2} = 0, \quad W_3 = \dfrac{1+1+2}{1+1+2} = 1, \quad W_4 = \dfrac{0}{2} = 0,$
  $W_5 = \dfrac{1}{1} = 1$

- $W_C = \dfrac{1+1+2+1}{3+3+2+1+1+2+2+1} = \dfrac{1}{3}$
- $W_e = \dfrac{1+1+2+1}{10} = \dfrac{1}{2}$
- $\widetilde{W} = \dfrac{2}{10}(0) + \dfrac{2}{10}(0) + \dfrac{4}{10}(1) + \dfrac{1}{10}(0) + \dfrac{1}{10}(1) = \dfrac{1}{2}$
- $\widetilde{\widetilde{W}} = \dfrac{max\{2,1\}+1+max\{1,1\}+1+1+2}{10} = \dfrac{4}{5}$.

Como puede observarse, $\widetilde{\widetilde{W}}$ es mayor que W_e y que W_C y toma en cuenta, por ejemplo, qué tan "contenida" está la clase $\{1,2,3\}$ en la categoría $\{1,3\}$. Puede observarse además que el conteo realizado en el numerador de $\widetilde{\widetilde{W}}$ incluye a las sumas inferiores de cada categoría.

Volviendo al ejemplo de la sección **1.2.2**, en el que los valores de W_C y W_e calculados para la tabla respecto al conjunto de atributos $P = \{x^2, x^4\}$, son $W_C = \dfrac{1}{17}$ y $W_e = \dfrac{1}{9}$, al calcular estos nuevos índices resutan:

$\widetilde{W} = \dfrac{5}{9}\dfrac{1}{17} + \dfrac{4}{9}(0) = \dfrac{5}{153}$

$\widetilde{\widetilde{W}} = \dfrac{max\{2,1\}+max\{1,1\}+max\{1,2\}+1}{9} = \dfrac{2}{3}$.

Este último valor refleja mejor el grado de dependencia parcial de la clasificación que viene con la tabla de datos, de los dos atributos x^1 y x^2.

La implementación y valoración de los dos índices $\widetilde{W}$ y $\widetilde{\widetilde{W}}$ es un trabajo que queda abierto y sería interesante retomarlo, una vez concluido el presente trabajo de tesis.

1.2.4 Reducciones de atributos

Dado $P \subseteq X$, un atributo x^j de P se dice *redundante*, si al eliminarlo de P no cambia la partición inducida, es decir, si $\Omega/(\wp - \{x^j\}) = \Omega/\wp$.

P es un conjunto *independiente* de atributos si en P no se encuentran atributos redundantes, por lo que al quitar cualquier atributo de un conjunto independiente se produce una partición más gruesa de Ω.

Una *reducción* del conjunto de atributos $X = \{x^1, ..., x^p\}$ es cualquier subconjunto P de X que sea independiente y genere la misma partición que todos los atributos juntos. Es común que a la tabla de datos se le puedan encontrar varias reducciones distintas. El *núcleo* se define como la intersección de todas sus reducciones.

En el ejemplo presentado en la sección **1.2.2**, sus reducciones son $\{x^1, x^2, x^4\}$, $\{x^1, x^3, x^4\}$ y $\{x^2, x^3, x^4\}$, pues en los tres casos se induce la misma partición que el conjunto completo $\{x^1, x^2, x^3, x^4\}$ de atributos; el núcleo es $\{x^1, x^2, x^4\} \cap \{x^1, x^3, x^4\} \cap \{x^2, x^3, x^4\} = \{x^4\}$.

Los índices de calidad presentados en la sección **1.2.1** proporcionan una manera de cuantificar la medida en que la variable categórica (la clasificación dada *a priori*) es explicada por las variables presentes en el conjunto P.

La idea de una implementación de conjuntos aproximados es intentar reducir el número de atributos, manteniendo los índices de calidad y exactitud en un nivel lo más alto posible. Estas reducciones, además de requerir cálculos muy exhaustivos en caso de tratar tablas muy grandes, no siempre están asociadas al óptimo de esos índices. Para evitar los cálculos exhaustivos y en busca del máximo de esos criterios, el presente trabajo de investigación lo aborda mediante algoritmos genéticos.

En el capítulo 2 se exponen los conceptos básicos de los algoritmos genéticos y algunas consideraciones teóricas y algorítmicas referentes a la convergencia de tales algoritmos. En el capítulo 3 se expone el algoritmo genético específico para abordar el problema de cálculo de conjuntos aproximados. En el capítulo 4 se presentan los resultados numéricos de los experimentos realizados con diversas tablas de datos .

La tabla inicial de datos puede verse como un conjunto de datos de entrenamiento para formar un sistema de discriminación que culmina con la obtención de una o varias reducciones del conjunto de atributos. De esta manera, si elegimos una de las reducciones, podemos omitir columnas redundantes de la tabla. De igual manera, podemos eliminar de la tabla las

filas redundantes, convirtiéndose en una tabla sintetizada que puede verse como la base de conocimiento a partir de la cual se genera un conjunto de reglas para un sistema experto. Esto entra en el contexto de las técnicas de adquisición de conocimiento a partir de ejemplos (técnicas inductivas) ([33]).

Al introducir objetos nuevos se miden los valores de las variables presentes en una determinada reducción y, de acuerdo con estos valores, lo que se persigue es la clasificación de esos objetos en alguna de las categorías. Este último objetivo es compartido por algunos métodos de discriminación, ciertos modelos de redes neuronales y los árboles de decisión. Por la conexión que tienen con el tema, a continuación se presenta un resumen de esos modelos.

1.3 El análisis discriminante

Los métodos de discriminación se presentan en el análisis de datos cuando se quieren tener criterios para clasificar objetos en una de k categorías posibles, a partir de p variables explicativas o predictoras, generalmente numéricas. Los datos de partida de los algoritmos de discriminación consiten en n observaciones de las variables explicativas y cada una de éstas clasificada en alguna de las k clases de la variable categórica (variable a explicar). En [30] y [29] se citan dos objetivos complementarios, clásicos del análisis discriminante:

1. El *descriptivo*, que busca las combinaciones lineales más apropiadas de las va- riables para separar lo mejor posible las k categorías y dar una representación geométrica de esa separación. Este objetivo corresponde a los métodos descriptivos del análisis de datos, tales como el Análisis Factorial Discriminante, el Análisis en Componentes Principales y el Análisis de Correspondencias ([29], [35]).

2. El *decisional*, que pretende decidir en cuál categoría se ubican los nuevos individuos de acuerdo con los valores que muestra en las variables explicativas. En este objetivo se ubican los métodos probabilísticos.

1.3.1 Métodos geométricos

En el caso de que las variables explicativas sean numéricas, se aplica un *análisis factorial discriminante* (AFD). Este método consiste en la búsqueda de nuevas varia- bles sintéticas, que son combinaciones lineales de las p variables, y corresponden a las direcciones en $\mathbb{R}^p$ que separan lo mejor posible los k grupos de observaciones. Geométricamente, el método consiste en buscar ejes en $\mathbb{R}^p$ tales que los centros de gravedad de los grupos de datos se proyecten en tales ejes con dispersión máxima (dispersión inter-clases), a la vez que el grado de cohesión interna de los grupos sea alta (dispersión intra-clase mínima).

Para discriminar la clase a la que pertenece un individuo x, la asignación se realiza determinando el grupo cuyo centro de gravedad sté a una distancia mínima de x, usando una métrica proporcionada por el análisis discriminante.

Para un estudio detallado del tema, puede consultarse [30], o la recopilación de G. Celeux en [4], especialmente el capítulo 5, de A. Kobilinsky y el capítulo 1, de G. Saporta.

1.3.2 Métodos probabilísticos

Los métodos probabilísticos se basan en la regla bayesiana para calcular las proba- bilidades condicionales. En el caso de variables numéricas, el planteamiento del método es como sigue:

Si $p_1, p_2, ..., p_k$ son las proporciones de cada clase respecto al total del conjunto de individuos. Si $P(x|Y_i)$ es la probabilidad de que un individuo x pertenezca a la clase Y_i, entonces, según la regla de Bayes, la probabilidad (condicional) $P(Y_i|x)$ de que x provenga de la clase Y_i es

$$P(Y_i|x) = \frac{p_i P(x|Y_i)}{\sum_{j=1}^{k} p_j P(x|Y_j)}$$

La discriminación consiste en asignar x a la clase Y_i tal que $P(Y_i|x)$ sea máxima, lo cual se logra cuando $p_i P(x|Y_i)$ es máxima.

Para realizar los cálculos anteriores se necesita estimar las probabilidades $P(x|Y_j)$, para $j = 1, ..., k$ y uno de los métodos es el de los vecinos más cercanos ([35]). Este consiste en escoger un número K y se determinan los K vecinos de x en Ω más cercanos. Luego se determina n_j como el número de esos K vecinos que pertenecen a la clase j. Se estima $P(x|Y_j) = \frac{n_j}{K}$.

1.3.3 El método Disqual

En [30], G. Saporta presenta un método que llama *Disqual* (discriminación cualitativa) para hacer discriminación cuando las p variables explicativas x^1, x^2,...,x^p son cualitativas. A grandes rasgos, Disqual consiste en lo siguiente:

Si x^1, x^2,...,x^p , tienen respectivamente m_1, m_2,...,m_p modalidades, se siguen los siguientes pasos:

- *Paso 1:* Se efectúa un *análisis de correspondencias múltiples* (ACM) de las variables x^1, x^2,...,x^p de la tabla disyuntiva $(X_1|X_2|...|X_p)$ (cada X_i es de tamaño $n \times m_j$), obteniéndose m ejes factoriales.

- *Paso 2:* Cada objeto ω_1,... ω_n, proyectado sobre cada uno de los m ejes factoriales, tiene en éstos coordenadas z^1, z^2,...,z^m. Se reemplazan las p variables cualitativas por sus m coordenadas en los ejes factoriales. Se tiene ahora una tabla con las variables explicativas cuantitativas z^1, z^2,...,z^m.

- *Paso 3:* Se realiza una análisis factorial discriminante sobre las m variables cuantitativas obtenidas en el paso anterior.

- *Paso 4:* Para la discriminación de la pertenencia de nuevos objetos a cada una de las clases, se usa un *factor discriminante* d, que es una combinación lineal de las z^j que a su vez son combinaciones lineales de las indicatrices de las x^i, quedando al final d expresado como una combinación lineal de las indicatrices de las x^i. Se reduce así el problema a atribuir a cada categoría de cada variable un puntaje d que es la suma de

los puntajes obtenidos en las categorías de las p variables. En [31] se explica con algún detalle la formulación de la función de puntaje.

1.4 Las redes neuronales

Las redes neuronales constituyen otra de las alternativas para hacer discriminación.

Una *red neuronal* es un modelo computacional inspirado en el funcionamiento en paralelo conocido o teorizado del cerebro humano. Esencialmente, la red es un grafo dirigido de nodos interconectados. A los nodos se les llama formalmente *neuronas* y a las conexiones se les llama *sinapsis*. Las sinapsis tienen asociados pesos llamados *pesos sinápticos* que indican qué tan activadas están las conexiones en un momento dado.

Cada neurona en la red recibe N entradas $x_1, x_2, ..., x_N$ a través de N sinapsis. En un grupo de neuronas, la neurona j-ésima realiza una suma ponderada de los valores de entrada:

$$h_j = \sum_{i=1}^{N} w_{ji} x_i + \theta_j$$

donde:

w_{ji} es el peso ponderado de la conexión i-ésima que llega a la neurona j.

θ_j es un umbral o parámetro de control.

La *salida* de la neurona se calcula mediante una *función de activación* f que determina el estado de la neurona:

$$\sigma_j = f(h_j)$$

Esta función f puede ser de varios tipos: binaria, sigmoide, escalera, etc. La figura 1.1 ilustra la forma gráfica que usualmente tienen las funciones de activación de tipo sigmoide.

Figure 1.1: *Función de activación de tipo sigmoide.*

Dependiendo de la manera en que estén agrupadas las neuronas y del tipo de cone- xiones entre esos grupos, así será la arquitectura de la red; hay, por ejemplo, redes de dos capas, redes multicapas y redes totalmente conectadas. En la figura 1.2 cada neurona es representada por un círculo. Las flechas indican las conexiones direccionadas de una neurona a otra. Cada nodo tiene una sola salida que puede ser trasmitida a varios nodos.

Figure 1.2: *Ejemplo de red neuronal con tres capas.*

Los modelos de redes neuronales dependen de la arquitectura de red que se esté usando, de las características de los nodos y de las *reglas de entrenamiento* o *aprendizaje* (supervisado

o no supervisado). En algunos tipos de redes el aprendizaje ocurre cuando los pesos se van adaptando de acuerdo con los datos o *patrones* de entrenamiento que van ingresando; estas reglas especifican una manera de inicializar los pesos de la red y cómo éstos irán cambiando. Hay otras redes, como la de Hopfield y la de Hamming, que son generalmente usadas con pesos fijos y lo que se adapta es la salida de los nodos.

El *aprendizaje supervisado*, que es lo que nos interesa, se realiza cuando al pasar un patrón de entrenamiento por la red los pesos son adaptados, una vez que se compara la salida final con con la salida esperada y conocida del patrón. El conjunto de patrones es pasado sucesivamente por la red una y otra vez a lo largo de cierto número de iteraciones, hasta lograr estabilidad. Es en esta etapa en la que se dice que la red está entrenada y ahora, servirá para clasificar o reconocer nuevos patrones.

1.4.1 Método de retropropagación del gradiente

Este método es aplicable a una red con una capa de entrada, una capa de salida y una o más capas ocultas, como la de la figura 1.2, con conexiones sólo entre neuronas de capas consecutivas. Además se supone que en cada neurona la función de activación es una función sigmoide derivable.

Para entrenar la red se realiza un aprendizaje supervisado, como sigue: Se modifican los pesos de la red introduciendo cada patrón de entrenamiento y se compara la salida obtenida con la deseada (o conocida). Primero se modifican los pesos de las conexiones que llegan a la capa de salida y luego, trabajando hacia atrás, hacia la capa de entrada, se van modificando los pesos entre capas sucesivas con el fin de reducir los errores en cada nivel.

En un momento t del entrenamiento, el error en la neurona j de la capa de salida ($j = 1, ..., M$), es $d_j - s_j$, siendo d_j la salida deseada y s_j la salida obtenida. El error cuadrático

global es

$$E_t = \frac{1}{2}\sum_{j=1}^{M}(d_j - s_j)^2$$

. Al pretender minimizar este error, se deriva E_t respecto a los pesos y, cambiando los pesos en la dirección contraria al gradiente[2], se obtiene la ley de aprendizaje:

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_j x_i$$

donde:

$$\delta_j = \begin{cases} (d_j - s_j)f'(\sigma_j) & \text{si } j \text{ pertenece a la capa de salida,} \\ f'(\sigma_j)\sum_k \delta_k w_{kj} & \text{si } j \text{ pertenece a una capa oculta,} \\ & \text{recorriendo } k \text{ las neuronas de la capa siguiente.} \end{cases}$$

El problema de discriminación puede modelarse siguiendo una regla de aprendizaje supervisado ([34]) en una red organizada en varias capas. Lo normal sería crear una capa de salida con igual número de neuronas que de clases conocidas en que vienen clasificados los patrones de entrenamiento; cada una de estas neuronas representando a una de las clases que se quieren discriminar y su valor de salida sería 0 o 1. Otra posibilidad es que las neuronas de la capa de salida también tengan salidas binarias, pero que las clases a discriminar estén codificadas como tiras binarias, de manera que, por ejemplo, tres neuronas de este tipo en la capa de salida pueden representar $2^3 = 8$ clases distintas, simplificando aún más la estructura.

Para un estudio detallado de la discriminación usando modelos neuronales, puede consultarse a C. Chabanon y B. Dubuisson en [4], así como en [34]. En [20] y [21] se halla un desarrollo bastante extenso del tema de redes neuronales.

1.5 Árboles de decisión

Un *árbol de decisión* es un grafo dirigido asociado a un conjunto Ω de n mediciones de p atributos cualitativos x^1,...,x^p con Ω clasificado en k categorías o clases $Y_1, ..., Y_k$. Las hojas

[2]De la teoría de diferenciación de funciones de varias variables, es bien conocido que, para un estado cualquiera de éstas, la disminución más grande que tiene una función es en la dirección contraria al gradiente.

del árbol corresponden a las clases y en todos los demás nodos, incluyendo el nodo raíz, hay alguno de las atributos, y de aquí salen dos o más aristas, cada una correspondiente a las distintas modalidades que toma el atributo.

El árbol se construye a partir del conjunto de patrones y luego se utiliza con fines de discriminación de datos nuevos. Cuando un nuevo patrón sin clasificar es introducido al árbol de decisión desde arriba hacia abajo, en cada nodo no terminal se toma una decisión sobre la modalidad que presenta el atributo asociado a éste, para dirigirse en la dirección de la rama que corresponda. Finalmente, al llegar a una hoja se clasifica en la categoría contenida en ésta.

1.5.1 Algoritmo ID3

Uno de los algoritmos más difundidos para construir árboles de decisión es el llamado ID3, de Quinlan ([26]). Este algoritmo utiliza conceptos de la teoría de la información, tales como *entropía* y *ganancia*. A grandes rasgos, su construcción se realiza escogiendo, para cada nodo, el atributo que maximice la ganancia de información; de este nodo sale una rama por cada modalidad del atributo.

$\forall\ i = 1, ..., k$ sea $n_i = Card(Y_i)$, entonces $\dfrac{n_i}{n}$ es la probabilidad de que un patrón esté en la clase i.

Se define la *entropía global* $E(\Omega)$ como:

$$E(\Omega) = \sum_{i=1}^{k} -\frac{n_i}{n} log_2 \left(\frac{n_i}{n}\right)$$

Cuando el test es aplicado al atributo x^s, $(s = 1, ..., p)$, la tabla es dividida en J clases, donde J es el número de modalidades de la variable x^s. Si $n_{sj}(i)$ es el número de patrones en el subconjunto j que pertenece a la clase Y_i de la partición conocida de Ω ($j = 1, ..., J$, $i = 1, ..., k$) . Si además n_{sj} denota el número de individuos en la clase j (de la partición

inducida por x^s), es decir, $n_{sj} = \sum_{i=1}^{k} n_{sj}(i)$, entonces, la *entropía de la variable* x^s se calcula como:

$$E(\Omega, x^s) = \sum_{j=1}^{J} \frac{n_{sj}}{n} \cdot E(\Omega, x^s, j)$$

donde

$$E(\Omega, x^s, j) = \sum_{i=1}^{k} -\frac{n_{sj}(i)}{n_{sj}} \log_2 \left(\frac{n_{sj}(i)}{n_{sj}} \right)$$

Además, se define la *ganancia de información* del atributo x^s como:

$$Gan(x^s) \ = \ E(\Omega) - E(\Omega, x^s)$$

El siguiente es el algoritmo ID3, escrito en forma recursiva (tal y como fue programado en [2]), aplicado a una tabla de datos T.

ID3(T):

(a) Formar un nodo para la tabla T.

(b) Comprobar si T es homogenea; es decir, si todas sus filas están en la misma clase. Si T es homogenea, asociar al nodo formado la clase a la que pertenecen todas las filas de T y termina el programa.

(c) Si T no es homogenea, escoger la variable x^{s_0} que "abre", donde
$Gan(x^{s_0}) = Max\{Gan(x^s), \quad s = 1, ..., p\}$.
Asociar al nodo la variable x^{s_0}.

(d) Si $\{T_1, T_2, ..., T_J\}$ son las subtablas que se han obtenido de la partición de T inducida por la variable x^{s_0}, asociado a cada una de estas subtablas y a su correspondiente valor de x^{s_0}, formar un arco que salga del nodo.

(e) Para $j = 1, ..., J$ aplicar ID3(T_j)

(f) Fin.

El algoritmo ID3 y otras variantes fueron programados usando Mathematica 3.0 y el programa, junto con varios árboles de decisión generados a partir de distintas tablas de datos, se encuentran en [2].

En [12] se presenta una modificación del ID3 que hace uso de conjuntos aproximados maximizando el criterio $\widetilde{\widetilde{W}}$ planteado en la sección **1.2.3**, tomando P como un conjunto de un solo atributo. En [40] y en [15] se relacionan la construcción de un árbol de decisión con la de una red neuronal.

1.5.2 Ejemplo

La tabla 1.1 fue tomada de [1] y se refiere a un grupo de once estudiantes a los que se les ha medido cinco variables: Experiencia,título, nacionalidad, dirección y entrevista. Además, cada uno de los estudiantes está ubicado en una de siete clases.

Nombre	**Experiencia**	**Título**	**Nacionalidad**	**Dirección**	**Entrevista**	**Clase**
Sandra	2	3	1	2	3	**1**
Lucía	3	4	1	3	4	**2**
Ana	3	4	0	0	4	**3**
Julio	2	3	1	2	3	**1**
Mario	3	4	1	4	1	**4**
Jorge	3	4	1	4	1	**4**
Errol	2	3	1	2	3	**1**
Manuel	3	4	0	3	4	**5**
Edgar	3	3	1	2	3	**6**
Walter	2	2	1	2	3	**7**
Silvia	3	3	1	2	3	**6**

Table 1.1: *Tabla de 11 estudiantes clasificados en 7 clases, a los que se les ha medido 5 atributos cualitativos.*

En la figura 1.3 aparece el árbol generado para esta tabla mediante ID3. Como puede notarse, una de las variables explicativas no se usa del todo en el árbol y por lo tanto, al generar el árbol se eliminan variables innecesarias. En algunas de las corridas del programa presentadas en [2], las variables presentes en el árbol corresponden a una reducción del

conjunto de atributos, por lo que la generación de árboles de decisión mediante un mecanismo como éste viene a ser otra herramienta para la obtención de conjuntos aproximados y para la eliminación de información superflua.

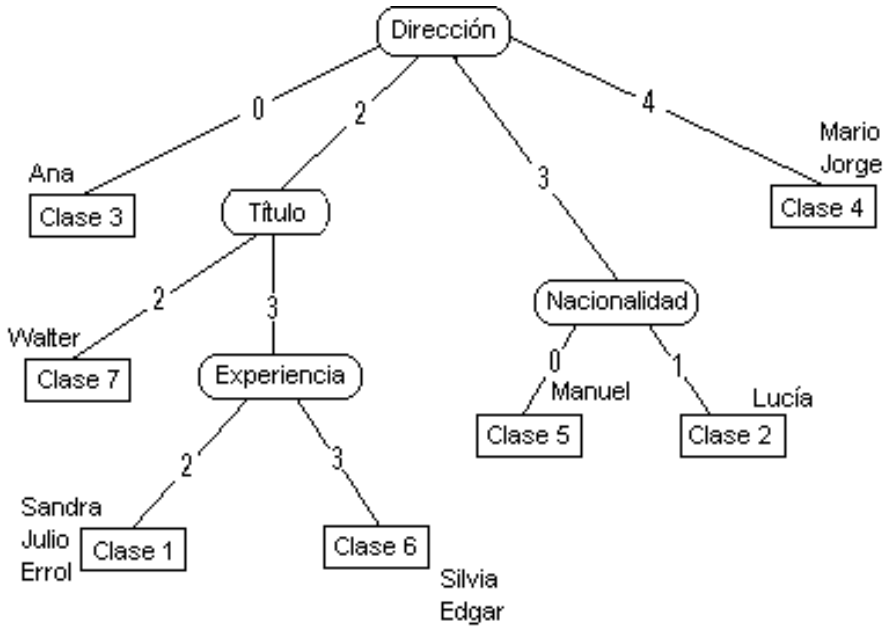

Figure 1.3: *Arbol de decisión generado por ID3 .*

Chapter 2

Algoritmos genéticos

Un algoritmos genético (AG) es un paradigma inspirado en el comportamiento y composición genética conocidos de los seres vivos, tales como la selección natural, la mutación y el cruzamiento, con el fin de representar y resolver ciertos problemas de optimización susceptibles de ser enfocados bajo estos conceptos y para los que los métodos de optimización tradicionales, o bien no son aplicables, o bien no producen soluciones satisfactorias. En muchos de estos casos los AG han probado ser heurísticas muy robustas para localizar óptimos globales. Para esto último, actualmente existen otras técnicas que también se utilizan para buscar óptimos globales, tales como *búsqueda tabú* y *sobrecalentamiento simulado*([6]), con buenos resultados en problemas de Estadística y Análisis de Datos (ver [36], [37], [38] y [39]).

2.1 Conceptos básicos

Un algoritmo genético tiene un carácter iterativo y actúa sobre una población de cierto conjunto individuos que son una representación formal de las soluciones factibles del problema. En cada iteración del algoritmo, sujeto a cierta probabilidad y a la evaluación de cada individuo en la función a optimizar, se realizan operaciones de cruzamiento, mutación y selección, para obtener una nueva población que entendemos como pertenecientes a una nueva generación.

Típicamente, un problema de optimización que se modele mediante un AG debe tener las siguientes características:

A. Una *representación genética* de las soluciones factibles del problema debe ser posible. A estas representaciones o a las soluciones factibles asociadas se les suele dar el nombre de *individuos*. Usualmente la representación genética de las soluciones factibles consisten en un cromosoma (representación *aploide*) o varios (representación *diploide*). Las tiras cromosómicas contienen L *genes*. En el algoritmo genético se trabaja iterativamente con una *población* de individuos que irá cambiando de una generación a la siguiente. Una *generación* corresponde a la población que se tenga en una iteración particular. La población inicial se genera a partir de algún criterio y usualmente es al azar.

B. Una *función evaluadora* f positiva para medir el grado de adaptabilidad de los individuos. A f se le llama *función de adecuación* (o *fitness en inglés*). Esta función corresponde, en la analogía biológica, a la medición de la adaptabilidad del individuo al medio ambiente, y en el caso de la optimización matemática, corresponde a la función objetivo o una modificación de ésta. En caso de que f tome valores negativos o que sea un problema de minimización el que se esté resolviendo se hacen las traslaciones o modificaciones en f que sean necesarias para trabajar con otra función que sí sea positiva, o para adaptarlo a un problema de maximización.

C. El problema de optimización debe permitir la definición de unos *operadores genéticos* que serán aplicados a la población que se tiene en cada iteración, para pasar de una generación a otra. Los operadores genéticos más utilizados son el *cruzamiento*, la *mutación* y la *selección*.

D. Se deben especificar los *parámetros* que intervienen en el algoritmo, tales como tamaño de la población, probabilidad de cruzamiento, probabilidad de mutación, el tipo de reproducción, etc.

En la generación t del algoritmo genético se tiene una población Q_t de N individuos. A esta población se le aplica un *proceso de selección* basado en el grado de adaptación al medio,

de ma- nera que entre más alto sea el valor de f medido a un individuo, mayor probabilidad tendrá de ser seleccionado. Se forma así una *población intermedia* PI. Teniendo como restricción los parámetros del algoritmo, se aplican los operadores genéticos de mutación y cruzamiento a la población seleccionada para obtener una nueva población Q_{t+1}; de esta manera se ha pasado de una generación a otra que contiene, muy probablemente, la información genética de los individuos mejor adaptados de la generación anterior. Este proceso de paso entre generaciones se repite sucesivamente hasta llegar a un estado que sugiera que ha ocurrido en cierto sentido convergencia o cuando se ha llegado a un máximo de generaciones. En el siguiente seudo-código de un AG se toma como criterio de finalización el haber llegado a un máximo número de generaciones (MaxNumGen).

Programa evolutivo:

$t \longleftarrow 0$

Generar Q_0 (población Inicial)

Evaluar f en Q_0

Mientras $t \leq MaxNumGen$:

Seleccionar Población Intermedia PI de Q_t

Hacer Cruce entre individuos de QI

Hacer Mutaciones en PI

$Q_{t+1} \longleftarrow PI$

Evaluar f en Q_{t+1}.

$t \longleftarrow t + 1$

Fin.

2.1.1 Mecanismo de selección de individuos.

Para la selección de los individuos que van a formar la población intermedia se simula un mecanismo aleatorio de *ruleta* (ver 2.1) en la que a cada individuo se le asigna una porción

de ésta, proporcional a su evaluación en la función de adecuación. Se hace "girar" la ruleta y se elige al individuo que termine señalado por ésta. De esta manera, siguiendo la analogía de la selección natural, los individuos mejor adaptados tienen una mayor probabilidad de ser seleccionados y, por tanto, de heredar su información genética de una generación a la siguiente. En [9] se hace una variante del mecanismo de la ruleta para asegurar una mayor probabilidad de que los individuos mejor evaluados sean seleccionados. Esta variante consiste en asignar un mayor peso en la ruleta del que ya tiene respecto a los demás, al individuo que resulte mejor evaluado. Esto le da una mayor probabilidad de sobrevivencia.

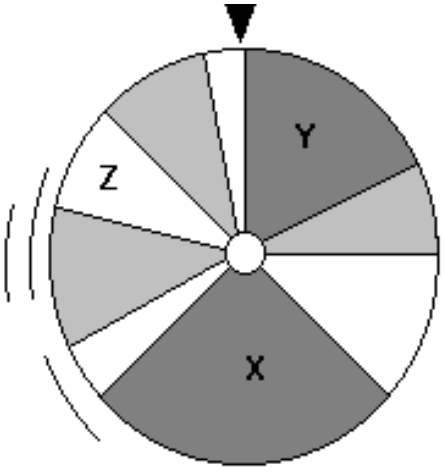

Figure 2.1: *Simulación de la ruleta en proporción a la evaluación de cada individuo. X y Y tienen mayor probabilidad de resultar favorecidos que Z.*

2.1.2 El operador de mutación:

Dado un individuo representado por un solo cromosoma compuesto por L genes, una mutación consiste en un cambio que, con cierta probabilidad le ocurre a alguno de sus genes, cambiando su valor por alguno del resto de valores permitidos (ver Fig. 2.3). En el caso de que la representación sea binaria una mutación consiste en elegir al azar alguno de los genes y cambiar su valor por el complementario.

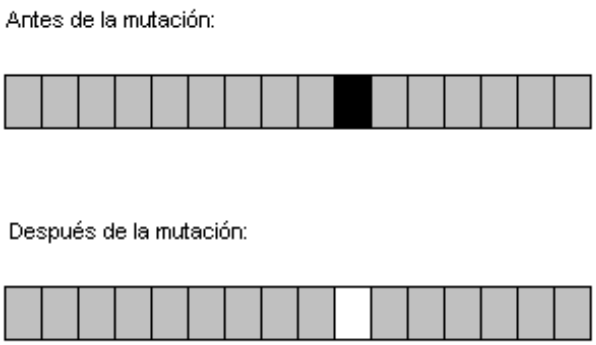

Figure 2.2: *Mutación ocurrida en un gen específico del cromosoma. Su contenido cambia por otro de los valores permitidos.*

2.1.3 El operador de cruzamiento:

La reproducción ocurre con el cruzamiento entre individuos. Una de las alternativas para realizar el cruzamiento entre dos individuos (padres) se ilustra en la figura 2.3 y consiste en elegir al azar dos puntos de corte y, de acuerdo con una probabilidad de cruzamiento, intercambiar los bloques separados por esos cortes para formar un par de hijos. Esto asegura que los hijos hereden bloques completos de información genética de ambos padres.

Figure 2.3: *Cruzamiento de dos individuos y formación de dos hijos con información genética de ambos padres.*

Otra forma de realizar el cruzamiento es escogiendo múltiples puntos de corte, pero la forma más común de realizarla es eligiendo un solo punto (y esta fue la forma escogida en el presente trabajo) y luego intercambiar los bloques correspondientes. En la literatura de algorimos genéticos se mencionan otros operadores especializados que en problemas específicos pueden contribuir a un mejoramiento significativo del AG (ver , [16] y [10]); entre ellos, el operador de inserción, dominancia, inversiones, reordenamiento, la formación de nichos, y otros.

2.2 Los esquemas

Dada una representación de soluciones factibles de un problema de optimización que ha sido mode- lado mediante un algoritmo genético, podemos suponer, para efecto de análisis, que tal representación es binaria. Dentro de este contexto, un *esquema* es una tira de símbolos que viene a ser una descripción de un conjunto de tiras iguales en ciertas posiciones; las posiciones que resulten variables en tal conjunto pueden representarse con un asterisco.

Así, puede definirse un esquema como un arreglo $S = x_1x_2...x_L$, donde $x_i \in \{0, 1, *\}$. Una tira binaria específica, $b = b_1b_2...b_L$, es una *instancia* del esquema S si cumple que $\forall i = 1, ..., N$: $b_i \in \{0, 1\}$, en caso de que $x_i = *$, o $b_i = x_i$, en el caso de que $x_i = 0$ o $x_i = 1$. Por ejemplo, en el esquema $S = 1 * 010*$ sus posiciones 1, 3, 4 y 5 son fijas, por lo que representa al conjunto de instancias $\{100100, 100101, 110100, 110101\}$. Según [10], la importancia de los esquemas en los algoritmos genéticos radica en que las poblaciones de tamaño N manejan una cantidad del orden de 3^N esquemas, por lo que la cantidad de soluciones factibles representada en la población, en el fondo es muy grande respecto al tamaño de ésta.

Dado un esquema S, el *orden* de S, denotado $o(S)$ es el número de posiciones fijas que tiene; por ejemplo, $o(101 * *) = 3$. La *longitud* de S, $l(S)$ es la cantidad de posiciones que hay entre la primera y la última posición fija en S; si no hay posiciones fijas en S, se define $l(S) = 0$; por ejemplo, $l(*101 * 1) = 5$, $l(* * *10*) = 2$.

Sea $f : F \longrightarrow \mathbb{R}_+^*$ la función evaluadora a maximizar, donde $F = \{0,1\}^L$ es el espacio de soluciones factibles y $\mathbb{R}_+^*$ es el conjunto de números reales positivos; sea Q una población de tamaño K. Se define la *utilidad* de un esquema S respecto a la población Q, denotada $u(Q,S)$, como el promedio de f en el conjunto de instancias de S presentes en la población Q. Esta utilidad mide el grado de importancia de S en la optimización de f. Formalmente,

$$u(Q,S) = \frac{1}{|S \cap Q|} \sum_{\omega \in S \cap Q} f(\omega)$$

La siguiente conjetura fue propuesta por Goldberg ([10]) y se refiere a la esperanza de sobrevivencia de un esquema, de una generación a la siguiente.

Conjetura de los esquemas: Sean Q_{t+1} y Q_t, $t \in \mathbb{N}$, las poblaciones en las generaciones $t+1$ y t, respectivamente, obtenidas por el algoritmo genético. Entonces:

$$E(S \cap Q_{t+1}) \geq |S \cap Q_t| \frac{u(Q_t,S)}{u(Q_t,F)} (1 - c(Q_t,S) - m(Q_t,S))$$

donde:

E : denota la esperanza matemática,

$c(Q_t,S)$: denota la probabilidad de que una instancia de S sea destruida por cruzamiento y

$m(Q_t,S)$: denota la probabilidad de que una instancia de S sea destruida por mutación.

En [10] se afirma además que

$$\begin{aligned} c(Q_t,S) &= p_c \frac{l(S)}{N-1} \\ m(Q_t,S) &= o(S) p_m \end{aligned}$$

donde p_c y p_m son, respectivamente, los parámetros de probabilidad de cruzamiento y de mutación.

La desigualdad de la conjetura de los esquemas toma la forma:

$$E(S \cap Q_{t+1}) \geq |S \cap Q_t| \frac{u(Q_t,S)}{u(Q_t,F)} (1 - p_c \frac{l(S)}{N-1} - o(S) p_m \ )$$

De acuerdo con esta última desigualdad, los esquemas de corta longitud, de orden pequeño, y con una buena proporción de utilidad respecto a la población tendrían una esperanza alta de pasar de una generación a la siguiente. Estos esquemas tienen tendencia a multiplicarse considerablemente en las poblaciones sucesivas pues son difíciles de romper por mutaciones y cruzamientos. Se estima que la convergencia de los AG's a un óptimo útil, reposa en la habilidad de combinar estos bloques mediante cruzamientos (hipótesis de los *bloques de construcción*, ver [16], [10]).

2.3 Convergencia de los AG's

La demostración de convergencia de los algoritmos genéticos hace uso de las cadenas de Markov. La propiedad fundamental de un sistema markoviano es la ausencia de memoria: la evolución futura del sistema no depende más que del instante presente y no de su evolución pasada. La secuencia de poblaciones producidas por el algoritmo genético es considerada como una cadena de Markov homogénea ([16],[28]), en la que cada estado corresponde a una población particular.

2.3.1 Cadenas de Markov finitas

Una *cadena de Markov* es un proceso estocástico $\{X_t, \quad t = 0, 1, 2, ...\}$ (ver [27]) y consiste en una sucesión de variables aleatorias que toman un número finito o contable de *estados* posibles. En el caso finito, asumimos que el espacio de estados es $I\!E = \{1, ..., n\}$. La expresión $X_t = i$, indica que el proceso se encuentra en el estado i en el tiempo (o etapa) t.

En cada etapa t de la cadena de Markov hay una matriz cuadrada $P = (p_{ij}(t))$ de tamaño $n \times n$ llamada *matriz de transición*, donde $p_{ij}(t)$ es la probabilidad de pasar del estado i al estado j, sin que dependa de los estados anteriores. Como las probabilidades son no negativas y en cada etapa debe ocurrir alguna transición de un estado i a algún estado j (j puede ser igual a i), se tiene que $p_{ij}(t) \geq 0$ y $\sum_{j=1}^{n} p_{ij}(t) = 1$.

Si la matriz P es la misma a lo largo de todo el proceso, la cadena de Markov se dice

homogénea; en adelante, y para nuestro propósito, se supondrá que la cadena es homogénea y de espacio de estados finito.

Dada una distribución inicial $q^{(0)}$, ésta consiste en una matriz fila, tal que $\forall j \in \{1, ..., n\}$, $q_j^{(0)} = Prob(X_0 = j)$. La distribución $q^{(t)}$ de la cadena en la etapa t, $q_j^{(t)} = Prob(X_t = j)$, $\forall j \in \{1, ..., n\}$, es $q^{(t)} = q^{(0)}P^t$ (P^t denota la potencia t-ésima de matriz P). De acuerdo con esto último, una cadena de Markov homogénea está enteramente determinada por su distribución inicial y por la matriz de transición P. La siguiente terminología de matrices es importante en el análisis de las cadenas de Markov homogéneas; en todos los casos la matriz $A = (a_{ij})$ que se menciona se supone cuadrada, de tamaño $n \times n$.

- A es *no negativa*, si $\forall i, j \;\; a_{ij} \geq 0$, con $0 \leq i, j \leq n$.
- A es *positiva*, si $\forall i, j \;\; a_{ij} > 0$, con $0 \leq i, j \leq n$.

Si A es no negativa,

- A es *primitiva*, si $\exists k \in I\!N$, tal que A^k es positiva.
- A es *reducible*, si mediante el intercambio de filas y columnas, puede ser llevada a la forma

$$A = \begin{pmatrix} C & 0 \\ R & T \end{pmatrix}$$

donde C y T son matrices cuadradas.

- A es *estocástica*, si para toda fila i, $\sum_{j=1}^{n} a_{ij} = 1$.

Si A es estocástica,

- A es *positiva por columna*, si en cada columna de A hay al menos una entrada estrictamente positiva.
- A es *estable*, si todas sus filas son iguales.

De acuerdo con las definiciones anteriores, la matriz de transición de una cadena de Markov homogénea debe ser estocástica.

Los siguientes resultados de matrices de transición serán necesarios en el modelaje de AG's me- diante cadenas de Markov.

Proposición 1:

i) Si X, Y son matrices estocásticas de tamaño $n \times n$, entonces XY es también estocástica.

ii) Si $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$ son matrices estocásticas cualesquiera de tamaño $n \times n$; si, además, B es positiva y C es positiva por columna, entonces ABC es positiva.

Demostración:

i) Para toda fila i de la matriz XY,

$$\sum_{j=1}^{n}(XY)_{ij} = \sum_{j=1}^{n}\sum_{k=1}^{n} x_{ik}y_{kj} = \sum_{k=1}^{n}\sum_{j=1}^{n} x_{ik}y_{kj} = \sum_{k=1}^{n} x_{ik}\sum_{j=1}^{n} y_{kj} = \sum_{k=1}^{n} x_{ik} \cdot 1 = 1,$$

Por lo tanto, XY es estocástica.

ii) Como es bien conocido, $ABC = (AB)C$. Ahora bien, para toda entrada ij de AB,

$$(AB)_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} > 0$$

pues debido a que A es estocástica, $\sum_{k=1}^{n} a_{ik} = 1$, por lo que la fila i de A tiene al menos una entrada positiva; como B es positiva, esto conduce a la desigualdad anterior.
De igual forma, como AB es positiva y C es positiva por columna, se concluye que $\forall i, j$,

$$(ABC)_{ij} = ((AB)C)_{ij} = \sum_{k=1}^{n}(AB)_{ik}c_{kj} > 0$$

pues $(AB)_{ik} > 0\ \forall i, k$ y en la columna j de C hay al menos una entrada no nula. Por lo tanto, ABC es positiva. □

El siguiente es considerado un teorema fundamental en la teoría de las cadenas de Markov.

Teorema 1: Sea P una matriz estocástica primitiva, $n \times n$. Entonces

$$\lim_{t\to\infty} P^t = \Pi = (1, 1, ..., 1)'\pi$$

donde Π es una matriz estable, positiva, con todas sus filas iguales a la distribución límite π, siendo $\pi = q^0 \lim_{t\to\infty} P^t = q^0\Pi$ con sus elementos no nulos. Además la distribución π es independiente de la distribución inicial q^0 (Nota: $(1, 1, ..., 1)'$ denota la transpuesta de $(1, 1, ..., 1)$).

Demostración: Como P es primitiva, $\exists t_0 \in \{1, 2, 3, ...\}$ tal que P^{t_0} es positiva.

Primer caso: Si $t_0 = 1$ entonces P es positiva.

Sea $\epsilon > 0$, escogido de tal manera que

$$\epsilon < Min\{p_{ij} : i, j = 1, ..., n\}$$

Sea x un vector columna cualquiera de entradas positivas. Se definen los valores m_0 y M_0 como:

$$m_0 = Min\{x_i : i = 1, ..., n\}$$

$$M_0 = Max\{x_i : i = 1, ..., n\}$$

Para el vector columna $Px = (\sum_{k=1}^n p_{1k}x_k, ..., \sum_{k=1}^n p_{nk}x_k)' > 0$ se definen los valores m_1 y M_1 como:

$$m_1 = Min\{(Px)_i : i = 1, ..., n\} = Min\{\sum_{k=1}^{n} p_{ik}x_k \quad : i = 1, ..., n\} \geq \sum_{k=1}^{n} p_{ik}m_0 = m_0$$

$$M_1 = Max\{(Px)_i : i = 1, ..., n\} = Max\{\sum_{k=1}^{n} p_{ik}x_x \quad : i = 1, ..., n\} \leq \sum_{k=1}^{n} p_{ik}M_0 = M_0$$

Por lo tanto, se cumple que $0 < m_0 \leq m_1$ y $0 < M_1 \leq M_0$.

Además, la siguiente desigualdad es válida:

$$M_1 - m_1 \leq (1 - 2\epsilon)(M_0 - m_0)$$

Para demostrarla, probemos primero que $M_1 \leq M_0 - \epsilon(M_0 - m_0)$:

$\forall\, i = 1, ..., n$:

$$\begin{aligned}
(Px)_i &= \sum_{k=1}^{n} p_{ik}x_k \\
&= p_{ij}m_0 + \sum_{k=1,k\neq j}^{n} p_{ik}x_k \quad \text{para algún } j \\
&\leq p_{ij}m_0 + \sum_{k=1,k\neq j}^{n} p_{ik}M_0 \\
&= p_{ij}m_0 + (\sum_{k=1,k\neq j}^{n} p_{ik})M_0 \\
&= p_{ij}m_0 + (1 - p_{ij})M_0 \\
&= M_0 - p_{ij}(M_0 - m_0)
\end{aligned}$$

Como cada componente de Px es menor o igual que $M_0 - p_{ij}(M_0 - m_0)$, se deduce que

$$M_1 < M_0 - \epsilon(M_0 - m_0)$$

pues

$$M_1 = Max\{(Px)_i : i = 1, ..., n\} < M_0 - \epsilon(M_0 - m_0)$$

De manera análoga, se comprueba que cada componente de Px es mayor o igual que $M_0 - \epsilon(M_0 - m_0)$, por lo que se concluye que

$$m_1 > m_0 - \epsilon(M_0 - m_0)$$

Usando las dos desigualdades :

$$M_1 < M_0 - \epsilon(M_0 - m_0) \quad \text{y} \quad m_1 > m_0 - \epsilon(M_0 - m_0)$$

se concluye que:

$$\begin{aligned} M_1 - m_1 &< M_0 - \epsilon(M_0 - m_0) - (m_0 - \epsilon(M_0 - m_0)) \\ &= M_0 - m_0 - 2\epsilon(M_0 - m_0) \\ &= (1 - 2\epsilon)(M_0 - m_0) \end{aligned}$$

Como P es estocástica, y por la manera de escoger ϵ,

$$\epsilon < Min\{p_{ij} : i, j = 1, ..., n\} \leq \frac{1}{2}$$

De aquí, se puede asegurar que $0 < 1 - 2\epsilon < 1$.

Para todo $k = 1, ..., n$, el k-ésimo vector columna de P^2 es Px, donde x es el k-ésimo vector columna de P. Se cumplen las desigualdades:

$$\begin{cases} 0 < m_0 \leq m_1 \leq M_1 \leq M_0 \ , \\ M_1 - m_1 \leq (1 - 2\epsilon)(M_0 - m_0) \end{cases}$$

Para la matriz P^3, el k-ésimo vector columna de P^3 es Px, donde x es el k-ésimo vector columna de P^2, se definen valores M_2 y m_2 tal y como se hizo antes con M_1 y m_1. Se cumple:

$$\begin{cases} 0 < m_0 \leq m_1 \leq m_2 \leq M_2 \leq M_1 \leq M_0 \ , \\ M_2 - m_2 \leq (1 - 2\epsilon)(M_1 - m_1) \leq (1 - 2\epsilon)^2(M_0 - m_0) \end{cases}$$

En general, el vector k-ésimo columna de P^{t+1} es Px, donde x es el k-ésimo vector columna de P^t y se definen los valores M_t y m_t, como se hizo antes con M_0, m_0,M_1, m_1, . . . y satisfacen las desigualdades:

$$\begin{cases} 0 < m_0 \leq m_1 \leq ... \leq m_t \leq M_t \leq ... \leq M_1 \leq M_0 \ , \\ M_t - m_t \leq (1 - 2\epsilon)^t(M_0 - m_0) \end{cases}$$

Esto justifica la formación de dos sucesiones de números $\{m_t\}$ y $\{M_t\}$, que dependen de k la primera de ellas monótona, creciente y acotada superiormente, y la segunda monótona, decreciente y acotada inferiormente, por lo que ambas convergen.

Además, para toda columna k y cualquiera que sea la fila i se tiene que $0 < m_t \leq (P^{t+1})_{ik} \leq M_t$; como $0 \leq \lim_{t\to\infty}(M_t - m_t) \leq \lim_{t\to\infty}(1-2\epsilon)^t(M_0 - m_0) = 0$, se concluye que $\{m_t\}$ y $\{M_t\}$ convergen a un mismo valor $\pi_k > 0$.

Finalmente, se concluye que

$$\lim_{t\to\infty}(P^t)_{ik} = \lim_{t\to\infty} m_{t-1} = \lim_{t\to\infty} M_{t-1} = \pi_k$$

para toda columna k de P^t y no depende de la fila i. Denotando Π a la matriz con todas sus filas iguales a $\pi = (\pi_1, \pi_2, ..., \pi_n)$, siendo π la distribución límite, con todas sus entradas positivas, se tiene que

$$\lim_{t\to\infty} P^t = \Pi$$

A continuación se prueba que la distribución límite π no depende de la distribución inicial $q^{(0)}$:

Como $\lim_{t\to\infty} P^t = \Pi$, entonces

$$\begin{aligned}
\lim_{t\to\infty} q^{(0)}P^t &= q^{(0)}\Pi \\
&= (q_1^{(0)}, q_1^{(0)}, ..., q_n^{(0)}) \begin{pmatrix} \pi_1 & \dots & \pi_n \\ \pi_1 & \dots & \pi_n \\ \vdots & \ddots & \vdots \\ \pi_1 & \dots & \pi_n \end{pmatrix} \\
&= (\pi_1 \sum_{i=1}^{n} q_i^{(0)}, ..., \pi_n \sum_{i=1}^{n} q_i^{(0)}) \\
&= (\pi_1, ..., \pi_n) \\
&= \pi
\end{aligned}$$

Para finalizar, un cálculo sencillo nos muestra que la matriz límite Π es igual a $(1, 1, ..., 1)'(\pi_1, \pi_2, ..., \pi_n)$.

Segundo caso: Si $t_0 > 1$, como la potencia P^{t_0} es estocástica positiva, aplicando el primer caso a P^{t_0} se tiene que $\lim_{t\to\infty}(P^{t_0})^t = \lim_{t\to\infty} P^{t_0 t} = \Pi$. Falta concluir que $\lim_{t\to\infty} P^t = \Pi$, lo cual no es evidente.

Según el algoritmo de la división euclideana, $\forall t = 0, 1, 2, ...$ existen enteros no negativos r, s tales que $t = s\, t_0 + r$, donde $0 \leq r < t_0$.

Cualquiera que sea el valor de s, al tender t a ∞, s también lo hace, por lo tanto, $\lim_{s\to\infty} P^{t_0 s} = \Pi$

Si $r = 1$, $\lim_{s\to\infty} P^{t_0 s+1} = \lim_{s\to\infty} P\ P^{t_0 s} = P\ \lim_{s\to\infty} P^{t_0 s} = P\Pi$. Ahora, $\forall i = 1, ..., n$, (fila i de $P\Pi$) = (fila i de P)Π; como P es estocástica, se aplica en (fila i de P) el mismo razonamiento hecho anteriormente a q_0, para concluir que (fila i de $P\Pi$) = π. Por lo tanto, $P\Pi = \Pi$.

Si $1 < r < t_0$, se tiene que $\lim_{s\to\infty} P^{t_0 s+r} = \lim_{s\to\infty} P^r\ P^{t_0 s} = P^r\ \lim_{s\to\infty} P^{t_0 s} = P^r\Pi = P(P(...(P\Pi)...)) = \Pi$

Por lo tanto, $\lim_{s\to\infty} P^r\ P^{t_0 s} = \Pi$, $\forall r = 0, 1, ..., t_0 - 1$. De esta manera, se ha demostrado que $\lim_{t\to\infty} P^t = \Pi$. □

Otras formas de demostrar el teorema 2 pueden consultarse en [13], pág. 123 y en [14], capítulo 3. La idea para la anterior demostración fue tomada de [17], completando muchos de los detalles omitidos por el autor. La siguiente proposición nos da una manera concreta de calcular la distribución límite π de una matriz de transición en una cadena de Markov y a la vez establece la unicidad de la distribución límite.

Proposición 2 Si P es una matriz estocástica primitiva, la distribución límite π del teorema 2 es la única solución del sistema:

$$\begin{aligned} xP &= x \\ \sum_{i=1}^{n} x_i &= 1 \end{aligned}$$

donde $x \geq 0$

Demostración:

π es solución, puesto que $\sum_{i=1}^{n} \pi_i = 1$ y para cualquiera que sea la distribución inicial q_0 se tiene:

$$\begin{aligned} \pi P &= (\lim_{t\to\infty} q^{(0)} P^t) P \\ &= \lim_{t\to\infty} q^{(0)} P^{t+1} \\ &= \pi \end{aligned}$$

Además, si x es cualquier otra solución, con $x \geq 0$ y $\sum_{i=1}^{n} x_i = 1$, entonces:

$$x = xP \Longrightarrow x = xP^2 \Longrightarrow \ldots \Longrightarrow x = xP^t \ \ \forall t \in I\!N$$

Por lo tanto,

$$x = \lim_{t\to\infty} xP^t = \pi$$

□

De la proposición 2 se concluye que la distribución límite asociada a una matriz de transición estocástica primitiva P es el único autovector izquierdo de norma euclídea 1, asociado al autovalor 1. El siguiente teorema es otro resultado de límite de matrices de transición, similar al anterior y se refiere al caso en que la matriz de transición no es primitiva pero sí es reducible. Se omite la demostración (ver [13]).

Teorema 2: Sea P una matriz estocástica reducible, $P = \begin{pmatrix} C & 0 \\ R & T \end{pmatrix}$, donde C es una matriz estocástica primitiva, $m \times m$ y R y T no nulas. Entonces:

$$\Pi = \lim_{k\to\infty} P^k = \lim_{k\to\infty} \begin{pmatrix} C^k & 0 \\ \sum_{i=0}^{k-1} T^i R C^{k-1-i} & T^k \end{pmatrix} = \begin{pmatrix} C^\infty & 0 \\ R^\infty & 0 \end{pmatrix}$$

siendo Π una matriz estable, con $\Pi = (1, ...1)'\pi$ donde $\pi = q^0\Pi$ es única e independiente de la distribución inicial q_0, y π satisface: $\pi_i > 0$ para $i = 1, ..., m$ y $\pi_i = 0$ para $i = m+1, ..., n$.

□

2.4 Análisis de AG's con cadenas de Markov

En un AG, la función f a optimizar está definida sobre individuos en $\{0,1\}^L$, suponiendo que la representación cromosómica de las soluciones factibles está compuesta por L genes binarios. El paso de una población de tamaño K a otra puede ser modelado en una cadena de Markov sobre el espacio $I\!E = \{0,1\}^{LK}$. Cada estado corresponde a una y sólo una población que pueda ser obtenida en las diferentes generaciones del algoritmo genético.

Para analizar la convergencia del algoritmo genético se puede hacer una pequeña variación de forma del programa evolutivo presentado anteriormente y propuesto por Goldberg, para obtener lo que en [16] y [28] llaman *algoritmo genético canónico*:

Algoritmo genético canónico

$t \longleftarrow 0$

Generar Q_0 (población Inicial)

Evaluar f en Q_0

Repita

1 Hacer Cruce entre individuos de Q_t

2 Hacer Mutaciones en Q_t

3 Evaluar f en Q_t.

4 Seleccionar la población Q_{t+1} de Q_t

5 $t \longleftarrow t + 1$

Hasta que se cumpla con algún criterio de finalización.

Fin.

Para capturar los cambios probabilísticos en una población, producidos por los operadores genéticos, se plantea una matriz de transición P que se escribe como el producto de tres matrices de transición intermedias para la selección, las mutaciones y el cruzamiento de individuos (ver [16] y [28]): $P = C \cdot M \cdot S$. Estas matrices de transición intermedias modelan los cambios probabilísticos que hacen variar la población Q_t en los pasos 1, 2 y 4 del algoritmo genético canónico.

Notación: En algunas partes del resto del capítulo usaremos la siguiente notación. Dada una población (estado) i de tamaño K,

i_k denota el k-ésimo individuo (suponiendo que los individuos del estado i están identificados por un índice k, con $k \in \{1, ..., K\}$).

$i_k(t)$ denota el k-ésimo individuo del estado i en la generación t.

La probabilidad de que el individuo k del estado i sea seleccionado puede escribirse como

$$\frac{f(i_k)}{\sum_{j=1}^{K} f(i_j)}$$

Teorema 3: La matriz $P = C \cdot M \cdot S$ de transición del algoritmo genético, con probabilidad de mutación $p_m \in \ ]0, 1[$, probabilidad de cruzamiento $p_c \in [0, 1]$ y probabilidad de selección proporcional a la evaluación de los individuos, es positiva.

Demostración:

Las matrices de transiciones intermedias $C = (c_{ij})$, $M = (m_{ij})$ y $S = (s_{ij})$ son estocásticas pues las tres modelan de manera probabilística el paso de un estado a otro.

M es positiva, pues, suponiendo que el operador de mutación se aplica independientemente a cada uno de los *bits* que forman la población, y que $p_m \in \ ]0, 1[$, entonces $m_{ij} = p_m^{H_{ij}}(1 - p_m)^{NK - H_{ij}} > 0$, donde H_{ij} denota la distancia de Hamming entre las dos representaciones binarias de i y j.

S es positiva por columna: Una vez que el operador de mutación en el algoritmo ha conducido a un estado i, cualquiera que éste sea, la probabilidad de que la población i no cambie por el mecanismo de selección puede verse como un muestreo sin reposición, multiplicando las probabilidades de que cada individuo sea seleccionado, sabiendo que los anteriores ya fueron escogidos, quedando,

$$s_{ii} = \Pi_{k=1}^{K} \frac{f(i_k)}{\sum_{j=k}^{K} f(i_j}) \geq \Pi_{k=1}^{K} \frac{f(i_k)}{\sum_{j=1}^{K} f(i_j}) > 0$$

Por lo tanto, S es positiva por columna.

De la proposición 1, se concluye que $P = CMS$ es positiva. □

De acuerdo con los teoremas 3 y 1, se concluye que el AG canónico con los parámetros $p_m \in \ ]0,1[$, probabilidad de cruzamiento $p_c \in [0,1]$ y probabilidad de selección proporcional a la evaluación de los individuos, puede verse como una cadena de Markov cuya matriz $P = CMS$ es positiva y por lo tanto existe una única distribución límite positiva π, por lo que en el límite el AG canónico tiene asociadas probabilidades no nulas de encontrarse en cualquier estado, independientemente de la distribución inicial. A una cadena de Markov cuya matriz de transición sea positiva se le llama *ergódica*.

Un AG *converge* al óptimo global *sii* $\lim_{t\to\infty} P\{Z_t = f^*\} = 1$, donde Z_t, en cada generación t es una variable aleatoria que consiste en la máxima evaluación de f en los individuos de la población P_t y f^* indica el óptimo global de f sobre todo el espacio de soluciones factibles.

Teorema 4: El AG con los parámetros indicados en el teorema 4 no converge al óptimo global.

Demostración:

Sea $i \in I\!E$ cualquier estado tal que $\max\{f(i_k), \ \ k = 1...K\} < f^*$; sea $q_i^{(t)}$ la probabilidad de que el AG se encuentre en tal estado en la etapa t, y $Z_t = \max\{f(i_k), \ \ k = 1...K\}$.

Se tiene que $P\{Z_t \neq f^*\} \geq q_i^{(t)} \iff P\{Z_t = f^*\} \leq 1 - q_i^{(t)}$. Por lo tanto,

$$\lim_{t\to\infty} P\{Z_t = f^*\} \leq 1 - \lim_{t\to\infty} q_i^{(t)} = 1 - \pi_i < 1$$

Debido a esto último, no se satisface la convergencia del AG. □

De acuerdo con el teorema 4, el AG canónico no converge al óptimo global y esto hace parecer a los AG's poco útiles. Sin embargo, dado que un AG genético es modelado mediante una cadena de Markov ergódica, del siguiente teorema se concluye que en el proceso iterativo, el óptimo global es visitado al menos una vez, tras un número finito de iteraciones. Esto permite, haciéndole un pequeño cambio al algoritmo, asegurar la convergencia, obligando en cada etapa a conservar para la siguiente al individuo que alcance la máxima evaluación entre toda la población.

Teorema 5: En una cadena de Markov ergódica el tiempo esperado de transición desde un estado i a otro estado j es finito, cualesquiera que sean los estados i y j.
La demostración se encuentra en [13]. □

2.4.1 Adaptación del AG para asegurar convergencia

Para cualquier etapa t del AG, denotamos con ω_t^* al individuo con mejor desempeño de Q_t, definido tal que $f(\omega_t^*) = \max\{f(\omega) : \omega \in Q_t\}$. La modificación consiste en aumentar en 1 el tamaño de la población manejada por el AG, de manera que un individuo particular, digamos el primero, siempre será el ω_{t-1}^* de la etapa anterior y a éste no se le hacen mutaciones ni cruzamientos, aunque en la etapa t sí podría ser sustituido por otro con mejor desempeño.

Algoritmo genético modificado

$t \longleftarrow 0$

Generar Q_0 (población Inicial)

Evaluar f en Q_0

Obtener ω_0^* y ubicarlo de primero en Q_0, intercambiando individuos, de ser necesario.

Repita

1 Hacer Cruce entre los individuos de Q_t, excepto el primero.

2 Hacer Mutaciones en Q_t, excepto en el primero.

3 Evaluar f en Q_t.

4 Calcular ω_t^*.

5 Incluir ω_t^* como primer miembro de Q_{t+1}.

6 Seleccionar de Q_t los K individuos restantes para completar la población Q_{t+1}.

5 $t \longleftarrow t+1$

Hasta que se cumpla con algún criterio de finalización.

Fin.

Como puede observarse, en la modificación que precede, la copia del mejor individuo de Q_t en Q_{t+1} se realiza antes de llevar a cabo la selección de los individuos restantes hasta completar Q_{t+1}. Otra modificación consiste en realizar la búsqueda del mejor individuo después de realizar la selección. Con ambas modificaciones, se prueba en [28] que el AG converge al óptimo global.

Chapter 3

Algoritmo genético para obtener conjuntos aproximados

Debido a que, dada una tabla de datos, la búsqueda de conjuntos aproximados se realiza pretendiendo mantener altos los índices de calidad y a su vez disiminuir en la medida de lo posible la cantidad de atributos que forman los conjuntos, tal búsqueda se puede enfocar desde la perspectiva de los algoritmos genéticos, haciendo una adecuada representación cromosómica de los conjuntos y la definición de una función de adecuación apropiada para el problema y de las operaciones genéticas.

A continuación se presentan en detalle esas adaptaciones del algoritmo genético para el problema de otención de conjuntos aproximados.

3.1 Representación genética

Recordemos que en la teoría de conjuntos aproximados presentada en el capítulo 1, $X = \{x^1, x^2, ..., x^p\}$ denota el conjunto de los p atributos explicativos que vienen medidos en la tabla de datos de entrada.

Cada subconjunto de X se representa como un cromosoma binario de longitud p. Dado

$P \subseteq X$, $P = (e_1, e_2, ..., e_p)$, con $e_j \in \{0, 1\}$ $j = 1, ..., p$ donde

$e_j = 0$: significa que el atributo j no está en P.

$e_j = 1$: significa que j está presente en P.

Tal representación es apropiada para abarcar todo el espacio de soluciones factibles del problema, que es el conjunto de todos los subconjuntos de X.

Por ejemplo, si $X = \{x^1, x^2, x^3, x^4, x^5\}$ es el conjunto de atributos, el subconjunto $P = \{x^2, x^4\}$ se representa como (0 1 0 1 0). De igual manera, la tira binaria (1 1 1 0 0) representa al subconjunto $P = \{x^1, x^2, x^3\}$.

3.2 Función de adecuación (*fitness*).

Dado $P \subseteq X$ un subconjunto de atributos en su representación binaria[1], definimos q como el el número de ceros que hay en P; q es, por lo tanto, el número de atributos ausentes en P; por la manera de definirlo, se tiene que $0 \leq q \leq p$. Por ejemplo, si $X = \{x^1, x^2, x^3, x^4, x^5\}$, $P = (0\ 1\ 1\ 1\ 0)$ (P es la representación binaria del conjunto $\{x^2, x^3, x^4\}$), entonces $q = 2$.

En una primera implementación del algoritmo había usado como función evaluadora

$$f(P) = \alpha + W_C(1 + \alpha\frac{q}{p})$$

donde W_C es el índice de calidad de aproximaciÍon de Pawlak, obtenida con el conjunto P y α es una constante positiva que cumple un doble propósito:

1^o: Asegurar que $f(P)$ sea positivo, pues es un requisito indispensable en el algoritmo genético.
En efecto, $1 + \alpha\frac{q}{p} \geq 1$ y $W_C \geq 0 \Longrightarrow W_C(1 + \alpha\frac{q}{p}) \geq 0 \Longrightarrow \alpha + W_C(1 + \alpha\frac{q}{p}) > 0 \Longrightarrow f(P) > 0$.

[1]En adelante, a menos que se indique otra cosa, todo subconjunto P de X se usará en su representación binaria

2^o: Pretender que el algoritmo genético premie no solo que un conjunto tenga un índice de calidad alto, sino también que el número de atributos ausentes contribuya a aumentar $f(P)$ en una proporción α [2].

Tal manera de definir $f(P)$ no dió muy buenos resultados puesto que el AG tendía a localizar conjuntos con muy pocos ceros. La siguiente definición de $f(P)$ es una modificación del anterior, sustituyendo W_C por $\frac{W_C+W_e}{2}$. Esta modificación produjo mejores resultados y desde entonces se ha trabajado con ésta:

$$f(P) = \alpha + \frac{W_C + W_e}{2}(1 + \alpha\frac{q}{p})$$

Por ejemplo, dada la tabla de datos de entrada :

Ω	x^1	x^2	x^3	x^4	Y_j
1	0	1	1	0	**1**
2	2	1	0	1	**1**
3	0	0	0	1	**1**
4	1	0	2	0	**1**
5	0	1	1	0	**1**
6	2	1	0	1	**2**
7	0	1	1	0	**2**
8	1	0	2	1	**2**
9	0	0	0	1	**2**

y el subconjuto de atributos $P = (0\ 1\ 1\ 1) = \{x^2, x^3, x^4\}$, la siguiente tabla muestra en su última columna la pertenencia de cada una de las observaciones con alguna de las 4 clases de equivalencia de la partición generada por P.

[2]Lograr que W_C sea alto no es suficiente para definir la función evaluadora, pues el valor más alto de tal índice se logra también cuando P es el conjunto completo X de atributos y se pretende llegar a reducciones que, a lo mejor, son subconjuntos propios de X.

Ω	x^1	x^2	x^3	x^4	$[\omega]$
1	0	1	1	0	**1**
2	2	1	0	1	**2**
3	0	0	0	1	**3**
4	1	0	2	0	**4**
5	0	1	1	0	**1**
6	2	1	0	1	**2**
7	0	1	1	0	**1**
8	1	0	2	1	**5**
9	0	0	0	1	**3**

Como puede notarse, P genera la siguiente partición sobre Ω:

$$\Omega/P = \{\{1,5,7\},\{2,6\},\{3,9\},\{4\},\{8\}\}$$

Al comparar esta partición con la de las categorías:

$$\{Y_1, Y_2\} = \{\{1,2,3,4,5\},\{6,7,8,9\}\}$$

la función *evalueConjunto* detecta que las únicas clases de Ω/P que están contenidas en alguna de las categorías son $\{4\} \subseteq Y_1$ y $\{8\} \subseteq Y_2$, por lo que la suma inferior es

$$\underline{S} = card(\{4\}) + card(\{8\}) = 2$$

Además, como $\{1,5,7\} \cap Y_1 \neq \emptyset$, $\{2,6\} \cap Y_1 \neq \emptyset$, $\{3,9\} \cap Y_1 \neq \emptyset$, $\{4\} \cap Y_1 \neq \emptyset$, $\{1,5,7\} \cap Y_2 \neq \emptyset$, $\{2,6\} \cap Y_2 \neq \emptyset$, $\{3,9\} \cap Y_2 \neq \emptyset$, $\{8\} \cap Y_2 \neq \emptyset$, entonces la suma superior es

$$\overline{S} = (3+2+2+1) + (3+2+2+1) = 16$$

Por lo tanto, si el parámetro α es $\alpha = 0.1$, *evalueConjunto* da como resultado:

$$W_C = \tfrac{2}{16} = 0.125$$

$$W_e = \tfrac{2}{9} = 0.222$$

$$f(P) = 0.1 + \frac{0.125+0.222}{2}(1 + 0.1\frac{1}{5}) = 0.10447$$

3.3 Selección y operadores genéticos

Para empezar, se definen las siguientes variables:

POBL: Es la población que se tiene en un momento determinado del algoritmo genético.

TP: Tamaño de la población.

MNG: Máximo número de generaciones.

t: Número de generación.

eval: Vector que contiene las evaluaciones de todos los conjuntos de POBL.

$$\text{Eval}[i] = f(\text{POBL[i]}) \;\; i = 1, ..., TP$$

3.3.1 Mecanismo de selección

Dada una población la probabilidad de que un conjunto sea seleccionado para pasar a la generación siguiente es se hace implementando una simulación de la ruleta, tal y como se presentó en el capítulo 2.

Procedimiento de Ruleta (retorna un k entero, con $1 \leq k \leq TP$) :

- $\triangleright$ suma $\longleftarrow \sum_{i=1}^{TP} eval[i]$
- $\triangleright$ $r \longleftarrow NumAleatorioEn[0, 1]$
- $\triangleright$ $k \longleftarrow 1, \quad acum \longleftarrow 0.0$
- $\triangleright$ Mientras $(k < TP) \wedge (r > acum + \frac{eval[k]}{suma})$:
  - $\cdot$ $acum \longleftarrow acum + \frac{eval[k]}{suma}$
  - $\cdot$ $k \longleftarrow k + 1$

▷ Retorne k

Procedimiento de Selección:

▷ Para $i = 1...TP$:
· $i_0 \longleftarrow Ruleta$
· PoblTempo[i] $\longleftarrow$ POBL[i_0]
▷ POBL $\longleftarrow$ PoblTempo

3.3.2 Mutaciones

Debido a que la representación cromosómica es binaria, las mutaciones se realizan de la manera más sencilla: escogiendo al azar uno de los p genes y, con probabilidad de mutación p_m, cambiar su estado de 0 a 1 o viceversa.

Procedimiento de mutación:

▷ Para $i = 1...TP$:
· r $\leftarrow$ NumAleatorioEn[0,1]
· Si $r \leq p_m$:
· $j \leftarrow$ NumAleatorioEn[1,...,p]
· Si $POBL[i][j] = 0$: $POBL[i][j] \leftarrow 1$
· Si $POBL[i][j] = 1$: $POBL[i][j] \leftarrow 0$

Por ejemplo, si $P = (0\ 1\ 1\ 0\ 1) = \{x^2, x^3, x^5\}$ es uno de los conjuntos de la población, en una generación dada, si la probabilidad de mutación es $p_m = 0.05$ y si en el momento que le corresponde su turno en el procedimento de mutación el número aleatorio generado en el intervalo $[0, 1]$ es $r = 0.03$ y si la posición j que se escoge al azar entre 1 y 5 resulta ser $j = 5$ entonces se cambia el 1 de esta posición por un 0, quedando el conjunto $(0\ 1\ 1\ 0\ 0) = \{x^2, x^3\}$; si en cambio $j = 4$, el 0 se cambia por un 1, resultando el conjunto $(0\ 1\ 1\ 1\ 1) =$

$\{x^2, x^3, x^4, x^5\}$.

3.3.3 Cruzamientos

Los cruzamientos se hicieron recorriendo cada par de conjuntos consecutivos[3]. Para cada pareja de conjuntos, si con probabilidad p_c se decide cruzarlos, entonces se escoge al azar un punto de corte entre 1 y $p-1$. El cruzamiento ocurre al intercambiar los bloques binarios a partir de ese punto de corte.

Se podrían implementar otras variantes para realizar cruzamientos: una de eśtas es elegir dos puntos de corte y, con probabilidad p_c intercambiar el bloque del centro. Otra posibilidad es recorrer cada una de las posiciones y, con probabilidad p_c intercambiar los genes.

Procedimiento de cruzamiento:

- ▷ Para $i = 1...(TP-1)$:
  - · r ← NumAleatorioEn[0,1]
  - · Si $r \leq p_c$:
    - · padre1 ← POBL[i]
    - · padre2 ← POBL[i+1]
    - · j ← NumAleatorioEn[1,...,p-1]
    - · hijo1 ← padre1[1..j] + padre2[(j+1)..p]
    - · hijo2 ← padre2[1..j] + padre1[(j+1)..p]
    - · POBL[i] ← hijo1
    - · POBL[i+1] ← hijo2
  - · $i \leftarrow i+2$

Por ejemplo, dadas las dos filas de la población:

[3]esta manera de implementar los cruzamientos hace recomendable que la población esté formada por un número par de conjuntos para asegurar que el último de la población también participa

$$\text{padre1} = (0\ 1\ 0\ 1\ 1\ 1\ 1) = \{x^2, x^4, x^5, x^6, x^7\}$$
$$\text{padre2} = (1\ 1\ 1\ 1\ 0\ 0\ 0) = \{x^1, x^2, x^3, x^4\}$$

Si $p_c = 0.5$ y el número aleatorio en $[0, 1]$ generado es $r = 0.3$, se decide realizar su cruzamiento. Si se ha elegido al azar el punto de cruce $j = 4$, al intercambiar los bloques queda:

$$\text{hijo1} = (0\ 1\ 0\ 1\ 0\ 0\ 0) = \{x^2, x^4\}$$
$$\text{hijo2} = (1\ 1\ 1\ 1\ 1\ 1\ 1) = \{x^1, x^2, x^3, x^4, x^5, x^6, x^7\}$$

3.4 Algoritmo

Estructuras de datos:

Básicamente, he usado dos estructuras de datos compuestas en el programa C++:

TIndividuo : Es una estructura que sirve para almacenar un conjunto de atributos (`conjunto`), junto con las mediciones que se le hacen, tales como su desempeño (`ev`), la calidad (`cal`) y la exactitud (`exa`) con que aproxima al conjunto de datos. En la corrida del algoritmo se forma un arreglo de estructuras de este tipo los conjuntos de mejor desempeño en la función evaluadora que se van obteniendo a lo largo de las iteraciones.

TAG: Es una clase que sirve de contexto para poner en funcionamiento el algoritmo genético. Cualquier objeto de esta clase contiene la siguiente información:

`experto:` Matriz $n \times (p+1)$ de enteros. En su columna $(p+1)$ está la categoría en que viene clasificado cada uno de los datos de entrada.

`DATOS:` Matriz $n \times (p+1)$ de enteros. A diferencia de la anterior, la última columna de esta matriz depende de cada subconjunto P de atributos pues en esta posición se escribe la clase de equivalencia que pertenece cada dato de entrada.

`POBL:` Matriz $TP \times p$ de enteros . Para mantener la población de TP subconjuntos.

`suminf:` Vector de TP enteros. En cada posición i se guarda la suma inferior calculada para el conjunto i de la población.

`sumsup:` Vector de TP enteros. En cada posición i se guarda la suma inferior calculada para el conjunto i de la población.

`calidad`:— Vector de TP flotantes. Contiene la calidad de la aproximación para cada conjunto de la población.

`exactitud:` Vector de TP flotantes. Exactitud de la aproximación para cada conjunto de la población.

`eval:` Vector de TP flotantes. Evaluación del desempeño de cada conjunto de la población.

`testeado:` Vector e n booleanos. Para el conteo que se hace para determinar las sumas inferior y superior.

`esquema:` Vector de p símbolos tomados de $\{*, 0, 1\}$. En esta variable se guarda algún esquema que sea detectado en la población final del algoritmo genético.

Además, los objetos de esta clase tienen un comportamiento determinado por las siguientes funciones:

`TAG(nomArch):` Constructor de la clase. Lee n, p y la tabla de datos de un archivo; además realiza todas las asignaciones de memoria y todas las inicializaciones, incluyendo la poblacioón inicial.

`~TAG():` Destructor de la clase. Libera la memoria asignada, una vez que el objeto quede inactivo y fuera de contexto.

`mutacion():` Recorre todos los conjuntos de la población y en cada uno intenta realizar una mutación.

`cruzamiento():` Recorre de dos en dos los conjuntos de la población y los intenta cruzar.

`ruletaEnPobl()`: Simula el mecanismo de la ruleta. Retorna el índice del conjunto favorecido de la población.

`seleccion()`: Realiza una selección de la población a partir de una población dada y usando el mecanismo de la ruleta.

`igualesFilasEnDATOS(i,j,subcAtribs)`: Retorna si dos filas de la tabla de DATOS son iguales o no respecto a un subconjunto de atributos.

`particionarDATOS(int *subconjunto)`: Se obtiene la tabla de datos particionada, de acuerdo con un subconjunto de atributos.

`numClasesDelExperto()`: Retorna el número de clases en que vienen clasificados los datos de la tabla de entrada.

`evalueConjunto(s)`: Calcula la evaluación del conjunto que corresponde al índice s de la población.

`evalueToda()` : Evalúa todos los conjuntos presentes en la población. De paso. se actualizan las entradas de los vectores *eval*, *calidad* , *exactitud*, *suminf* y *sumsup*.

`mejorDePobl` : Retorna el índice del conjunto de la población mejor evaluado.

`estaEnLosMejores(int *elConj)` : Verifica si un conjunto está en la lista de los mejor evaluados.

`actualiceListaMejores(mejorIndice)`: Inserta un mejor conjunto que se encuentra en cierto índice de la población en la lista de los mejores, en caso de no estar incluido.

`alFinalDePobl(fila)`: Traslada un conjunto que está en cierta fila de la población para el final de ésta.

`iteracionDelAG()`: Se encarga de realizar una iteración del algoritmo genético.

`programaEvolutivo(archEntrada)` : Se realizan las iteraciones del algoritmo genético a largo de todas las generaciones, a partir de un archivo en el que se encuentran los datos de entrada. En cada iteración, se va guardando la información pertinente en un archivo que se llama igual que el que trae los datos de entrada, pero en lugar de tener extensión ".dat" tiene extensión ".gen".

`detectarEsquemas()` : En la población final del AG, detecta si se ha formado un esquema. Termina llenando el vector *esquema* con los símbolos $*$, 1 o 0, definiendo $\forall j = 1...p$:

$$\text{esquema[j]} = \begin{cases} 0, \text{ si la cantidad de ceros en la columna } j \text{ es mayor que } (1 - 1.5p_m)TP \\ 1, \text{ si la cantidad de unos en la columna } j \text{ es mayor que } (1 - 1.5p_m)TP \\ *, \text{ de otro modo.} \end{cases}$$

Los detalles de definición de ambas estructuras de datos pueden consultarse en el archivo `AG_CAV2.H` del apéndice 1. Los detalles de implementación de las funciones que definen el comportamiento de la clase AG se encuentran en el archivo `AG_CAV2.CPP` del mismo apéndice.

A continuación se presenta el algoritmo genético implementado para el cálculo de conjuntos aproximados.

Programa evolutivo:

· Abrir archivo de entrada.
·Leer del archivo de entrada n, p y la tabla de datos y
generar una población inicial.
·Cerrar archivo de entrada.

·Crear el archivo de salida y escribir en éste un encabezado, con información
· referente a los parámetros del algortimo.

·Para $t = 1...MNG$:
· Realizar mutaciónes en la población.
· Hacer cruzamientos en la población.
· Evaluar el desempeño de los conjuntos que forman toda la población.
· Hacer selección de los conjuntos que formarán la siguiente generación.
· Calcular el conjunto de mejor desempeño de la población, ubicarlo al final
de ésta y actualizar la lista de conjuntos de mayor desempeño.

· Guardar en el archivo de salida el conjunto de mejor desempeño.

·Cerrar el archivo de salida

Programa principal:

· Concretar el archivo: `entrada`

· Definir el valor de los parámetros globales del algoritmo genético

· Declarar un objeto de la clase `TAG G(entrada)`.

· `G.programaEvolutivo(entrada)`

· Fin.

3.5 Consideraciones sobre la implementación del algoritmo

3.5.1 Implementación en MATHEMATICA

En un principio, se programó el algortimo mediante MATHEMATICA© con el fin de aprovechar los programas que ya trae implementados para el manejo de conjuntos, la partición de tablas de datos de acuerdo con un conjunto de variables, las operaciones con conjuntos, etc. Sin embargo, el programa resultó ser muy lento pues al correrlo con tablas de datos no muy grandes y con poblaciones de tamaño moderado (TP = 50, 100, por ejemplo) duraba varias horas en un computador personal con un procesador de 133 Mhz. Esto obstaculizaba grandemente la posibilidad de correrlo para tablas grandes, con poblaciones mayores, con un número grande de generaciones o de probarlo haciendo algunas variaciones de los parámetros, por lo que opté por programarlo en C++ (Borland, Vers. 4.52).

En el Apéndice 1 se encuentra el programa realizado en C++ y, con el fin de que quede documentada la mayoría del trabajo de programación realizado y, por el interés que pueda suscitar en los posibles lectores que acostumbren a programar en MATHEMATICA, en el

Apéndice 2 se incluye también este primer programa realizado y que tuvo que ser abandonarlo para mejorar la eficiencia.

3.5.2 Implementación en C++

Como mencioné anteriormente, implementé un algoritmo genético aplicado al cálculo de conjuntos aproximados, esta vez con el lenguaje C++, con el fin de aprovechar algunas de las ventajas de la *orientación a objetos*, de poder trabajar con un programa compilado y de la capacidad de este lenguaje para la asignación de memoria dinámica y, por ende, de la flexibilidad para que el programa pueda correr con tablas de tamaño muy grande, todo con miras a disminuir dramáticamente los tiempos de ejecución. Efectivamente, tal disminución se logró y se obtuvo un programa mucho más eficiente, pese a la cantidad de orden combinatorio de cálculos que se realizan en cada corrida.

Chapter 4

Resultados numéricos

A continuación se presentan algunas tablas de datos a los que se les ha aplicado el algoritmo genético planteado. Al final del capítulo aparece un cuadro con las estadísticas de desempeño del algoritmo tras haber sido aplicado cierto número de veces a cada tabla de datos.

En la mayoría de las tablas, para comparar las reducciones obtenidas por el programa, hemos recurrido principalmente a alguna de las tres fuentes:

- La fuente bibliográfica de donde se obtuvo la tabla, si tiene calculadas las reducciones de atributos.

- El programa para la generación de aŕboles de decisión, ID3, (en [2]) , que usualmente retorna alguna de las reducciones.

- El software *Rosetta* que, para fines académicos, es de uso libre y ha sido desarrollado conjuntamente por el Grupo de Sistemas de Conocimiento, del Departamento de Computación y Ciencias de la Información de la Universidad de Ciencia y Tecnología de Noruega (NTNU), y el Grupo de Lógica del Instituto de Matemáticas de la Universidad de Varsovia, Polonia. Este es un sistema de procesamiento de datos para generar reducciones, reglas, etc., que involucra a muchos investigadores y programadores.

4.1 Tabla sobre estudiantes

Volviendo a la tabla 1.1 del apartado 1.5.2, se tienen los siguientes resultados.

```
Tabla: estuds.dat

Cantidad de datos de entrada: 11
Cantidad de variables explicativas: 5

Generaciones:  1000
Tam. Pobl.: 200
Pm :  0.06
Pc:  0.6
```

Resultado:

En todas las 1000 generaciones el AG obtuvo como mejor individuo (11110), con un índice de calidad de 0.357143, de exactitud de 0.454545 y un valor de 0.357143 en la función evaluadora. Este mejor individuo corresponde al conjunto de atributos *Experiencia, Título, Nacionalidad, Dirección* } , excluyendo a la variable *Entrevista*. Este resultado armoniza con el árbol de decisión obtenido a partir de esta misma tabla de datos y que fue presentado en el cap1tulo 1, pues las variables que aparecen en el árbol son las mismas que hay en la única reducción calculada en esta corrida.

Esquema obtenido en la población final :

$$(1 * 11*)$$

4.2 Tabla sobre circuitos eléctricos

La tabla 4.1 fue tomada de [23], pág. 188, y es una tabla de verdad para modelar un determinado circuito eléctrico. Se sabe que tiene una única reducción, que es $\{a, b, d, e\}$, y ésta también se obtiene al correr el programa *Rosetta*).

caso	**a**	**b**	**c**	**d**	**e**	**Clase**
1	0	0	0	0	0	0
2	0	0	0	1	0	0
3	0	1	1	1	0	0
4	0	1	1	0	1	0
5	1	1	1	1	1	0
6	1	0	0	0	1	0
7	0	0	0	1	1	1
8	0	1	0	0	0	1
9	0	1	1	0	0	1
10	1	1	0	0	0	1
11	1	1	0	1	0	1
12	1	1	1	1	0	1
13	1	1	1	0	0	1
14	1	0	0	1	1	1
15	1	0	0	1	0	1

Table 4.1: *"Switching circuits", con 15 medicione, 2 categorías y 5 atributos.*

```
Tabla: switch.dat
Cantidad de datos de entrada: 15
Cantidad de variables explicativas: 5
```

```
Generaciones:  2000
Tam. Pobl.: 500
Pm :  0.05
Pc:  0.5
```

Resultado:

En todas las generaciones el mejor individuo calculado fue (11011), que corresponde al conjunto de atributos $\{a, b, d, e\}$, que es la única reducción ya conocida de la tabla de datos. Además, la calidad de la aproximación es de 1 y la exactitud, de 0.4.

En la población final se obtuvo el esquema:

$$(11 * 11),$$

lo cual era de esperar, pues, dado que tiene sólo una reducción, en la que sólo se excluye el atributo c, éste era el que tenía más posibilidades de estar o no presente en los individuos de una población, mientras que los demás era de esperar que, salvo mutaciones, se mantuvieran en un alto porcentaje en las diferentes filas de la población.

4.3 Tabla de datos sobre felinos

```
Tabla: felinos.dat
Cantidad de datos de entrada: 30
Cantidad de variables explicativas: 7
```

```
Generaciones:  2000
Tam. Pobl.: 500
Pm :  0.05
Pc:  0.5
```

Resultado:

Esta tabla tiene 131 reducciones calculadas con *Rosetta*. De éstas, 17 son de 3 atri- butos y las demás son de 4, 5 y hasta 6 atributos. El AG obtuvo exactamente las 17 mejores (tabla 4.3), todas con tres atributos, con calidad y exactitud iguales a 1 y un valor de 1.0027 en la función evaluadora. Por la manera en que el AG está concebido, es normal que haya detectado sólo las reducciones de cardinalidad tres, pues la definición del *fitness* que hemos hecho estimula la ausencia de atributos.

Un hecho interesante es que al correr el programa ID3 para obtener un árbol de decisión, el conjunto de atributos que son evaluados en éste son $\{a, f, g\}$, que corres- ponde a una de las reducciones (la número 10 de la tabla 4.3).

La tabla 4.5 se obtiene a partir de la 4.4, eliminando las filas iguales respecto al conjunto de atributos {a, f, g}. De esta manera, se tiene un sistema de reglas de discriminación en bruto, sin el tratamiento final para que se vean como reglas formaeles de decisión. En todo caso, tiene mucha relación con el árbol de decisión generado.

Finalmente, puede observarse que la tabla 4.5 es una simplificación muy grande de la tabla original, en cuanto a tamaño y legibilidad. Los datos pasaron de ser abundantes, y poco claros, a ser un conjunto de reglas que, en el sentido de clasificación, contiene la información sintetizada de la tabla original.

4.4 Tabla de datos zoológicos

```
Tabla: zoo.dat
Cantidad de datos de entrada: 101
Cantidad de variables explicativas: 16
```

```
Generaciones:  100
Tam. Pobl.: 200
Pm :  0.05
Pc:  0.5
```

La tabla de datos zoológicos (tablas 4.6 y 4.7), junto con la de datos de felinos es una de las tablas que sirven para probar diversos métodos de análisis de datos.

Consiste en las mediciones de 16 atributos hechas a 101 especies de animales, tales como : 1) Tiene pelo, 2) tiene plumas, 3) pone huevos, 4) produce leche, 5) vuela, 6) es acuático, 7) es depredador, 8) posee dientes, 9) posee columna vertebral, 10) respira soplando, 11) es venenoso, 12) posee aletas, 13) número de patas, 14) posee cola, 15) es doméstico y 13) tiene el tamaño de un león. Todos los atributos, excepto el 13 (número de patas), son binarios. Este último, en el contexto de la tabla, toma los valores 0, 2, 4, 5, 6 y 8 patas. Además las especies están clasificadas en los siguientes grupos:

Clase 1: Mamífero

Clase 2: Ave

Clase 3: Reptil

Clase 4: Pez

Clase 5: Batracio

Clase 6: Insecto

Clase 7: Molusco

Resultado:

La tabla 4.8 muestra 14 reducciones calculadas por el AG. Todos estos conjuntos de atributos tienen asociado algun índice de 1, tanto en el índice de calidad como el de exactitud. Puede

notarse que tanto la variable 6 como la 13 están presentes en todas las reducciones.

Esquema detectado :

$$(0 * * * * 1 * 1 * * * * 1 * * *)$$

4.5 Estadísticas

A continuación presentamos una tabla (4.9) en la que se presentan las estadísticas de desempeño del AG, al haber sido aplicado determinado número de veces a cada una de las tablas de datos. En la última columna, aparece el porcentaje de veces que las corridas del AG visitaron el óptimo

a	b	c	d	e	f	g	h	i	j	k	l	m	n	Clase
1	1	2	1	1	2	3	3	3	2	2	1	1	2	1
3	1	2	3	1	2	3	3	3	2	2	1	1	1	1
2	1	2	3	1	2	3	3	2	1	2	1	2	1	1
2	1	2	3	1	2	3	3	2	2	2	2	2	1	1
2	2	2	1	1	2	2	2	2	3	2	2	2	1	1
4	1	2	3	1	2	2	2	2	3	2	3	2	1	2
1	1	2	2	1	1	2	3	2	3	2	2	2	1	3
2	1	2	1	2	1	2	2	2	1	1	3	2	2	3
2	1	2	2	1	1	2	2	2	2	1	3	2	1	3
2	2	2	2	2	1	2	2	2	1	2	2	2	1	3
1	1	2	2	2	1	2	2	1	1	1	3	2	2	3
2	1	2	2	1	1	1	1	2	2	1	3	1	1	3
1	1	2	2	1	1	1	2	2	3	1	3	2	1	3
1	2	2	3	2	1	1	2	1	2	1	3	2	1	3
1	1	2	3	1	1	1	1	1	1	1	3	2	1	3
2	1	2	3	1	1	1	1	1	2	1	3	2	1	3
1	2	2	2	1	1	1	1	1	2	1	3	1	1	3
3	1	2	3	1	1	1	1	1	2	1	3	2	2	3
1	1	2	2	2	1	1	1	1	1	1	3	2	1	3
2	1	2	3	1	1	1	1	1	2	1	3	2	1	3
2	1	2	2	1	1	1	1	1	2	1	3	2	1	3
1	2	2	3	1	1	1	1	1	1	1	3	2	1	3
1	1	2	3	1	1	1	1	1	2	1	3	2	1	3
2	1	2	2	1	1	1	1	1	1	1	3	2	2	3
1	2	2	3	1	1	1	1	1	1	1	3	2	1	3
4	1	2	3	1	1	1	1	1	3	1	3	2	1	3
2	1	2	3	1	1	1	1	1	2	1	3	2	1	3
1	1	2	3	1	1	1	1	1	2	1	3	2	1	3
2	2	2	3	1	1	1	1	2	2	1	2	2	1	3
2	1	1	1	1	1	3	2	2	3	1	2	1	2	4

Table 4.2: *Datos sobre felinos*

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	Atributos
1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	{c,f,l}
2	0	1	0	0	0	1	1	0	0	0	0	0	0	0	{b,f,g}
3	1	0	0	0	0	0	1	0	0	1	0	0	0	0	{a,g,j}
4	0	0	0	0	0	1	0	0	0	0	0	1	0	1	{f,l,n}
5	0	0	0	1	0	0	1	0	0	1	0	0	0	0	{d,g,j}
6	1	0	0	0	0	1	0	0	0	1	0	0	0	0	{a,f,j}
7	0	0	0	1	0	1	0	0	0	0	0	1	0	0	{d,f,l}
8	0	0	0	1	0	0	1	0	0	0	1	0	0	0	{d,g,k}
9	0	0	0	1	0	0	1	0	0	0	0	1	0	0	{d,g,l}
10	1	0	0	0	0	1	1	0	0	0	0	0	0	0	{a,f,g}
11	0	0	0	1	0	1	1	0	0	0	0	0	0	0	{d,f,g}
12	0	0	0	1	0	0	0	0	0	1	1	0	0	0	{d,j,k}
13	0	0	0	1	0	0	0	0	0	0	1	1	0	0	{d,k,l}
14	0	0	0	0	0	1	0	0	0	0	0	1	1	0	{f,l,m}
15	0	0	0	1	0	1	0	0	0	1	0	0	0	0	{d,f,j}
16	0	0	0	0	0	1	1	0	0	0	0	1	0	0	{f,g,l}
17	1	0	1	0	0	1	0	0	0	0	0	0	0	0	{a,c,f}

Table 4.3: *Reducciones de la tabla de felinos obtenidas por el AG.*

a	f	g	Clase	a	f	g	Clase
1	2	3	1	2	1	1	3
3	2	3	1	1	1	1	3
2	2	3	1	3	1	1	3
2	2	3	1	1	1	1	3
2	2	2	1	2	1	1	3
4	2	2	2	2	1	1	3
1	1	2	3	1	1	1	3
2	1	2	3	1	1	1	3
2	1	2	3	2	1	1	3
2	1	2	3	1	1	1	3
1	1	2	3	4	1	1	3
2	1	1	3	2	1	1	3
1	1	1	3	1	1	1	3
1	1	1	3	2	1	1	3
1	1	1	3	2	1	3	4

Table 4.4: *Datos sobre felinos usando la reducción* $\{a, f, g\}$

a	f	g	Clase
1	2	3	1
3	2	3	1
2	2	3	1
2	2	2	1
4	2	2	2
1	1	2	3
2	1	2	3
2	1	1	3
1	1	1	3
3	1	1	3
4	1	1	3
2	1	3	4

Table 4.5: *Datos sobre felinos usando la reducción* $\{a, f, g\}$*, omitiendo las filas iguales.*

1	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	1
2	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
3	1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	1
4	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
5	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
6	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	1
7	1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	1
8	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
9	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
10	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1
11	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
12	1	0	0	1	1	0	0	1	1	1	0	0	2	1	0	0	1
13	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
14	1	0	0	1	0	0	1	1	1	1	0	0	2	0	1	1	1
15	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	1
16	1	0	0	1	0	0	0	1	1	1	0	0	2	0	0	1	1
17	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	0	1
18	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	0	1
19	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
20	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
21	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
22	1	0	0	1	0	1	1	1	1	1	0	0	4	1	0	1	1
23	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	0	1
24	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
25	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	0	1
26	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	1
27	1	0	1	1	0	1	1	0	1	1	0	0	4	1	0	1	1
28	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
29	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	1
30	0	0	0	1	0	1	1	1	1	1	0	1	0	1	0	1	1
31	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
32	1	0	0	1	0	0	1	1	1	1	0	0	4	1	1	1	1
33	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1
34	1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	1
35	1	0	0	1	0	1	1	1	1	1	0	1	0	0	0	1	1
36	1	0	0	1	0	1	1	1	1	1	0	1	2	1	0	1	1
37	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0	0	1
38	1	0	0	1	1	0	0	1	1	1	0	0	2	1	0	0	1
39	1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	0	1
40	1	0	0	1	0	0	0	1	1	1	0	0	2	1	0	1	1
41	1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	1

52	0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	2
53	0	1	1	0	0	1	1	0	1	1	0	0	2	1	0	1	2
54	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0	0	2
55	0	1	1	0	0	0	1	0	1	1	0	0	2	1	0	1	2
56	0	1	1	0	1	1	1	0	1	1	0	0	2	1	0	0	2
57	0	1	1	0	1	1	1	0	1	1	0	0	2	1	0	0	2
58	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0	0	2
59	0	1	1	0	1	1	0	0	1	1	0	0	2	1	0	1	2
60	0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	1	2
61	0	1	1	0	1	0	0	0	1	1	0	0	2	1	0	0	2
62	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0	3
63	0	0	0	0	0	1	1	1	1	0	1	0	0	1	0	0	3
64	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0	3
65	0	0	1	0	0	0	0	0	1	1	0	0	4	1	0	1	3
66	0	0	1	0	0	0	1	1	1	1	0	0	4	1	0	0	3
67	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
68	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	4
69	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
70	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
71	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	4
72	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	0	4
73	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
74	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	4
75	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	4
76	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	0	4
77	0	0	1	0	0	1	0	1	1	0	0	1	0	1	0	0	4
78	0	0	1	0	0	1	1	1	1	0	1	1	0	1	0	1	4
79	0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	4
80	0	0	1	0	0	1	1	1	1	1	0	0	4	0	0	0	5
81	0	0	1	0	0	1	1	1	1	1	1	0	4	0	0	0	5
82	0	0	1	0	0	1	1	1	1	1	0	0	4	1	0	0	5
83	0	0	1	0	0	1	0	1	1	1	0	0	4	0	0	0	5
84	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0	0	6
85	0	0	1	0	1	0	0	0	0	1	0	0	6	0	0	0	6
86	1	0	1	0	1	0	0	0	0	1	1	0	6	0	1	0	6
87	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0	0	6
88	0	0	1	0	1	0	1	0	0	1	0	0	6	0	0	0	6
89	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0	0	6
90	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0	0	6
91	1	0	1	0	1	0	0	0	0	1	1	0	6	0	0	0	6
92	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	eval	Atributos
	1	0	0	1	0	1	0	0	0	1	0	1	1	1	0	1	1.002	{ 1,4,6,10,12,13,14,16}
2	0	1	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1.002	{ 2,3,5,6,8,10,13,14}
3	1	0	1	0	0	1	0	1	1	0	0	0	1	0	0	1	1.00213	{ 1,3,6,8,9,13,16}
4	0	1	1	0	0	1	1	1	0	1	0	0	1	0	0	0	0.00213	{ 2,3,6,7,8,10,13}
5	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	0.00213	{ 3,5,6,8,10,13,14}
6	0	0	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0.00213	{ 3,4,5,6,7,8,13}
7	0	0	0	1	0	1	0	0	1	1	0	1	1	0	1	0	0.00213	{ 4,6,9,10,12,13,15}
8	0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	1	1.00225	{ 2,3,6,9,13,16}
9	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	1.00238	{ 3,6,9,13,16}
10	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	0.00238	{ 3,4,6,9,13}
11	0	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0.00238	{ 3,6,8,10,13}
12	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1	1.00238	{ 3,6,8,13,16}
13	0	0	1	1	0	1	0	1	0	0	0	0	1	0	0	0	0.00238	{ 3,4,6,8,13}
14	0	0	0	1	0	1	0	1	0	0	0	1	1	0	0	0	0.00238	{ 4,6,8,12,13}

Table 4.8: *Reducciones de los datos zoológicos calculadas por el AG.*

Tabla	Población	Generaciones	Óptimo	Núm. corridas	Porcentaje
Felinos.dat	500	500	1.00257	500	100
Zoo.dat	200	500	1.00238	100	100
switch.dat	500	1000	1.0014	500	100

Table 4.9: *Estadísticas de efectividad del AG propuesto .*

Conclusiones

El presente trabajo de investigación ha pretendido calcular reducciones óptimas mediante algoritmos genéticos. Queda abierto el abordar el problema práctico-teórico por medio de otras técnicas de optimización, como búsqueda tabú y sobrecalentamiento simulado.

Chapter 5

Programa *C++*

AG aplicado a CA - Programa en C++

```
//  AG_CAV2.H


#ifndef _AG_CAV2_H
#define _AG_CAV2_H
#include<stdlib.h>
#include<iostream.h>
#include <alloc.h>
#include <string.h>
#include <fstream.h>
#define SI 's'
#define NO 'n'


struct TIndividuo
{
  int  *conjunto;  // El subconjunto de atributos.
  float ev;        // El desempeo
  float cal;       // La calidad
  float exa;       // Exactitud
};
```

```
class TAG
{  public:
       int   **DATOS, **experto;
       int   **POBL;
       int   *suminf, *sumsup;
       float *calidad, *exactitud, *eval;
       char  *testeado;
       char  *esquema;

       TAG(char *nomArch);
      ~TAG();
       void mutacion();
       void cruzamiento();
       int  ruletaEnPobl();
       void seleccion();
       int  igualesFilasEnDATOS(int i,int j,int *subcAtribs);
       int  particionarDATOS(int *subconjunto);
       int  numClasesDelExperto();
       void evalueConjunto(int s);
       void evalueToda();
       int  mejorDePobl();
       int  estaEnLosMejores(int *elConj);
       void actualiceListaMejores(int mejorIndice);
       void alFinalDePobl(int fila);
       void iteracionDelAG();
       void programaEvolutivo(char *archEntrada);
       void detectarEsquemas();
};
#endif
```

```
// AG_CAV2.CPP

#include "ag_cav2.h"
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <string.h>
#define  Numejores 25
char   procesar[200];

int n;       // Nmero de filas de la tabla de entrada.
int p;       // Nmero de variables explicativas
int q;       // q = No. de variables ausentes en un conjunto de p variables.

int MaxNumGen;

int TP;
float Pm =0.08;
float  Pc =0.6;
float   alfa=0.001;

TIndividuo LosMejores[(Numejores+1)];

float NoAleat()
{
   return rand() / 32767.1;
};

void error(int i)
{
  switch(i){
     case 1: cout << "Error al reservar memoria para alguna fila de DATOS o experto.\n ";
```

```
                        break;
     case 2: cout << "No se pudo asignar memoria a DATOS o a experto.\n ";
                        break;
     case 3: cout << "Error: fall res. mem. en testeado";
                        break;
     case 4: cout << "Error al reservar memoria, alguna fila de POBL\n";
                        break;
     case 5: cout << "Error al reservar memoria a POBL\n";
                        break;
     case 6: cout << "No se pudo asignar memoria para PoblTempo \n";
                        break;
     case 7: cout << "Error al reservar memoria, alguna fila de PoblTempo.\n";
                        break;
     case 8: cout << "Error al res. mem. para 'conjunto' en evalue..\n";
                        break;
     case 9:  cout << "Error causado al dividir por cero en evalueConjunto..\n";
                        break;
     case 10: cout << "La suma de 'ruletaEnPobl' es cero..\n";
                        break;
     default:cout << "error no tipificado. JLEB\n";
   };
     getche();
     exit(1);
};

TAG::TAG(char *nomArch)    // constructor de la clase TAG.
{   fstream data(nomArch,ios::in);
    if (data)
        {
            int ncols, valorij;
            data >> n >> ncols;
            p= ncols-1;
```

```
    DATOS    = new int*[(n+1)];
    experto  = new int*[(n+1)];
    testeado = new char[(n+1)];
    if(!DATOS && !experto) error(2);
    for(int i=0; i<(n+1); i++)
      {
        DATOS[i] = new int[(p+2)];
        experto[i] = new int[(p+2)];
        if(!DATOS[i] || !experto[i]) error(1);
      };
      if(!testeado) error(3);
      for(i=0; i<n; i++)
      {
        for(int j=0; j<ncols; j++)
          {
             data >> valorij;
             DATOS[i][j] = valorij;
             experto[i][j] = valorij;
          };
      testeado[i]=NO;
};
data.close();
}

POBL   = new int*[(TP+1)];
suminf = new int[(TP+1)];
sumsup = new int[(TP+1)];
calidad  = new float[(TP+1)];
exactitud = new float[(TP+1)];
eval     = new float[(TP+1)];
esquema  =new char[(p+1)];
if(!POBL) error(5);
```

```
for(int k=0; k<(TP+1); k++)
    {
     POBL[k] = new int[(p+1)];
     if(!POBL[k]) error(4);
     };
     // Inicializar el generador de nmeros aleatorios
     randomize();
     int SEMILLA;
     SEMILLA= random(30000);
     srand(SEMILLA);

     if(!suminf||!sumsup||!eval||!calidad||!exactitud) error(6);
       for(int i=0;i<TP;i++)
         {
           suminf[i]= 0;
           sumsup[i] = 0;
           eval[i] = 0.0;
           calidad[i]= exactitud[i]= 0.0;
           for(int j=0; j<p ; j++)
             if ( NoAleat()< 0.5 )
                POBL[i][j]=0;
             else
                POBL[i][j]=1;
         };
       for(i=0;i<Numejores;i++)
         { LosMejores[i].conjunto = POBL[i];
           LosMejores[i].ev = 0.0;
           LosMejores[i].cal = 0.0;
           LosMejores[i].exa = 0.0;
         };

       for(int j=0;j<p;j++)
```

```
                    esquema[j]='*';
}


TAG::~TAG()        // destructor de la clase TAG.
{
    for(int i=0; i<(n+1);i++)
      {
         delete [] DATOS[i];
         delete [] experto[i];
      };
    delete [] DATOS;
    delete [] experto;
    delete [] testeado;
    for(int k=0; k<(TP+1);k++)
    delete [] POBL[k];
    delete [] POBL;
    delete [] suminf;
    delete [] sumsup;
    delete [] calidad;
    delete [] exactitud;
    delete [] eval;
    delete [] esquema;
};

void TAG::mutacion()
{
   int j;
   randomize();
   for(int i=0;i<TP;i++)
      if(NoAleat()<=Pm)
         {
```

```
            j = random(p);
            POBL[i][j] ? POBL[i][j] = 0 : POBL[i][j] = 1;
          };
};

void TAG::cruzamiento()
{ int tempo;
  randomize();
  for(int i=0; (i+1) < TP ; i=i+2 )
     {
        if(NoAleat()<=Pc)
           {
             int j0=1+random(p-1);
             for(int j=j0;j<p;j++)
             {
               tempo = POBL[i][j];
               POBL[i][j] = POBL[(i+1)][j];
               POBL[(i+1)][j]=tempo;
              };
            };
    };
}

int TAG::ruletaEnPobl()
{
  double suma = 0.0, s = 0.0;
  for(int i=0;i<TP;i++)
   {
     suma = suma + eval[i];
   };
  if(suma == 0.0) error(10);
  float r;
```

```
  r=NoAleat();
  int k=0;
  while( (k<TP) && (r > s+ eval[k]/suma ) )
    {
      s=s + eval[k]/suma;
      k++;
      };
  return k;
}

void TAG::seleccion()
{
   int **PoblTempo=new int*[(TP+1)];
   int  *infTempo= new int[(TP+1)];
   int  *supTempo= new int[(TP+1)];
   float *evalTempo=new float[(TP+1)];
   float *caliTempo=new float[(TP+1)];
   float *exacTempo=new float[(TP+1)];
   if(!PoblTempo) error(6);
   for(int k=0; k<(TP+1); k++)
    {
      PoblTempo[k] = new int[(p+1)];
      if(!PoblTempo[k]) error(7);
    };
    int suerte;
    for(int i=0;i<TP;i++)
       { suerte = ruletaEnPobl();
         for(int j=0;j<p;j++)
             PoblTempo[i][j] = POBL[suerte][j];
         infTempo[i]=suminf[suerte];
         supTempo[i]=sumsup[suerte];
         evalTempo[i]=eval[suerte];
```

```
         caliTempo[i]=calidad[suerte];
         exacTempo[i]=exactitud[suerte];
        };
    for(i=0;i<TP;i++)
     {
        for(int j=0;j<p;j++)
           POBL[i][j]=PoblTempo[i][j];
        suminf[i]=infTempo[i];
        sumsup[i]=supTempo[i];
        eval[i]=evalTempo[i];
        calidad[i]=caliTempo[i];
        exactitud[i]=exacTempo[i];
      };

    for(int fila=0; fila<(TP+1);fila++)
    delete []PoblTempo[fila];
    delete []PoblTempo;
    delete []infTempo;
    delete []supTempo;
    delete []evalTempo;
    delete []caliTempo;
    delete []exacTempo;
}

int TAG::igualesFilasEnDATOS(int unaFila, int otraFila, int *subcAtribs)
{
   for(int j=0; j<p; j++)
       if( (subcAtribs[j] && (DATOS[unaFila][j]!= DATOS[otraFila][j] ) ) )
           return 0;
   return 1;
}
```

```
int TAG::particionarDATOS(int *subconjunto)
{ /* Col. p de DATOS es la dedicada a describir la particin
       en que queda cada fila de acuerdo con el 'subconjunto'
       del conjunto total de atributos. */
   int actual=1;
   for(int i=0; i<n; i++) DATOS[i][p]=0 ;
   for(i=0; i<n; i++)
     { if ( DATOS[i][p] == 0 )
            DATOS[i][p] = actual;
            else continue;
            for(int fila=i+1; fila<n; fila++)
               if(!DATOS[fila][p]  && igualesFilasEnDATOS(i,fila,subconjunto) )
                   DATOS[fila][p] = actual;
            actual++;
     }
   return(actual-1);
}

int TAG::numClasesDelExperto()
{
   int cuentaClases = 0;
   for(int i=0;i < n;i++)
       if ( experto[i][p] > cuentaClases)
           cuentaClases = experto[i][p];
   return (cuentaClases);
}

void TAG::evalueConjunto(int s)
{
   int inf = 0, sup = 0,i;
   int *conjunto;
   conjunto = new int [(p+1)];
```

```
if(!conjunto) error(8);
for(int col = 0; col<p; col++)
    conjunto[col]=POBL[s][col];
particionarDATOS(conjunto);
int inicio, pivote, primero, ultimo, cardi;
int numClases = numClasesDelExperto();
for(i=0;i<n;i++)
   testeado[i]=NO;
pivote = -1;
       // Corazn de la rutina para evaluar:
for(int clase=1; clase<=numClases;clase++)  // de las clases del experto:
   {
      inicio = pivote+1;
      int tmp=inicio;
      //  Localiza el pivote:
      while(tmp<n)
         { if( experto[tmp][p]  == experto[inicio][p] )
             tmp++;
           else break;
         };
      pivote = tmp-1;
      primero = -1;
      for(int b=inicio;(b<=pivote);b++)
        if(testeado[b]==NO)
           {
             primero = b;
             ultimo = b;
             cardi=0;
             for(i=0; i< n; i++)
               if((testeado[(b+i)%n]==NO)&&(DATOS[(b+i)%n][p]==DATOS[primero][p] ))
                {
                 testeado[(b+i) % n]=SI;
```

```
                    cardi++;
                    ultimo = (b+i) % n;
                   };
                  if(primero <= pivote)
                    { sup += cardi;
                      if((primero<=ultimo)&&(ultimo<=pivote))
                        inf+=cardi;
                    };
               };
        for(int y=0;y<n;y++)
             testeado[y]=NO;
      };   // fin de para cada clase del experto.
     suminf[s] = inf;
     sumsup[s] = sup;
     if(p==0||sup==0||n==0) error(9);
     q=0;
     for(int j=0;j<p;j++)
        if (!POBL[s][j]) q++;
     calidad[s] = (float)inf /(float)sup ;
     exactitud[s] = (float)inf / (float)n ;
     eval[s] =  alfa + (1.0 + alfa*(float)q/(float)p)*( calidad[s] + exactitud[s] )/2 ;
     delete [] conjunto;
 }

void TAG::evalueToda()
 {
    for(int i=0;i<TP; i++)  // P' evaluar toda la poblacin.
       evalueConjunto(i);
 }

int  TAG::mejorDePobl()
 {
```

```
   int iMejor = 0;
   for(int i =0;i<TP;i++)
      if (eval[i] > eval[iMejor])
          iMejor = i ;
   return iMejor;
 }

int TAG::estaEnLosMejores(int *elConj)
 {
   int  numIguales;
   for(int i=0;i<Numejores;i++)
     {
       numIguales = 0;
       for(int j=0;j<p;j++)
          {
            if(elConj[j]!=LosMejores[i].conjunto[j])
               break;
            numIguales++;
           };
         if(numIguales>=p)
              return 1;
      };
      return 0;
 }

void TAG::actualiceListaMejores(int mejor)
{
  for(int i=(Numejores-1);i>=0;i--)
     if((eval[mejor]>=LosMejores[i].ev)&& !estaEnLosMejores(POBL[mejor]) )
        {
          for(int k=0;k<i;k++)
             LosMejores[k] = LosMejores[(k+1)];
```

```
          LosMejores[i].conjunto = POBL[mejor];
          LosMejores[i].ev = eval[mejor];
          LosMejores[i].cal = calidad[mejor];
          LosMejores[i].exa = exactitud[mejor];
          return;
        }
}

void TAG::alFinalDePobl(int fila)
 {
   if(fila < TP-1){
     int tempo;
     float tempofl;
     for(int j=0;j<p;j++)
        {
          tempo = POBL[fila][j];
          POBL[fila][j]=POBL[(TP-1)][j];
          POBL[(TP-1)][j]=tempo;
         };
      tempo = suminf[fila];
      suminf[fila]=suminf[TP-1];
      suminf[TP-1]=tempo;
      tempo = sumsup[fila];
      sumsup[fila]=sumsup[TP-1];
      sumsup[TP-1]=tempo;
      tempofl = calidad[fila];
      calidad[fila]=calidad[TP-1];
      calidad[TP-1]=tempofl;
      tempofl = exactitud[fila];
      exactitud[fila]=exactitud[TP-1];
      exactitud[TP-1]=tempofl;
      tempofl = eval[fila];
```

```
      eval[fila]=eval[TP-1];
      eval[TP-1]=tempofl;
    } ;
}

void TAG::iteracionDelAG()
{
   mutacion();      //<----  MUTACION
   cruzamiento();   // <---  CRUZAMIENTO
   evalueToda();    // <---  EVALUACION
   int mejorFila = mejorDePobl();
   actualiceListaMejores(mejorFila);
   alFinalDePobl(mejorFila);
   seleccion();     // <----- SELECCION
}

void TAG::programaEvolutivo(char *archEntrada)
{
   fstream O;     // Para ir guardando los resultados en un archivo.
   char *archSalida =archEntrada;
   archSalida = strtok(archSalida,".");
   strcat(archSalida,".gen");
   O.open(archSalida,ios::out);
   O<<"C o n j u n t o s   A p r o x i m a d o s"<<endl<<endl;
   O<<"Archivo: "<<archSalida << endl;
   O<<"Nmero de datos de entrada: "<< n << endl;
   O<<"Nmero de variables explicativas: "<< p<<endl<<endl;
   O<<"Parmetros del Algoritmo Gentico: "<<endl;
   O<<"-----------------------------------" << endl;
   O<<"Nmero de generaciones: "<< MaxNumGen <<endl;
   O<<"Tamao de la Poblacin: "<< TP << endl;
   O<<"Probabilidad de mutacin:    "<< Pm << endl;
```

```
O<<"Probabilidad de cruzamiento: "<< Pc << endl << endl;
O<<endl<<endl << "Poblacin inicial: "<< endl;
for(int i=0;i<TP;i++)
   {
     for(int j=0;j<p;j++)
        O<<POBL[i][j];
     O<<endl;
   };
O<<endl << endl;
O<<"Mejores individuos de cada generacin t=1..";
O<< MaxNumGen << ':' << endl;
O<<"t > MejorConjunto > Eval > calidad > exactitud " << endl;
O<<"------------------------------" << endl;
gotoxy(5,4); cout << "Generacin:";
for(int t = 1; t <= MaxNumGen; t++)
    {
      iteracionDelAG();
      O<<t<< " > " ;
      for(int j=0;j<p;j++)
      O<<POBL[(TP-1)][j];
      O<<" > "<<eval[(TP-1)]<<" > "<<calidad[(TP-1)];
      O<<" > "<<exactitud[(TP-1)] << endl;
      gotoxy(17,4); cout << t;
    };
 gotoxy(10,6); cout << " Proceso realizado satisfactoriamente.";
 detectarEsquemas();
 O<<endl<<endl << "Poblacin final: "<< endl;
 for(i=0;i<TP;i++)
    {
      for(int j=0;j<p;j++)
           O<<POBL[i][j];
      O<< " -> " << eval[i] << endl;
```

```
       };
    O<<endl << endl;
    O<<"Esquema: " << endl;
    for(int j=0;j<p;j++)
       O<<esquema[j];
    O<<endl<<endl<<"Los " << Numejores <<" mejores conjuntos:"<< endl;
     for(i=0;i<Numejores;i++)
       {  O<< i << "->";
          for(int j=0;j<p;j++)
             O<<LosMejores[i].conjunto[j];
          O<<" "<<LosMejores[i].ev << " > ";
          O<<LosMejores[i].cal<<" > "<<LosMejores[i].exa<<endl;
       }
  O.close();
}

void TAG::detectarEsquemas()
{  int sumaUnos, sumaCeros;
   for(int j=0;j<p;j++)
       {
         sumaUnos  = 0;
         sumaCeros = 0;
         for(int i=0;i<TP;i++)
                  POBL[i][j] ? sumaUnos++ : sumaCeros++;
         if( sumaUnos  >= (1-Pm)*TP*0.99 ) esquema[j] ='1';
         if( sumaCeros >= (1-Pm)*TP*0.99 ) esquema[j] ='0';
       };
}


// Este es el Programa Principal...
main()
```

```
 {
        char *Entrada = "amiard.dat";
        MaxNumGen = 500;
        TP = 400;
        TAG G(Entrada);
        G.programaEvolutivo(Entrada);
        return 0;
}
```

Chapter 6

Programa en *Mathematica*

AG aplicado a CA - Programa en Mathematica

```
almacen = {
 {2, 2, 1, 2 }, {2, 2, 1, 1 }, {2, 1, 1, 2 }, {1, 2, 2, 2 }, {1, 2, 2, 1 }, {1, 2, 1, 2 },
 {1, 2, 1, 1 }, {1, 1, 2, 2 }, {1, 1, 2, 1 }, {1, 1, 1, 2 }, {1, 1, 1, 1 }, {2, 2, 2, 2 },
 {2, 2, 2, 1 }, {2, 2, 1, 2 }, {2, 2, 1, 1 }, {2, 1, 2, 1 }, {2, 1, 1, 2 }, {2, 1, 1, 1 },
 {1, 2, 2, 2 }, {1, 2, 2, 1 }, {1, 2, 1, 2 }, {1, 2, 1, 1 }, {1, 1, 2, 2 }, {1, 1, 2, 1 },
 {1, 1, 1, 2 } };

Yalmacen={ {1,2,3,4,5,6,7,8,9,10,11},
                {12,13,14,15,16,17,18,19,20,21,22,23,24,25}  } ;

(* DATOS DE ENTRADA:  *)
   tabla = almacen;
   Y = Yalmacen;

  (*  Parmetros de Conjuntos Aproximados:   *)
      p=Length[tabla[[1]]];   (*solo atributos*)
      n=Length[tabla];
      alfa=0.01;
```

```
   (* Parmetros globales del Algoritmo Gentico :  *)
      Pc = 0.5;    (* Probabilidad de cruzamiento *)
   Pm = 0.08;   (* Probabilidad de mutacin    *)
   TP=20;  (* Tama\no de la poblacin     *)
   MaxNumGen=50; (* Mximo nmero de generaciones *)

(*==== MANEJO DE CONJUNTOS APROXIMADOS ====*)

(*A es subconjunto de B?   Retorna True o False *)

subset[A_,B_]:=Module[{ AcB = False },
       If[Length[ Intersection[ A ,  B ] ]==Length[A], AcB=True ];
       AcB
       ]; (* fin modulo subset *)

(* Interseccion de A y B no vacia?    Retorna True o False *)

IntersNoVacia[A_,B_]:=Module[{AintB = False},
    If[Length[Intersection[A,B]]>0, AintB=True];
    AintB
    ];    (* fin de IntersNoVacia *)

Needs["DiscreteMath`Combinatorica`"]

(* sumInfSup:  recibe P subc de Q y devuelve los dos valores
   {suminf, sumsup} que  consisten, respectivamente en la
    aproximacin inferior y superior asociadas con P   *)

sumaInfSup[P_]:=Module[{i,j,k,R={},fili,tbaux,reduccion={},
                                  suminf=0, sumsup=0},
     If[ Length[P]==0,Return[{0.0,0.0}] ];
```

```
      (*calculo de particiones generadas por P*)
     R=IdentityMatrix[n];
     tbaux=Table[{},{k,1,n}]; (*n filas vacias*)
     Do[
           tbaux[[k]]=Part[tabla[[k]],P]  (*tabla con columnas segn P*)
          ,{k,1,n}
          ];   (* do *)

     Do[If[tbaux[[fili]]==tbaux[[k]],   (*Match*)
          R[[fili,k]]=1; R[[k, fili]] =1]; (*if*)
         ,{fili,1,n-1},{k,fili+1,n} ];
     particion=EquivalenceClasses[R];

     Do[     (* Pclase[[i]] es particion[[i]]*)
             If[subset[  particion[[j]],Y[[i]] ],
             suminf+=Length[ particion[[j]] ];
             sumsup+=Length[ particion[[j]] ];
             (*else*)
             If[ IntersNoVacia[particion[[j]],Y[[i]]],
                sumsup+=Length[ particion[[j]] ]
             ] ] (*if*)
            ,{i,1,Length[Y]},{j,1, Length[particion] }];
     Return[{N[suminf],N[sumsup]}];
];   (* fin modulo sumaInfSup *)

  (* convierte un subconjunto de atributos en binario
       a sus etiquetas, que son numeros ordinales {1,0,0,1} --> {1,4}   *)
ordinal[P_]:= Module[{i, conjunto={} },
         Do[ If[ P[[i]]==1, AppendTo[conjunto,i]
                  ],                (* fin de If *)
             {i,1,p} ];  (* fin de Do *)
         Return[conjunto]
```

```
];  (* fin de ordinal *)

  (*==PARA EL ALGORITMO GENETICO ==*)

  (* Inicializa la poblacin:    *)
 Poblacion=Partition[Table[Random[Integer,{0,1}],{TP*p}],p];

 F;  (* La lista de las evaluaciones de la poblacin *)
 Cali; (* Calidad de las aproximaciones          *)
 Exac; (* Exactitud de las aproximaciones          *)
(* Evaluacion de toda la poblacion de subconjuntos *)
evalToda[pobla_]:= Module[ {q, conjunto, sumas, calidad, exactitud, fitness},
Cali={}; Exac={}; F={};   (* del mismo tamao de la poblacin *)
   Do[
   conjunto = ordinal[ pobla[[i]]];
   q = p - Length[conjunto];
   sumas=sumaInfSup[conjunto];
   If[ sumas[[2]]==0, calidad = 0.0,calidad = sumas[[1]]/sumas[[2]]  ];
   exactitud = N[ sumas[[1]]/n ];
   fitness = alfa + (calidad+exactitud)*(1+ alfa*q/p)/2;
   AppendTo[Cali, calidad];
   AppendTo[Exac, exactitud];
   AppendTo[F, fitness];
   ,{i,1,TP}
       ]; (* Do *)
]; (* Fin de evalToda *)


(* MAXIMO:  Retorna el ndice iMax en que se localiza el
maximo valor de una lista de numeros reales *)

 maximo[R_]:=Module[{vMax,iMejor,i},
```

```
  vMax=R[[1]];
iMejor = 1;
            Do[ If[ R[[i]] > vMax ,
                                vMax=R[[i]];
          iMejor = i
        ,(*else*) If[ (R[[i]]== vMax) && (Random[] > 0.5),
                        vMax=R[[i]];
             iMejor = i,
                           ] (* if *)
        ] (* if *)
 ,{i,2,TP}
          ]; (* do *)
           Return[iMejor];
  ];(* fin de maximo *)

(* SUMA: retorna la suma de una lista de TP numeros reales *)

 suma[R_]:=Module[{acum=0.0, i},
        Do[ acum = acum + R[[i]];
           , {i,1,TP}]; (* do *)
        Return[acum];
]; (* fin de suma *)


(* RULETA: Retorna el No. de individuo de una poblacion,  seleccionado
  de acuerdo con el mecanismo de la ruleta  aplicado a la lista
de  valores de desempe\no del grupo de individuos  *)

ruleta[R_]:=Module[{r,s,k,v},
r = Random[];
v=R/suma[R];  (* para que la suma de 1 *)
k=1;
```

```
s=0.0;
While[(k<=TP)&&(r>s+v[[k]]),
      s=s+v[[k]];
      k++ ;
     ];
Return[k];
]; (* fin de ruleta *)

(* MUTACIONES: Recorre todos los individuos de la poblacion; en
cada uno se escoge una posicion al azar y, con
probabilidad Pm se cambia de 0 a 1 o de 1 a 0. *)

 mutaciones[pobla_]:= Module[{var,pT,x},
         pT = pobla;
         Do[ var=Random[Integer,{1,p}];
                 If[(Random[]< Pm),
                     x = ReplacePart[pT[[i]],Mod[(pT[[i]][[var]]+1),2],var];
   pT= ReplacePart[pT, x, i];
  ];  (* if *)
               ,{i,1,TP}
               ];  (* do *)
          Return[pT];
];  (* fin de mutaciones *)

(* CRUCEDOS : Procedimiento para cruzar dos padres. Con
   probabilidad Pc intercambia los bloques en un punto
   escogido al azar para formar dos hijos   *)

cruceDos[padre1_,padre2_]:= Module[{hijo1,hijo2,colaDel1,colaDel2,
               ptoDeCorte},
If[ (Random[]< Pc),
    ptoDeCorte=Random[Integer,{1,p-1}];
```

```
    hijo1=Take[padre1,ptoDeCorte];
    hijo2=Take[padre2,ptoDeCorte];
    colaDel1=Take[padre1,{ptoDeCorte+1,p}];
    colaDel2=Take[padre2,{ptoDeCorte+1,p}];
    hijo1 = Join[hijo1,colaDel2];
    hijo2 = Join[hijo2,colaDel1];
,(* else *)
    hijo1=padre1;
    hijo2=padre2
]; (* if *)
   Return[{hijo1,hijo2}];
]; (* fin de cruceDos *)

(* CRUZAMIENTO: Ejecuta el mdulo anterior, cruceDos, [n/2]
veces en una poblacin de tamao n.  *)

cruzamiento[pobla_]:= Module[{ip1=1,ip2=2,pT,Pad1,Pad2,hijos,
hijo1,hijo2},
          pT = pobla;
          While[ ip2<=TP ,
Pad1=pobla[[ip1]];
Pad2=pobla[[ip2]];
hijos=cruceDos[Pad1,Pad2];
hijo1=hijos[[1]];
       hijo2=hijos[[2]];
pT = ReplacePart[pT,hijo1,ip1];
pT = ReplacePart[pT,hijo2,ip2];
ip1=ip1+2;
  ip2=ip2+2;
];
            Return[pT];
];  (* fin de cruzamiento *)
```

```
(* SELECCION : Toma una poblacin y retorna otra poblacin con
los individuos seleccionados de acuerdo con el
mecanismo de la ruleta.  *)

seleccion[pobla_]:=Module[{ poblSelec={},numIndivSel,indivSelec },
  Do[
  numIndivSel=ruleta[F];
  indivSelec=pobla[[numIndivSel]];
  poblSelec = Append[poblSelec,indivSelec];
  ,{i,1,TP}
  ]; (* do *)
Return[poblSelec];
]; (* fin de seleccion *)



(* IMPRIMAPOBLACION: Imprime c/fila de la poblacion, su fitness,
calidad y exactitud *)

imprimaPoblacion[pobla_,texto_]:=Module[{},
   Print[""];  Print["Poblacin  ", texto];  Print[""];
Do[
   Print[pobla[[i]],"  ",F[[i]],"  ",Cali[[i]],"  ",Exac[[i]] ];
    ,{i,1,TP}
 ]; (* do *)
]; (* fin de imprimaPoblacion *)

buscaEsquema[pobla_]:=Module[{numCeros,numUnos,
       esquema={} },
Do[
```

```
  numUnos =0;
     Do[
        If[ ( pobla[[i,j]]==1 ),
         numUnos++;];
       ,{i,1,TP} ];
                    AppendTo[esquema,"*"];
  If[( (TP-numUnos)/TP <= Pm ),
                    esquema=ReplacePart[esquema,1,j];
             ,(* else *)
                If[( (numUnos/TP) <= Pm),
        esquema=ReplacePart[esquema,0,j]
                       ] (* if *)
                     ];(* if *)
   ,{j,1,p} ];
        Return[esquema];
];


(******* PROGRAMA PRINCIPAL *******)

(* Algoritmo Gentico Aplicado a Conjuntos Aproximados *)

AGACA[poblaux_]:=Module[{pobla,iMax,i,esq},
  pobla=poblaux;
evalToda[pobla];
Print["GenNo./MejorConj/MayorFitness/calidad/exactitud"];
Do[
    pobla=cruzamiento[pobla]; (* Cruzamientos *)
    pobla=mutaciones[pobla];  (* Mutaciones   *)
    pobla=seleccion[pobla];      (* Seleccin    *)
       evalToda[pobla];
             iMax=maximo[F];
```

```
Print[i,"/",pobla[[iMax]],"/",F[[iMax]],"/",Cali[[iMax]],"/",Exac[[iMax]] ];
           (* el mejor de la generacin t pasa a la  generacin t+1 *)
pobla= ReplacePart[pobla,pobla[[iMax]],TP];
F    = ReplacePart[F,F[[iMax]],TP];
Cali = ReplacePart[Cali,Cali[[iMax]],TP];
Exac = ReplacePart[Exac,Exac[[iMax]],TP];
 ,{i,1,MaxNumGen}
 ]; (* fin de do *)
Print[""];
   imprimaPoblacion[pobla,"final"];
Print[""];
Print["Esquema detectado: "];
         esq= buscaEsquema[pobla];
  Print[esq];

];   (* fin del Programa Principal AGACA *)

historial = {};
Print["Pruebas: "];

Do[
  Poblacion=Partition[Table[Random[Integer,{0,1}],{TP*p}],p];
  AGACA[Poblacion];
  AppendTo[historial,{Poblacion[[TP]], F[[TP]], Cali[[TP]],Exac[[TP]]} ];
  Print["prueba ", veces, " : " ,historial[[veces]] ];
  ,{veces,1,20}];
];
```

Chapter 3

Algunos árboles de decisión.

3.1 Árbol de decisión de felinos.

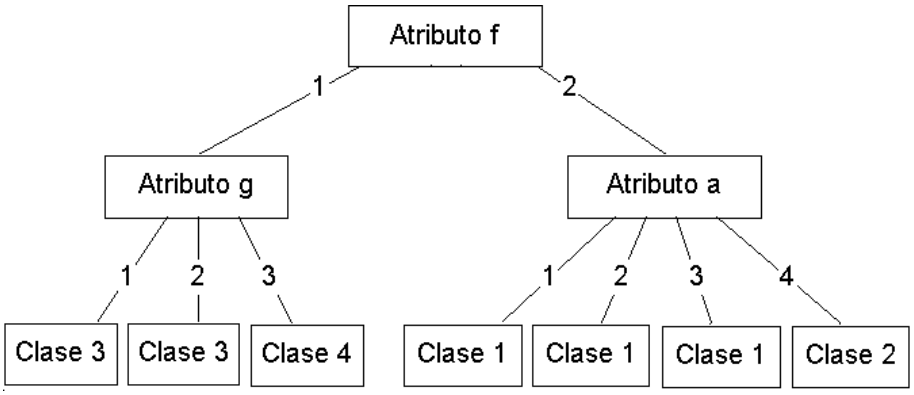

Figure 3.1: *Arbol de decisión de felinos generado por ID3 .*

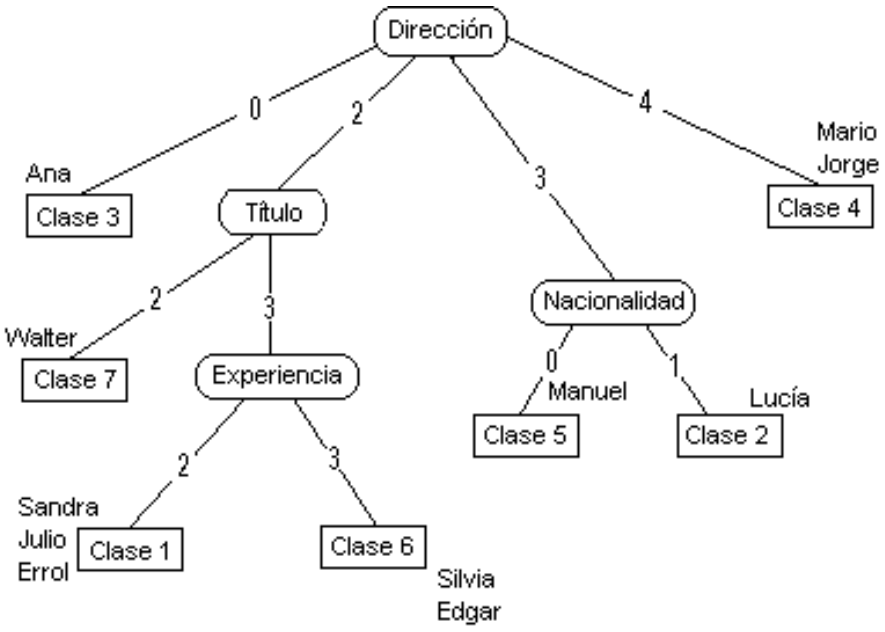

Figure 3.2: *Arbol de decisión de felinos generado por ID3 .*

Bibliography

[1] Astorga, A.(1997) *Conjuntos Aproximados: Teoría y Aplicaciones en Bases de Conocimiento y Análisis de Datos.* Tesis para optar al grado de *Magister Scientiae* en Computación, ITCR, Cartago.

[2] Astorga, A.; Espinoza, J. L.; Figueroa, G.; Mora W. (1999) "Árboles de decisión", en: IV informe, proyecto: problemas de programación matemática mediante técnicas estocásticas y neuronales, ITCR, Costa Rica.

[3] Cailliez, F.; Pagès, J.-P. (1976) *Introduction à l'Analyse des Données.* SMASH, Paris.

[4] Celleux, G. (1990) *Analyse Discriminante sur Variables Continues.* INRIA, Francia.

[5] Charon, I.; Germa, A.; Hudry, O. (1996) *Méthodes d'Optimisation Combinatoire.* Masson, París.

[6] Davis, L.; Steenstrup, M. (1987) "Genetic algorithms and simulated annealing: an overview", *Genetic Algorithms and Simulated Annealing*, L. Davis (ed.). Pitman, Londres.

[7] Diday, E.; Lemaire, J.; Pouget, J.; Testu, F. (1982). *Eléments d'Analyse de Données.* Ed. Dunod, París.

[8] Espinoza, J. L. (1997). "Partición óptima: el algoritmo de Fisher". *Revista de Matemática. Teoría y Aplicaciones.* Vol. IV (1). pp. 77-85, Costa Rica.

[9] Espinoza, J. L. (1995). "Algoritmos genéticos y optimización de funciones". Memorias del *IX Simposio de Métodos Matemáticos Aplicados a las Ciencias.* Turrialba, C. R. pp. 33-42.

[10] Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading-Mass.

[11] Goldberg, D.E.; Segrest, P. (1987) "Finite markov chain analysis of genetic algorithms". *Proceedings of the Second International Conference on Genetic Algorithms*, 1-8.

[12] Ho, T.; Nguyen, T.; Kimura, M. (1996) "Induction of decision trees using rough set theory " Japan Advanced Institute of Science and Technology. 923-12, Japón.

[13] Iosifescu, M. (1980) *Finite Markov Processes and Their Applications.* Chichester: Wiley.

[14] Karling, S.; Taylor, H. (1975). *A First Course on Stochastic Processes.* Second edition, Academic Press, N.Y.

[15] Lechevallier, Y. (1994) "Construcción eficaz de una red neuronal a partir de un árbol de decisión" *Métodos Matemáticos Aplicados a las Ciencias. VII y VIII Simposios.* Editorial de la Universidad de Costa Rica, pp. 53-102.

[16] Lerman, I. C.; Ngouenet, R. F. (1995) *Algorithmes Génétiques Séquentielles et Parallèles pour une Représentation Affine des Proximités.* Rapport de Recherche No. 2570, Janvier, INRIA, France.

[17] Marvall, D. (1974). *Cálculo de Probabilidades y Procesos Estocásticos.* Paraninfo, Madrid.

[18] Núñez, M. (1996) *Vida Artificial y Algoritmos Genéticos.* Notas de Curso, Maestría en Computación, ITCR, Costa Rica.

[19] Pastrana, J. F. (1994) "Enfoque bayesiano del análisis discriminante." *Métodos Matemáticos Aplicados a las Ciencias. VII y VIII Simposios.* Editorial de la Universidad de Costa Rica, pp. 111-116.

[20] Pao, Y. H. (1989) *Adaptative Pattern Recognition and Neural Networks.* Addison-Wesley.

[21] McCord, M.; Illingworth, W. T. (1990) *A Practical Guide to Neural Networks* . Addisson-Wesley.

[22] Pawlak, Z. (1982) "Rough sets" *Int. J. of Computer and Information Sciences*, 11(5), pp. 341-356.

[23] Pawlak, Z. (1985) *Rough Sets. Theoretical Aspects of Reasoning about Data*. Kluwer Publ. Co., Dordrecht.

[24] Pawlak, Z. (1986) "Rough classification of patients after highly selective vagotomy for duodenal ulcer " *Int. J. Man-Machine Studies*, 24, 413-433.

[25] Pawlak, Z. (1991) *Rough Sets. Theoretical Aspects of Reasoning About Data*. Kluwer Publ. Co., Dordrecht.

[26] Quinlan, J. R. (1986) "Induction of Decision Trees". *Machine Learning*, 1, 81-96.

[27] Ross, S. (1989) *Introduction to Probability Models*. Academic Press, Inc., 4th ed. U.S.A.

[28] Rudolph, G. (1994) "Convergence analysis of canonical genetic algorithms". *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, January.

[29] Saporta, G. (1990) "Introduction à la discrimination : Problematique et Méthodes" *Analyse Discriminante sur Variables Continues*. INRIA, Francia, capítulo 1.

[30] Saporta, G. (1994) "El análisis discriminante" *Métodos Matemáticos Aplicados a las Ciencias. VII y VIII Simposios.* Editorial de la Universidad de Costa Rica, pp. 75-102.

[31] Saporta, G. (1994) "Los métodos y las aplicaciones del *credit scoring* ". *Métodos Matemáticos Aplicados a las Ciencias. VII y VIII Simposios.* Editorial de la Universidad de Costa Rica, pp. 103-109.

[32] Slowinsky, R.; Stefanowski, J. (1989) "Rough classification in incomplete information systems", *Mathl. Comput. Modelling,* Vol. 12 (10-11), pp. 1347-1357.

[33] Trejos, J. (1994) *Contribution à l'Acquisition Automatique de Connaissances à Partir de Données Qualitatives.* Thèse de Doctorat, Université Paul Sabatier, Toulouse.

[34] Trejos, J. (1994) "Presentación de las redes neuronales: aplicaciones al análisis de datos" *Métodos Matemáticos Aplicados a las Ciencias. VII y VIII Simposios.* Editorial de la Universidad de Costa Rica, pp. 117-136.

[35] Trejos, J. (1996) *Métodos de Clasificación y Discriminación*. Notas de Curso, Maestría en Matemática, UCR, Costa Rica.

[36] Trejos, J.; Piza, E.; Murillo, A. (199X) "XXXX XX XXXXX ".

[37] Trejos, J.; Schektman Y. (199X) "XXXX XX XXXXX " .

[38] Trejos, J.; Villalobos, M. (199X) "XXXX XX XXXXX ".

[39] Trejos, J.; Piza; Castillo, W. (199X) "XXXX XX XXXXX ".

[40] Zethi, I. (1990) "Entropy Nets: From Decision Trees to Neural Networks", Proceedings of IEEE, vol 78.