UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

UN SISTEMA DE CLASIFICACION Y DE RECONOCIMIENTO DE PATRONES

T E S I S

QUE PARA OBTENER EL TITULO DE

M A T E M A T I C O

P R E S E N T A :

RAFAEL MORALES GAMBOA

México, D. F. 1988

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Contenido

Introducción

Los problemas de reconocimiento de patrones y de clasificación han existido siempre. Cuando miramos, escuchamos, tocamos o gustamos algo. muchas veces nuestra meta es contestar a la pregunta ¿qué es?, una manera sencilla y concisa de preguntar ¿qué *patrón*, de todos los que conozco, satisface este objeto? Los patrones que conocemos los hemos adquirido a través de nuestra vida, reconociéndolos en nuestro entorno: aprendemos que una pelota es redonda, un automóvil tiene llantas, un perro ladra, etc. Así, los problemas de reconocimiento de patrones y clasificación son del tipo de problemas que resolvemos todos los días, consciente e inconscientemente; pero ésto no significa que sean fáciles y su complejidad real se nos presenta cuando intentamos transmitir nuestra capacidad de resolverlos a una teoría matemática y, finalmente, a un dispositivo automático.

Resolver teóricamente y en la práctica los problemas de reconocimiento de patrones y de clasificación ha sido, y continua siendo, uno de los retos más importantes del desarrollo científico y tecnológico actual y se ha intentado solucionarlos desde puntos de vista muy variados. En el aspecto teórico, se han utilizado técnicas estadísticas y probabilísticas, métodos combinatorios, técnicas heurísticas, etc. El surgimiento y desarrollo de la Inteligencia Artificial tiene raices profundas en estos problemas.

La Teoría de Test constituye una aproximación a tales problemas desde la Matemática Discreta. Sus conceptos fundamentales son relativamente sencillos; pero, debido a su naturaleza combinatoria, los cálculos que requieren son de elevada complejidad.

El concepto básico de la Teoría de Test es el de *testor mínimo* para una muestra de un cierto universo de objetos y la aplicación práctica de la teoría en su estado actual requiere el cálculo de *todos* los testores mínimos para una muestra dada, los cuales deben ser localizados entre la colección de todos los subconjuntos de rasgos en términos de los cuales se describen los objetos universo en cuestión. Si tenemos en cuenta el hecho de que el número de subconjuntos crece exponencialmente con respecto al número de rasgos, podemos formarnos una idea de la dificultad de la tarea.

Se han diseñado varios algoritmos para encontrar todos los testores mínimos para una muestra de aprendizaje, que van desde algunos muy simples y "naturales" hasta otros más complejos que han necesitado para su elaboración de un análisis más profundo de la estructura interna de una muestra de aprendizaje y de las

características que distinguen a los testores mínimos de los demás subconjuntos de rasgos. En Cuba se han construido además sistemas de programas de computadora para reconocimiento de patrones y para clasificación basados en la Teoría de Test.

La presente tesis tiene dos objetivos principales. Por una parte, diseñar y probar un algoritmo para calcular los testores mínimos de una muestra dada que pueda ser utilizado en microcomputadoras con tiempos de ejecución razonables y, por otra parte, construir un pequeño sistema para reconocimiento de patrones y clasificación que lleve a la práctica los conceptos de la Teoría de Test y el nuevo algoritmo desarrollado.

En el capítulo primero se expone de manera general en qué consisten los problemas de reconocimiento de patrones y de clasificación, introduciendo la notación matemática necesaria.

En el capítulo segundo se presentan los conceptos fundamentales de la Teoría de Test, se describe el problema concreto que condujo al desarrollo de la tesis y el criterio que va a utilizarse para efectuar la clasificación de objetos.

En el capítulo tercero se expone el algoritmo diseñado para el cálculo de los testores mínimos, incluyendo las consideraciones que motivaron su diseño; se prueba que el algoritmo es correcto y se realizan una serie de estimaciones de su eficiencia, mediante una pequeño análisis de su complejidad y una tabla de comparaciones.

En el capítulo cuarto se describe el Sistema para Reconocimiento de Patrones y Clasificación (RECLA) que utiliza el algoritmo presentado en el capítulo tercero y el criterio de clasificación señalado en el capítulo segundo.

Finalmente, en el apéndice se presentan los programas en Pascal que conforman el sistema RECLA y un ejemplo de su funcionamiento.

Antes de terminar esta introducción, quiero decir que la notación utilizada y muchas de las ideas desarrolladas a lo largo de esta tesis se basan en los trabajos realizados por el Dr. José Ruíz Shulcloper y su grupo de trabajo del Instituto de Cibernética, Matemática y Física de la Academia de Ciencias de Cuba.

CAPITULO 1

Reconocimiento de Patrones y Clasificación

1.1. Un ejemplo.

Supóngase que se desea elaborar un mapa de la República Mexicana que indique los diferentes tipos de regiones que componen el territorio nacional desde el punto de vista agrícola, como base de un plan cuyo objetivo es optimizar el uso del suelo. Para la creación de dicho mapa se han considerado los diez productos agrícolas más importantes para la economía nacional, de manera que cada región de la República se relacionará en el mapa —mediante un color particular— con el cultivo que mejor se acomode a sus características. En otras palabras, lo que se desea es *clasificar* cada región del país de acuerdo al tipo de cultivo para el cual es más apta.

El primer paso consiste en *seleccionar* cuáles *características*, de entre las muchas que definen una región —como son el clima, el número de habitantes, la cercanía con la frontera, la composición del suelo, el número de estrellas que se observan a simple vista, etc.—, se tomarán en cuenta para describirla; de esta manera, cada región será *representada* por los *valores* que adquieren en ella las características escogidas.

Como información previa a la clasificación, es conveniente saber qué se necesita para obtener buenas cosechas de cada producto; en otras palabras, saber cuál es la *importancia* relativa de cada una de las características seleccionadas y cuáles valores de ellas son los apropiados para obtener buenas cosechas. Si se denota cada colección de valores adecuados como *patrón*, lo que se busca es *reconocer los patrones* asociados a cada uno de los diez productos elegidos. Una manera de obtener los patrones es analizando regiones bien conocidas por su alta productividad, para cada uno de los cultivos considerados. Póngase de ejemplo el frijol: se sabe que existen regiones del país en las cuales se obtienen excelentes cosechas de frijol; en consecuencia, se envían especialistas a cada una de ellas para adquirir información sobre los valores de las características seleccionadas y, tomando como base los datos proporcionados por los especialistas, se reconoce el patrón de la región del frijol. De la misma forma se procede con los demás productos.

Finalmente, comparando los datos de una cierta región —proporcionados por algunos especialistas enviados a ella— con los patrones reconocidos previamente, se decide a cuáles de todos ellos "satisfacen más" esos datos, obteniéndose la clasifi-

cación de la región en cuestión.

Como este ejemplo, en la realidad existen muchas situaciones que pueden identificarse como problemas de reconocimiento de patrones o de clasificación: en el diagnóstico médico, en el análisis de la escritura, en la prospección geológica, en el reconocimiento de imágenes, etc. Para solucionarlos, resulta conveniente abstraer de todos ellos formulaciones generales que puedan ser llevadas a expresiones matemáticas precisas. Posteriormente, una vez resueltos los problemas en su forma general, se adecúan las soluciones encontradas a los problemas prácticos originales.

1.2. Planteamiento general.

Sobre la base de las situaciones reales es posible formular de manera general los problemas de reconocimiento de patrones y de clasificación (de aquí en adelante, características, *rasgos* y *variables* se emplearán como sinónimos):

Por *reconocimiento de patrones* se entiende un proceso mediante el cual se puede determinar la importancia de cada uno de los rasgos, y sus valores correspondientes, que caracterizan cada una de las *clases* de *objetos* que conforman un cierto *universo*.

Como ejemplo, en el análisis de la escritura manuscrita, el proceso de reconocimiento de patrones permite establecer la importancia de las características y los valores correspondientes que hacen que una letra sea una *a* y no una *c*.

Por *clasificación* se entiende el proceso mediante el cual, dados un cierto universo de objetos dividido en clases (no necesariamente ajenas entre sí) y un miembro de él, se puede decidir la pertenencia o no pertenencia de dicho objeto a cada una de las clases que componen el universo.

Retomando el caso del análisis de la escritura manuscrita, dada una letra, el proceso de clasificación permite decidir qué letra es, esto es, determinar a cuál de todas las clases de letras pertenece.

Los problemas de reconocimiento de patrones y de clasificación consisten, por una parte, en *elaborar* algoritmos de reconocimiento y de clasificación eficaces y, por otra, en *aplicar* dichos algoritmos en el estudio de universos concretos, como pueden ser las regiones agrícolas de la República, los cardiópatas mexicanos, ciertas imágenes digitales, etcétera.

Como es de esperarse, la dificultad de la tarea depende fuertemente de la cantidad de información disponible. Por ejemplo, si se conocen todos los objetos del universo en cuestión y, para cada uno de ellos, la clase a la que pertenecen entonces el proceso de clasificación puede reducirse a buscar en un archivo con

los datos; mientras que si se carece completamente de información inicial puede necesitarse un proceso de aprendizaje previo al reconocimiento y la clasificación. En general, se señalan tres variantes de los problemas (Ruíz,84):

a) cuando se conoce el número de clases en que se divide el universo de objetos y de cada clase se conoce una *muestra*;

b) cuando se conoce el número de clases, sin ninguna otra información adicional;

c) cuando se desconoce desde el número de clases en que se descompone el universo de objetos.

La dificultad para encontrar una solución óptima —lo que ésto signifique en cada caso— aumenta conforme disminuye la cantidad de información disponible. Existen métodos, como el análisis de cúmulos, para el tratamiento de problemas del tipo (b) y (c). En particular, el presente trabajo se limita a problemas del tipo (a).

Es importante aclarar que existen muchas definiciones de los procesos de reconocimiento de patrones y de clasificación distintas a las presentadas aquí; más aún, en ocasiones suelen utilizarse como sinónimos, lo cual ha creado confusión sobre el significado de palabras como "reconocimiento", "clasificación" y "patrón". Se optó por las definiciones enunciadas porque se adaptan bien a los problemas que se derivan de la información inicial sobre el universo de objetos: si se tiene una muestra de cada una de las clases que componen el universo, es más o menos natural preguntarse ¿qué conocimiento sobre las clases puede extraerse de sus muestras? y ¿cómo se puede determinar la clase a la que pertenece un nuevo objeto a partir de las muestras disponibles? La primera pregunta conduce a la definición de reconocimiento de patrones y la segunda a la de clasificación tal como se dieron antes.

1.3. Hipótesis adicionales.

Los problemas de reconocimiento de patrones y clasificación, tal como fueron planteados en la sección anterior, son demasiado generales y vagos para ser utilizados como base de un estudio más profundo. Es necesario añadir algunas hipótesis restrictivas con el propósito de atenuar la dificultad de los problemas y hacer su expresión mucho más precisa, susceptible de una formulación matemática posterior. Una de las hipótesis más importantes se refiere a la manera como se van a representar los objetos; por ejemplo, considerando cada objeto del universo como definido a partir de un número fijo de variables, es fácil imaginar el universo de objetos como un subconjunto de algún espacio de dimensión finita; por otra parte,

considerando un objeto como descrito por una cadena de símbolos —que representan valores de las características o acciones para la construcción del objeto— de longitud arbitraria aunque finita, entonces puede concebirse el universo como un lenguaje y sus miembros como frases en el mismo.

En el marco de este trabajo, se consideran los objetos como puntos en un espacio de dimensión finita, bajo una serie de hipótesis adicionales, las primeras de las cuales se refieren a la naturaleza de los objetos que pueden estudiarse:

a) los objetos del universo pueden ser descritos en términos de un número finito de rasgos;

b) todos los objetos se definen utilizando los mismos rasgos;

c) cada rasgo puede ser medido y su rango de valores es finito.

Las siguientes hipótesis se refieren a la información disponible sobre el universo de objetos:

d) se conoce el número de clases en que se descompone el universo;

e) se tiene una muestra de cada una de las clases;

f) las muestras son ajenas entre sí;

g) puede desconocerse el valor de algunos rasgos en varios de los objetos de las muestras.

Las hipótesis (d) y (e) indican la *cantidad* de información previa, mientras que las hipótesis (f) y (g) indican la *calidad* de dicha información; la hipótesis (f) es restrictiva, mientras que la hipótesis (g) amplía la colección de problemas tratables al permitir "hoyos" en la información inicial.

Dicho lo anterior, el siguiente paso consiste en expresar los problemas de una manera precisa utilizando la notación matemática necesaria.

1.4. Planteamiento formal.

Para comenzar, sea N el número de rasgos en términos de los cuales serán descritos los objetos. A cada rasgo se le hace corresponder un índice en $\mathcal{R} = \{1, \ldots, N\}$, estableciendose de esta manera una relación de orden entre los rasgos; de aquí en adelante se hará referencia a los rasgos simplemente por su índice. Se denotará por $\mathcal{V}_j = \{0, \ldots, v_j, *\}$ el *conjunto de valores* del rasgo j con $j = 1, \ldots, N$, donde $*$ significa "desconocido", "indeterminado" o "ausencia de información".

1.4.1.Definición. *Un conjunto* Ω *es* una población *si*

$$\Omega = \bigcup_{i=1}^{r} \Omega_i \qquad con \qquad \Omega_i \subset \mathcal{V}_1 \times \cdots \times \mathcal{V}_N$$

y donde r *es el número de clases que componen a la población. A cada* Ω_i *se le llama* clase *de la población y a cada* $\omega \in \Omega$ *la* descripción *de un objeto (en general, se llama descripción de un objeto a cualquier elemento de* $\mathcal{V}_1 \times \cdots \times \mathcal{V}_N$*). Si* $\omega = (\omega_1, \ldots, \omega_N)$ *entonces a* ω_j *se le denomina el* valor del rasgo j *en* ω*; una descripción* ω *es* completa *si* $\omega_j \neq *$ *para* $j = 1, \ldots, N$ *e* incompleta *en otro caso.*

1.4.2.Definición. *Sea* $X \subset \mathcal{R}$ *un subconjunto de rasgos. Se dice que dos descripciones* ω *y* ω' *son* semejantes con respecto a X*, lo cual se denota por* $\omega \overset{X}{\sim} \omega'$*, si y sólo si* $\omega_j \neq \omega'_j$ *implica* $\omega_j = *$ *o* $\omega'_j = *$*, para todo* $j \in X$*. Si* $X = \mathcal{R}$ *entonces se dice que las descripciones son* semejantes *y se denota por* $\omega \sim \omega'$[1].

A la información disponible sobre el universo de objetos se le llamará *muestra de aprendizaje* y se define a continuación de acuerdo con las hipótesis establecidas en la sección anterior.

1.4.3.Definición. *Una* muestra de aprendizaje *de la población* Ω *es un subconjunto* $\Omega' \subset \Omega$ *tal que:*

a) $\Omega' = \bigcup_{i=1}^{r} \Omega'_i$ *con* $\Omega'_i \neq \emptyset$ *y* $\Omega'_i \subset \Omega_i$ *para* $i = 1, \ldots, r$*;*

b) si $\omega^i \in \Omega'_i$ *entonces no existe ningún* $\omega^k \in \Omega'_k$ *tal que* $\omega^i \sim \omega^k$ *para* $i \neq k$*.*

Dicho más llanamente, una muestra de aprendizaje es un subconjunto de la población tal que cada elemento pertenece a una sola clase y cada clase está representada por al menos un elemento.

Una vez expresado formalmente lo que se entiende por una población, una descripción de un objeto y establecida la estructura de la información disponible —bajo el concepto de muestra de aprendizaje—, quedan por definir de manera precisa los objetivos del reconocimiento de patrones y de la clasificación.

1.4.4.Definición. *Por un* vector diferenciante *se entiende un vector*

$$\varrho = (\varrho_1, \ldots, \varrho_N) \qquad con \qquad 0 \leq \varrho_j \leq 1,$$

[1] La definición puede expresarse de manera más general utilizando *operadores de diferencia* $\delta_j : \mathcal{V}_j \times \mathcal{V}_j \to \{0, 1\}$, $j = 1, \ldots, N$, de la siguiente manera: *dos descripciones* ω *y* ω' *son semejantes con respecto a* X *si y sólo si* $\delta_j(\omega_j, \omega'_j) = 0$ *para todo* $j \in X$. En este caso $\mathcal{V}_j$ puede ser cualquier conjunto, no necesariamente finito.

donde la componente ϱ_j se denomina el peso diferenciante *del rasgo j.*

El peso diferenciante de un rasgo es la expresión matemática de la noción intuitiva de la "importancia de un rasgo" para distinguir entre clases —véase la sección (2.2)—: un peso alto, cercano a uno, indica una característica esencial para la descripción de los objetos, cuyos valores deben tomarse muy en cuenta en el estudio de la población; mientras que un peso diferenciante bajo, cercano a cero, habla de una característica poco importante y que en ciertos casos puede despreciarse —dependiendo del problema particular.

El problema de reconocimiento de patrones consiste, dada una muestra de aprendizaje $\Omega' = \bigcup_{i=1}^{r} \Omega'_i$, en:

a) encontrar un vector diferenciante $\varrho = (\varrho_1, \ldots, \varrho_N)$, donde la componente ϱ_j indique la importancia del rasgo j para la muestra de aprendizaje;

b) escoger de cada muestra de clase Ω'_i un subconjunto $P_i \subset \Omega'_i$ de descripciones que sean *representativas* de la muestra, según un cierto criterio —véase la sección (2.3).

Por supuesto, se busca un vector diferenciante y una colección de subconjuntos de descripciones lo más exactos posible, esto es, que reflejen fielmente las peculiaridades de las clases que componen la población.

Por otra parte, considérese la función

$$\chi : \Omega \to [0,1]^r$$

tal que

$$\chi(\omega)_i = \begin{cases} 1 & \text{si } \omega \in \Omega_i \\ 0 & \text{si } \omega \notin \Omega_i. \end{cases}$$

El problema de clasificación consiste en construir una función

$$A : \Omega \to [0,1]^r$$

tal que

$$A(\omega)_i = \begin{cases} 1 & \text{signifique } \omega \in \Omega_i \\ 0 & \text{signifique } \omega \notin \Omega_i \\ x \in (0,1) & \text{signifique que no se pudo clasificar a } \omega. \end{cases}$$

que "aproxime" a χ, según un cierto criterio de proximidad o de lejanía —véase la sección (2.4). A $A(\omega)$ se le llama el *vector de clasificación* de ω con respecto

al *algoritmo de clasificación A*. Por supuesto, se pretende que el algoritmo de clasificación sea fácil y rápido de calcular.

Queda así expresado de manera formal lo que se entiende por reconocimiento de patrones y por clasificación en el contexto de este trabajo. En el capítulo siguiente se presentan una serie de conceptos, agrupados bajo el nombre de *Teoría de Test*, que se proponen como una posible solución a los problemas planteados.

CAPITULO 2

Teoría de Test

2.1. Características generales de la Teoría de Test.

Existen muchas aproximaciones a los problemas de reconocimiento de patrones y de clasificación, que van desde los métodos estadísticos y probabilísticos hasta los métodos de la Matemática Discreta y de la Inteligencia Artificial; cada uno tiene ventajas y desventajas importantes. Por ejemplo, las técnicas estadísticas y probabilísticas están respaldadas por una teoría bien fundamentada y potente, la cual permite una evaluación precisa de ellas; por su parte, las técnicas de la Inteligencia Artificial conducen muchas veces a mejores resultados, con más rapidez y precisión, pero su justificación formal es menos sólida en ciertos casos. En lo que a la Matemática Discreta se refiere, ha tomado auge en la práctica a últimas fechas debido, entre otras cosas, a la existencia de computadoras cada vez más potentes y su aplicación a la solución de problemas no numéricos; sus métodos pueden considerarse casos intermedios entre los anteriores, ya que están sostenidos por una teoría precisa más no siempre resultan eficientes y su análisis es en ocasiones muy complejo.

La Teoría de Test constituye una aproximación a los problemas de reconocimiento de patrones y de clasificación desde el punto de vista de la Matemática Discreta, lo cual significa que trabaja —esencialmente— con conjuntos finitos de valores utilizando métodos combinatorios.

Las aportaciones más importantes de la Teoría de Test se relacionan con el reconocimiento de patrones, especialmente con la determinación de la importancia o peso diferenciante de los rasgos que definen a los objetos de un cierto universo. En lo que se refiere al proceso de clasificación, se han modificado algoritmos conocidos introduciendo parámetros adicionales —como los pesos diferenciantes de los rasgos— o modificaciones aún más profundas —como la sustitución de métodos estadísticos por métodos combinatorios— y se han desarrollado también algoritmos originales.

2.2. Conceptos básicos.

Supóngase que se tiene una muestra de aprendizaje de una población. Por definición, esto significa, entre otras cosas, que no existen descripciones de objetos semejantes

en dos muestras de clases distintas. ¿Qué sucede si se ignora un rasgo, es decir, si se supone a los objetos descritos por todos los demás rasgos menos ese? Pueden darse dos casos, según aparezcan o no descripciones semejantes en muestras de clases distintas. En el primer caso se puede pensar que la característica ignorada es importante ya que es necesaria para distinguir entre descripciones de objetos pertenecientes a clases distintas, mientras que en el segundo caso puede intuirse que dicho rasgo tiene poca importancia pues su ausencia no altera la clasificación de las descripciones en la muestra de aprendizaje. Si se continua eliminando rasgos —siempre en el segundo caso— llegará un momento en que no será posible ignorar ningún otro, ya sea porque no queden más o porque aparezcan descripciones semejantes en muestras de clase distintas independientemente del rasgo que se elimine. Intuitivamente, puede considerarse los rasgos que quedan como "muy importantes" y los eliminados como "poco importantes"; sin embargo, siguiendo otro orden de eliminación es muy probable que se llegue a otro subconjunto de características que no pueden ignorarse, lo cual nos lleva a considerar *todos* los subconjuntos de rasgos que se pueden obtener por este método y determinar el peso diferenciante de cada rasgo de acuerdo al número de tales subconjuntos a los que pertenece.

Este es el argumento que yace bajo la definición que da la Teoría de Test del peso diferenciante de un rasgo. Como puede observarse, es un razonamiento sencillo; sin embargo, como se verá más adelante, esconde dificultades prácticas interesantes.

2.2.1.Definición. *Dada una muestra de aprendizaje* $\Omega' = \bigcup_{i=1}^{r} \Omega'_i$, *un conjunto de rasgos* X *es un* testor *si no existe una pareja de descripciones* $\omega^i \in \Omega_i$ *y* $\omega^k \in \Omega_k$ *con* $i \neq k$ *que sean semejantes con respecto a* X.

2.2.2.Definición. *Un conjunto* X *de rasgos es un* testor mínimo *si no existe un subconjunto propio* $X' \subset X$ *que sea un testor.*

2.2.3.Definición. *El* peso diferenciante ϱ_j *del rasgo* j *para la muestra de aprendizaje* Ω' *es*

$$\varrho_j = \frac{T_j}{T}$$

donde T_j *es el número de testores mínimos de* Ω' *que contienen el rasgo* j *y* T *es el número total de testores mínimos de* Ω'.

Esta es una propuesta de solución (Ruíz,84) para la primera parte del problema de reconocimiento de patrones —véase la sección (1.4). No es la única forma de medir el peso diferenciante de una característica dentro de la Teoría de Test; pero es la más utilizada y se usará a lo largo de este trabajo.

2.3. Peso informativo de la descripción de un objeto.

En el capítulo anterior se estableció que el objetivo del reconocimiento de patrones consiste en determinar el peso diferenciante de los rasgos que describen a los objetos y en escoger, de cada una de las muestras de clase que conforman a la muestra de aprendizaje, un subconjunto de descripciones de objetos "representativas" de su clase. La medida de la "representatividad" de la descripción de un objeto que propone la Teoría de Test se denota como *peso informativo* de la descripción, el cual se define como sigue:

2.3.1.Definición. *Sean Ω' una muestra de aprendizaje de una población Ω, ϱ un vector diferenciante, ω una descripción de un objeto en Ω y $X_\omega = \{ j \in \mathcal{R} \mid \omega_j \neq * \}$. El* peso informativo de ω con respecto a la clase $\Omega_i \subset \Omega$ es

$$\iota_i(\omega) = \sum_{j \in X_\omega} c_i(\omega_j)\varrho_j$$

donde $c_i(\omega_j)$ es el número de elementos en $S_i(\omega_j) = \{ \omega' \in \Omega'_i \mid \omega'_j = \omega_j \}$ entre el total de elementos de Ω'_i[1].

2.3.2.Definición. *Una descripción $\omega \in \Omega$ se dice que es un* ideal *de la clase Ω_i si y sólo si*

$$\iota_i(\omega) = \max\{ \iota_i(\omega') \mid \omega' \in \Omega'_i \}.$$

La propuesta de solución para la segunda parte del problema de reconocimiento de patrones —véase la sección (1.4)— es evidente y consiste en tomar

$$P_i = \{ \omega' \in \Omega'_i \mid \omega' \text{ es un ideal de la clase } \Omega_i \} \qquad i = 1, \ldots, r.$$

2.4. Un algoritmo de clasificación.

Como se dijo en la sección (2.1), dentro de la Teoría de Test se utilizan varios algoritmos de clasificación, algunos de los cuales son versiones modificadas de algoritmos más o menos conocidos, mientras que otros son resultados originales. En

[1] Igual que la definición (1.4.2), ésta también puede generalizarse con la introducción de operadores de diferencia y definiendo

$$S_i(\omega_j) = \{ \omega' \in \Omega'_i \mid \delta_j(\omega'_j, \omega_j) = 0 \}$$

este trabajo se presenta un algoritmo de clasificación basado en los ideales de clase definidos en la sección anterior, que es una versión modificada del *algoritmo del vecino más cercano* (Meisel,70). Las razones por las que se eligió este algoritmo son las siguientes:

a) ejemplifica la utilidad de la información obtenida en el proceso de reconocimiento de patrones;

b) su formulación es relativamente sencilla;

c) la cantidad de operaciones que requiere es relativamente pequeña —comparada, por ejemplo, con la de algoritmos que utilizan toda la muestra de aprendizaje.

Sin embargo, este algoritmo tiene sus desventajas, entre las que se encuentran el riesgo de pérdida de información, pues trabaja con un subconjunto de la muestra de aprendizaje. Por supuesto, la manera de elegir ese subconjunto —utilizando los pesos informativos de las descripciones— trata de evitar dicha pérdida y eliminar el ruido introducido por información superflua en la muestra de aprendizaje; pero no se tiene una garantía absoluta.

Supóngase definida una cierta *función de cercanía* entre un elemento de la población y un subconjunto de ella,

$$g : \Omega \times P(\Omega) \to [0, 1]$$

—el valor de $g(\omega, \Omega')$ es mayor entre más "cerca" se encuentre ω de Ω'— y sean $np, p \in [0, 1]$ dos números fijos con $np < p$.

2.4.1.Definición. *Si $\{P_i\}_{i=1}^{r}$ es la colección de subconjuntos de descripciones definida en la sección (2.3), $\omega \in \Omega$ y $g_0 = \max\{ g(\omega, P_i) \mid 1 \leq i \leq r \}$, entonces el vector de clasificación $A(\omega)$ es*

$$A(\omega)_i = \begin{cases} 1 & si\ g_0 = 0\ o\ \dfrac{g(\omega, P_i)}{g_0} \geq p. \\ 0 & si\ \dfrac{g(\omega, P_i)}{g_0} \leq np. \\ \dfrac{g(\omega, P_i)}{g_0} & en\ otro\ caso. \end{cases}$$

La definición anterior es muy general y, por consiguiente, es posible refinarla añadiendo detalles tales como una expresión particular de la función de cercanía y los valores correspondientes de los *umbrales de pertenencia* np y p.

Una expresión para la función de cercanía puede obtenerse de la manera siguiente:

i) Se define una seudodistancia[1] en cada uno de los conjuntos de valores de los rasgos, por ejemplo:

$$\partial_j : V_j \times V_j \to [0,1]$$

con

$$\partial_j(x,y) = \begin{cases} 0 & \text{si } x = * \text{ o } y = *. \\ \frac{1}{v_j}|x-y| & \text{en otro caso.} \end{cases}$$

ii) A partir de ellas, se define una seudodistancia en Ω, por ejemplo:

$$\partial : \Omega \times \Omega \to [0,1]$$

con

$$\partial(\omega,\omega') = \frac{1}{N}\sum_{j=1}^{N} \partial_j(\omega_j,\omega'_j).$$

iii) Se define la función de cercanía tomando como base la seudodistancia en Ω, por ejemplo:

$$g : \Omega \times P(\Omega) \to [0,1]$$

con

$$g(\omega,\Omega') = 1 - min\{\, d(\omega,\omega') \mid \omega' \in \Omega' \,\}.$$

Esta es la propuesta de solución para al problema de clasificación, tal y como fue planteado en el capítulo anterior.

2.5. Dificultades prácticas.

La definición del peso diferenciante de un rasgo que da la Teoría de Test presenta considerables dificultades prácticas, centradas en un punto: encontrar *todos* los

[1] Por una *seudodistancia* se entiende una función

$$\partial : X \times X \to \mathbb{R}$$

que satisface las siguientes condiciones:

i) $\partial(x,y) \geq 0$,

ii) $\partial(x,x) = 0$, y

iii) $\partial(x,y) = \partial(y,x)$,

para cualesquiera $x, y \in X$.

testores mínimos de una muestra de aprendizaje. La razón de las dificultades estriba —expresado burdamente— en el hecho de que el espacio de búsqueda, dada una muestra de aprendizaje con N rasgos, es el conjunto potencia de $\{1,\ldots,N\}$, que contiene 2^N elementos, lo que representa una cantidad gigantesca de posibilidades para un número razonable de rasgos.

Es claro que el algoritmo consistente en revisar todos los subconjuntos de rasgos para determinar cuáles de ellos son testores mínimos es muy ineficiente. Se han hecho intentos para hallar mejores soluciones al problema (Bravo,82) usando técnicas variadas. En Cuba se ha estado trabajando sobre el concepto de *conjunto básico* (Soto,81). Un *conjunto de diferencias* entre dos descripciones de objetos es el conjunto de rasgos en que esas descripciones difieren y un conjunto básico es un conjunto de diferencias que no contiene a ningún otro de tales conjuntos. La meta es reducir considerablemente el total de subconjuntos de rasgos que necesiten ser revisados para encontrar todos los testores mínimos y se han obtenido varios algoritmos que trabajan sobre los conjuntos básicos en vez de trabajar sobre las descripciones en la muestra de aprendizaje (Bravo,83; Ruíz,84). Sin embargo, el problema está lejos de ser resuelto y la búsqueda de mejores algoritmos continua.

CAPITULO 3

Algoritmo para encontrar los Testores Mínimos

En este capítulo se expone el algoritmo desarrollado para el cálculo de los testores mínimos de una muestra de aprendizaje. La exposición está organizada en tres partes: explicación de las motivaciones que condujeron el desarrollo del algoritmo, presentación de algunos intentos previos y, finalmente, planteamiento formal y prueba del algoritmo.

3.1. Consideraciones previas.

Como se dijo al final del capítulo anterior, la tarea de encontrar los testores mínimos de una muestra de aprendizaje con un número razonable de rasgos y descripciones de objetos no es fácil de llevar a cabo, lo cual se debe principalmente a la amplitud del espacio de búsqueda. El que podríamos llamar un "algoritmo natural" (Bravo,82) —que consiste en tomar el conjunto total de rasgos e ir eliminando sucesivamente cada uno de ellos, tomando recursivamente subconjuntos cada vez más pequeños— resulta ineficiente, lo que ha originado el desarrollo de otros algoritmos, los cuales pueden clasificarse en dos grupos:

1) los que trabajan directamente sobre la muestra de aprendizaje;
2) los que trabajan sobre la colección de los conjuntos básicos —véase la sección (2.5).

Estos grupos se dividen a su vez en dos subgrupos:

a) los *algoritmos de escala exterior*, que utilizan técnicas combinatorias para construir los subconjuntos de rasgos candidatos a testores mínimos, sin tomar en cuenta la estructura interna de la información disponible;
b) los *algoritmos de escala interior*, que construyen los candidatos a testores mínimos apoyándose en la estructura interna de la información.

Dada la naturaleza mecánica del cálculo de los testores mínimos, lo que se busca al diseñar algoritmos eficaces es poder escribir programas para computadoras que hagan los cálculos automáticamente. Desde este punto de vista, los algoritmos del grupo (2) se caracterizan por ser más rápidos; pero también por solicitar más recursos de la computadora, especialmente memoria.

En ese contexto, la idea central de este trabajo fue, en un principio, diseñar un algoritmo que resultara tanto o más rápido que los algoritmos del grupo (2) y que solicitara menos recursos del sistema de cómputo. Para lograrlo, se pensó en utilizar únicamente la muestra de aprendizaje, atendiendo a su estructura interna. En otras palabras, se pretendía reunir las mejores cualidades de cada uno de los dos grupos de algoritmos en uno solo —trabajos semejantes se han llevado a cabo en la Unión Soviética (Diukova,76; Aguila,80).

3.2. Primeros intentos.

Dos fueron los intentos previos más importantes, si bien en el primero de ellos no se avanzó mucho en su diseño:

- Se pensó en utilizar *técnicas heurísticas* para localizar los testores mínimos, es decir, construir un algoritmo "inteligente". Sin embargo, la naturaleza del problema no pareció adecuada para emplear métodos heurísticos, por una razón muy simple: se necesita calcular *todos* los testores mínimos. No se quiere decir con ésto que ese hecho haga imprescindible una búsqueda exhaustiva, comprobando cada uno de los posibles subconjuntos de rasgos; mas en cierta medida requiere de una búsqueda demasiado amplia, metódica y precisa, para que una técnica heurística resulte eficiente.
- Se pensó en construir los candidatos a testores mínimos por acumulación de rasgos, utilizando el criterio siguiente: los rasgos se seleccionan en forma creciente y sólo se añade un nuevo rasgo al candidato a testor si éste no constituye ya un testor y si se comprueba que el nuevo rasgo aporta "información relevante" —en el sentido de que ayuda a distinguir mejor entre descripciones de objetos pertenecientes a muestras de clases distintas. Entre las dificultades que surgieron en este intento sobresalen dos:

  a) se requería de mucho espacio en memoria de computadora —o su equivalente en tiempo— para determinar el estado actual del candidato a testor y, por consiguiente, para decidir si un nuevo rasgo aportaba más información;

  b) el proceso de añadir un rasgo nuevo al candidato a testor era muy lento, debido sobre todo al número de rasgos que se analizaban antes de encontrar uno adecuado.

Después de algún tiempo, resultó claro que el nuevo algoritmo debía poseer dos características importantes, a saber: *utilizar una búsqueda metódica y precisa* y *analizar el mayor número de rasgos posibles en cada paso*. El resultado se presenta en el resto del capítulo.

3.3. Planteamiento formal.

Como se recordará, se tiene una cierta población $\Omega = \bigcup_{i=1}^{r} \Omega_i$ que es un subconjunto de $\mathcal{V}_1 \times \cdots \times \mathcal{V}_N$, donde $\mathcal{V}_j = \{0, 1, \ldots, v_j, *\}$ es el conjunto de valores admisibles para el rasgos j, con $j = 1, \ldots, N$ —se adoptó la convención de denotar los rasgos por sus índices—; de esa población se conoce una muestra de aprendizaje $\Omega' = \bigcup_{i=1}^{r} \Omega'_i$ con $\Omega'_i \subset \Omega_i$ para $i = 1, \ldots, r$. La meta es diseñar un algoritmo que permita encontrar todos los testores mínimos de la muestra de aprendizaje, el cual debe poseer las características siguientes:

1) trabajar con la muestra de aprendizaje;
2) atender a la estructura interna de la muestra en la construcción de los candidatos a testores mínimos;
3) tomar en cuenta, en cada paso, tantos rasgos como sea posible;
4) poder ser utilizado en microcomputadoras para resolver problemas de buen tamaño —especialmente los que involucran un número considerable de rasgos— en un tiempo razonable[2].

La idea guía del algoritmo consiste en lo siguiente: primero, en introducir una *relación de orden* entre los subconjuntos de rasgos; a continuación, se debe *recorrer en orden* el conjunto de todos los subconjuntos de rasgos —que se denotará por $P(\mathcal{R})$— manteniendo una *lista de los testores más pequeños* que se vayan encontrando; de manera que, al finalizar el recorrido de $P(\mathcal{R})$, se tenga en la lista únicamente los testores mínimos.

Como ya se dijo, un algoritmo que analice uno por uno todos los subconjuntos de rasgos es de lo más ineficiente; por consiguiente, es necesario implementar maneras de "saltar" sobre los subconjuntos de rasgos que no sean testores y, mejor aún, sobre los que no sean testores mínimos. Es indispensable tener un *criterio* para decidir cuándo un subconjunto de rasgos es un testor mínimo y cuándo no lo es; más aún, es conveniente que dicho criterio sea fácil y rápido de calcular.

La parte más importante del algoritmo consiste en una *función de recorrido*

$$\varphi : P(\mathcal{R}) \to P(\mathcal{R})$$

creciente, que determina la forma en que se recorre $P(\mathcal{R})$ escogiendo todos los testores mínimos; es decir, si φ^n denota la n-ésima composición de la función y φ^0

[2] En la actualidad, parece ser que cincuenta es un número considerable de rasgos y doce horas es un tiempo razonable para resolver problemas de ese tamaño (en una microcomputadora). Por supuesto, existen casos en los que el número de rasgos es mucho mayor; por ejemplo, en el análisis de algunas imágenes digitales.

la identidad,

$$\Phi \equiv \{ X \subset \mathcal{R} \mid X = \varphi^n(\mathcal{R}) \text{ con } n \in \mathbb{N} \}$$

y

$$TM \equiv \{ X \subset \mathcal{R} \mid X \text{ es un testor mínimo} \},$$

entonces

$$TM \subset \Phi.$$

La función φ desarrollada en este trabajo se basa en el algoritmo de búsqueda conocido como *back-track* —véase por ejemplo (Wirth,76)—, el cual se modifica para permitir los saltos sobre aquellos subconjuntos de rasgos que se sabe —por el criterio insinuado anteriormente— no son testores mínimos.

3.4. Una relación de orden en $P(\mathcal{R})$.

Como se dijo antes, el primer paso hacia la elaboración del algoritmo consiste en definir una relación de orden en $P(\mathcal{R})$, el conjunto de todos los subconjuntos de rasgos. Si $X = \{x_1, \ldots, x_n\} \subset \mathcal{R}$ es un subconjunto de rasgos entonces se denotará

$$X = [x_1 \mid X'] \qquad \text{con} \qquad X' = \{x_2, \ldots, x_n\}.$$

Además $[\,] = \emptyset$.

3.4.1.Definición. *Sea* $\preceq$ *una relación en* $P(\mathcal{R})$ *tal que*

a) $X \preceq \emptyset$ *para todo* $X \in P(\mathcal{R})$;

b) si $X = [x \mid X']$, $Y = [y \mid Y']$ *y* $x < y$ *entonces* $X \preceq Y$;

c) si $X = [x \mid X']$, $Y = [x \mid Y']$ *y* $X' \preceq Y'$ *entonces* $X \preceq Y$;

d) nada más[3].

El siguiente resultado se desprende inmediatamente de la definición y es importante, pues significa que un testor mínimo no puede estar antes —con respecto a la relación— de un testor que lo contenga, lo que permite un manejo más eficiente de la lista de los testores más pequeños encontrados comentada en la sección anterior.

[3] Para mayor información sobre este tipo de definiciones véase (Wand,81). La relación de orden es la inducida por el algoritmo *back-track* al recorrer $P(\mathcal{R})$ eliminando rasgos en orden decreciente, como se verá más adelante.

3.4.2.Proposición. *Si $Y \subset X$ entonces $X \preceq Y$.*

Dem:

a) si $Y = \emptyset$ entonces $X \preceq Y$ por la condición (a) de la definición;

b) si $y_1 \neq x_1$ entonces $x_1 < y_1$ y la relación se cumple por la condición (b) de la definición;

c) si $y_1 = x_1$ entonces $Y' \subset X'$ y la relación se cumple por inducción y la condición (c) de la definición. q.e.d

A continuación, se prueba que la relación $\preceq$ está definida para cualquier pareja de elementos de $P(\mathcal{R})$, condición necesaria para que sea un orden total.

3.4.3.Proposición. *Si $X, Y \in P(\mathcal{R})$ entonces $X \preceq Y$ o bien $Y \preceq X$.*

Dem: Sean $X, Y \in P(\mathcal{R})$.

a) Si $X = \emptyset$ entonces $Y \preceq X$ y de la misma manera $X \preceq Y$ si $Y = \emptyset$.

b) Si $X = [x \mid X']$, $Y = [y \mid Y']$ y $x < y$ entonces $X \preceq Y$ y de la misma manera si $y < x$ entonces $Y \preceq X$.

c) Si $x = y$ entonces por inducción $X' \preceq Y'$ y entonces $X \preceq Y$, o bien $Y' \preceq X'$ y entonces $Y \preceq X$. q.e.d

Lo que resta es probar que $\preceq$ cumple con las condiciones de una relación de orden, a saber, que es *reflexiva*, *antisimétrica* y *transitiva*.

3.4.4.Proposición. *La relación $\preceq$ es un orden total en $P(\mathcal{R})$.*

Dem: Es reflexiva: $\emptyset \preceq \emptyset$ por la condición (a) de la definición y $[x \mid X'] \preceq [x \mid X']$ se obtiene aplicando inducción y la condición (c) de la definición.

Es antisimétrica: Sean $X, Y \in P(\mathcal{R})$ tales que $X \preceq Y$ e $Y \preceq X$. Si $X = \emptyset$ entonces por la condición (a) de la definición se tiene $Y = \emptyset$. Si $X \neq \emptyset$ entonces sólo se aplica la condición (b) de la definición teniendose $x_1 = y_1$ con $X' \preceq Y'$ e $Y' \preceq X'$; por inducción se tiene $X' = Y'$ de donde se obtiene finalmente que $X = Y$.

Es transitiva: Sean $X, Y, Z \in P(\mathcal{R})$ tales que $X \preceq Y$ e $Y \preceq Z$. El caso $Z = \emptyset$ es claro; si $Y = \emptyset$ entonces por la antisimetría de la relación y la condición (a) de la definición se tiene $Z = \emptyset$ y no hay más que hacer; análogamente para $X = \emptyset$. Queda únicamente el caso en que $X = [x \mid X']$, $Y = [y \mid Y']$ y $Z = [z \mid Z']$:

a) si $x < y$ e $y \leq z$ entonces $x < z$ y por lo tanto $X \preceq Z$;

b) si $x = y$ e $y < z$ entonces $x < z$ y por lo tanto $X \preceq Z$;

c) si $x = y$ e $y = z$ entonces $X' \preceq Y'$ e $Y' \preceq Z'$; por inducción se tiene $X' \preceq Z'$ y por lo tanto $X \preceq Z$. q.e.d

Los resultados anteriores son suficientes para concluir que $\preceq$ es una relación de orden entre subconjuntos de rasgos.

La definición recursiva de la relación permitió probar fácilmente que es una relación de orden. Es posible, y será útil más adelante, dar una caracterización no recursiva de dicha relación, lo cual se hace a continuación.

3.4.5.Proposición. *$X \preceq Y$ si y sólo si $X = Y$ o bien si existe $x_0 \in X$ tal que*
i) $x_0 \notin Y$ y
ii) para todo $y \in Y$ con $y < x_0$ se tiene que $y \in X$.

Dem: Supóngase que $X \preceq Y$ y $X \neq Y$, lo cual se denotará de de aquí en adelante como $X \prec Y$.

a) Si $Y = \emptyset$ entonces $X \neq \emptyset$ y no hay más que probar.

b) Si $X = [x \mid X']$, $Y = [y \mid Y']$ y $x < y$ entonces claramente $x \notin Y$ e (ii) se cumple trivialmente.

c) Si $x = y$ entonces $X' \prec Y'$ y el resultado se obtiene por inducción.

Supóngase que existe $x_0 \in X$ tal que se cumplen las condiciones (i) e (ii).

a) Si $Y = \emptyset$ no hay más que probar.

b) Si $X = [x_0 \mid X']$, $Y = [y \mid Y']$ y $x_0 \neq y$ entonces claramente $x_0 < y$ y por lo tanto $X \prec Y$.

c) Si $X = [x \mid X']$, $Y = [y \mid Y']$ y $x = y$ entonces $x_0 \in X'$ y las condiciones (i) e (ii) se cumplen considerando a X' e Y', de donde se sigue que $X' \prec Y'$ y por lo tanto $X \prec Y$. q.e.d

3.5. Una medida de error.

En la sección anterior se definió una relación de orden entre los subconjuntos de rasgos, con lo que se tiene una dirección de búsqueda en el conjunto de todos esos subconjuntos. A continuación se define un criterio para decidir cuándo un conjunto de rasgos tiene posibilidades de ser un testor mínimo y cuándo no las tiene.

Como ya se ha dicho antes —en la sección (2.2) del capítulo anterior— un testor para una muestra de aprendizaje es un subconjunto de rasgos que distingue perfectamente las muestras de cada clase que la componen, es decir, que no comete errores al clasificar las descripciones de objetos de la muestra de aprendizaje en su muestra de clase correspondiente. Por otra parte, el hecho de que un subconjunto

de rasgos no sea un testor no significa que sea incapaz de distinguir entre una descripción y otra, sino que confunde algunas descripciones pertenecientes a muestras de clase distintas.

La técnica desarrollada para distinguir testores mínimos se basa en *tomar la cantidad de descripciones que confunde un subconjunto de rasgos como una medida de su "error"*, siendo un testor aquel subconjunto de rasgos cuyo error es igual a cero y un testor mínimo aquel que no solamente tiene error nulo, sino que lo alcanza con un número mínimo de rasgos.

3.5.1.Definición. *Sean* $\Omega' = \bigcup_{i=1}^{r} \Omega'_i$ *una muestra de aprendizaje y* $X \subset R$ *un subconjunto de rasgos. Denótese por* $E(X)$ *el conjunto de todas las parejas de descripciones* $\{\omega, \omega'\}$ *tales que* ω *y* ω' *son semejantes con respecto a* X *y pertenecen a muestras de clase distintas, es decir*

$$E(X) = \left\{ \{\omega, \omega'\} \mid \omega \overset{X}{\sim} \omega', \text{ con } \omega \in \Omega_i,\ \omega' \in \Omega_k \text{ e } i \neq k \right\}.$$

El error *de* X *se define como la cardinalidad de* $E(X)$.

3.5.2.Corolario. *X es testor si y sólo si el error de X es igual a cero.*

Dem: Se sigue inmediatamente de la definición anterior y de la definición de testor. **q.e.d**

Es fácil comprobar que si X e Y son subconjuntos de rasgos con $X \subset Y$ entonces no sólo el error de X es mayor o igual al error de Y sino que además $E(Y) \subset E(X)$. Por otra parte, si $X = \{x_1, \ldots, x_n\}$ y se denota por X_j al conjunto de los rasgos en X menores o iguales al rasgo x_j, con j entre 1 y n, entonces a cada rasgo $x_j \in X$ podemos asociarle un valor

$$x_j \mapsto \text{error de } X_j$$

que indica el *error cometido por X hasta el rasgo x_j*, definiéndose de esta manera una función:

$$e_X : X \to \mathbf{N}$$
$$x_i \mapsto \text{error de } X_j.$$

Se sigue inmediatamente de la definición de e_X que se trata de una función decreciente y que el valor $e_X(x_j)$ depende únicamente de los rasgos en X_j y no de los rasgos en X mayores que x_j. Formalmente:

3.5.3.Proposición.

a) $E(X_k) \subset E(X_j)$ *si* $j \leq k$.

b) $e_X(x_k) \leq e_X(x_j)$ *si* $j \leq k$.

c) $e_{X_j}(x) = e_X(x)$ *para todo* $x \in X_j$, $j = 1, \ldots, n$.

Supóngase ahora que $e_X(x_j) = e_X(x_{j+1})$. Significa que se comete el mismo error con los rasgos en X_j que con los rasgos en X_{j+1}, o dicho de otro modo, que el rasgo x_{j+1} *no aporta ninguna información adicional* sobre la muestra de aprendizaje a la contenida en X_j y podemos concluir que el rasgo x_{j+1} está de más en el conjunto X. Dicho lo anterior, no resulta extraña la proposición siguiente:

3.5.4.Proposición. *Si* X *es un testor mínimo entonces* e_X *es inyectiva.*

Dem: Sea X un testor y supóngase $e_X(x_j) = e_X(x_{j+1})$. Esto significa que $E(X_j) = E(X_{j+1})$ y por consiguiente $E(X - \{x_{j+1}\}) = E(X_j \cup \{x \in X \mid x_{j+1} < x\}) = E(X_{j+1} \cup \{x \in X \mid x_{j+1} < x\}) = E(X) = 0$. Por lo tanto, $X' = X - \{x_{j+1}\}$ es un testor y X no es un testor mínimo. q.e.d

Se tiene así una condición *necesaria* para que un subconjunto de rasgos sea un testor mínimo. Nótese que verificar esta condición no sólo permite decidir cuándo un subconjunto de rasgos no es un testor, sino también indica cuándo un testor no es mínimo. Claramente, $X = \{x_1, \ldots, x_n\}$ es un testor si y sólo si $e_X(x_n) = 0$.

No es difícil encontrar ejemplos donde se observa que la inyectividad de la función e_X no es una condición *suficiente* para que un testor sea mínimo; en este sentido no es un criterio efectivo para distinguir testores mínimos. Sin embargo, no requiere más cálculos de los necesarios para decidir si un conjunto es un testor —véase la sección (3.7)— y, unido a una manera adecuada de recorrer el conjunto de todos los subconjuntos de rasgos, es posible asegurar que se encuentra a todos los testores mínimos.

3.6. La función de recorrido.

Habiendo definido ya una relación de orden en $P(\mathcal{R})$, el conjunto de todos los subconjuntos de rasgos, y elegido un criterio para decidir cuándo un subconjunto de rasgos *puede ser* un testor mínimo, lo que resta es definir la manera en que se va a recorrer $P(\mathcal{R})$ buscando los testores mínimos. Como se explicó en la sección (3.3), se va a definir una función

$$\varphi : P(\mathcal{R}) \to P(\mathcal{R})$$

tal que si

$$\Phi = \{ X \subset \mathcal{R} \mid X = \varphi^n(\mathcal{R}),\ n \in \mathbb{N} \}$$

y

$$TM = \{ X \subset \mathcal{R} \mid X \text{ es un testor mínimo} \},$$

entonces

$$TM \subset \Phi.$$

El valor de $\varphi(X)$ depende de dos cosas: de la inyectividad o no inyectividad de e_X y de si X es un testor o no. La definición de φ se hace a partir de tres funciones auxiliares que se detallan posteriormente.

1.6.1.Definición. *La función de recorrido φ es*

$$\varphi : P(\mathcal{R}) \to P(\mathcal{R})$$

tal que

a) si X es un testor y e_X no es inyectiva entonces

$$\varphi(X) = \sigma(X);$$

b) si X es un testor y e_X es inyectiva entonces

$$\varphi(X) = \beta(X);$$

c) si X no es testor entonces

$$\varphi(X) = \alpha(X).$$

Lo que hace la función σ es eliminar de X aquellos rasgos para los que se repiten los valores de e_X, dejando el elemento más pequeño de X asociado a cada uno de los valores de e_X. La justificación de la utilidad de σ se dió en la sección anterior y es que un testor X no puede ser mínimo si e_X no es inyectiva en él.

1.6.2.Definición. *La función de reducción σ es*

$$\sigma : P(\mathcal{R}) \to P(\mathcal{R})$$

tal que si $X = \{x_1, \ldots, x_n\}$ entonces

$$\sigma(X) = \{ x_i \in X \mid \text{para todo } x \in X \text{ con } x < x_i \text{ se tiene que } e_X(x) > e_X(x_i) \}$$

y

$$\sigma(\emptyset) = \emptyset.$$

3.6.3.Proposición. *Si X un subconjunto de rasgos entonces*
i) $E(\sigma(X)) = E(X)$ *y*
ii) $e_{\sigma(X)}(x) = e_X(x)$ *para cualquier* $x \in \sigma(X)$.

Dem: Por inducción sobre la cardinalidad de X. Si X contiene solamente un rasgo entonces $\sigma(X) = X$ y no hay más que probar. Supóngase que la proposición es válida para $X = \{x_1, \ldots, x_k\}$ y sea ahora $X = \{x_1, \ldots, x_{k+1}\}$:

a) si $e_X(x_k) = e_X(x_{k+1})$ entonces $x_{k+1} \notin \sigma(X)$, teniéndose $\sigma(X) = \sigma(X_k)$ y por inducción $E(\sigma(X)) = E(\sigma(X_k)) = E(X_k) = E(X)$ y $e_{\sigma(X)} = e_{\sigma(X_k)} = e_{X_k} = e_X$;

b) si $e_X(x_k) > e_X(x_{k+1})$ entonces por inducción $E(\sigma(X)) = E(\sigma(X_k) \cup \{x_{k+1}\}) = E(X_k \cup \{x_{k+1}\}) = E(X)$. q.e.d

3.6.4.Corolario. $e_{\sigma(X)}$ *es inyectiva.*

3.6.5.Corolario. *Si $X \subset \mathcal{R}$ es un testor, entonces $\sigma(X)$ también es un testor.*

La función σ permite encontrar, dado un testor que no puede ser mínimo, otro testor con buenas posibilidades de serlo. Como la función de error no da más información sobre como *reducir* el nuevo testor, es necesario *sustituir* algunos rasgos en él por otros. De hecho, si e_X es inyectiva en X, lo que se hace es sustituir el último rasgo contenido en X por los rasgos en $\mathcal{R} - X$ mayores que él.

3.6.6.Definición. *La función de sustitución β es*

$$\beta : P(\mathcal{R}) \to P(\mathcal{R})$$

tal que si $X = \{x_1, \ldots, x_n\}$ entonces

$$\beta(X) = X_{n-1} \cup \{ r \in \mathcal{R} \mid r > x_n \}$$

y

$$\beta(\emptyset) = \emptyset.$$

Esta es la función propia del algoritmo *back-track*, eliminando rasgos en orden decreciente.

Supóngase ahora que, recorriendo $P(\mathcal{R})$, se ha llegado a un subconjunto de rasgos X que no es un testor; lo que se tiene que hacer es sustituir algún rasgo en

X por otros. Si el último rasgo en X no es el último rasgo en $\mathcal{R}$ entonces puede aplicarse a X la función β ya definida; pero si el último rasgo en X es el último en $\mathcal{R}$ entonces aplicar la función β no sirve de nada, pues se tiene $\beta(X) \subset X$ y X *no* es un testor. En este caso, es necesario "regresar" en X hasta encontrar un rasgo que pueda ser sustituido por otros con la esperanza de encontrar un testor mínimo; para lo cual se define la siguiente función:

3.6.7.Definición. *La función de regreso y sustitución* α *es*

$$\alpha : P(\mathcal{R}) \to P(\mathcal{R})$$

tal que si $X = \{x_1, \ldots, x_n\}$ *entonces*

a) si $r < x_1$ *para toda* $r \in \mathcal{R} - X$ *entonces*

$$\alpha(X) = \emptyset$$

b) si $x_j = \max\{\, x \in X \mid \text{existe } r \in \mathcal{R} - X \text{ con } x < r \,\}$ *entonces*

$$\alpha(X) = X_{j-1} \cup \{\, r \in \mathcal{R} \mid r > x_j \,\}$$

y

$$\alpha(\emptyset) = \emptyset.$$

El primer resultado importante que contempla a las tres funciones σ, β y α ya definidas habla de su monotonicidad.

3.6.8.Proposición. $X \preceq \sigma(X), \beta(X), \alpha(X)$ *para todo* $X \in P(\mathcal{R})$.

Dem: El caso $X = \emptyset$ es obvio. Supóngase entonces que $X \neq \emptyset$:

a) $X \preceq \sigma(X)$ pues $\sigma(X) \subset X$.

b) Si $X = \{x_1, \ldots, x_n\}$ entonces $\beta(X) = X_{n-1} \cup \{\, r \in \mathcal{R} \mid r > x_n \,\}$ y la relación $X \preceq \beta(X)$ se sigue de la proposición (3.4.5) tomando $x_0 = x_n$ y $Y = \beta(X)$.

c) La prueba de que $X \preceq \alpha(X)$ se hará por inducción. Si $X = \{x_1\}$ y $\alpha(X)$ está dada por el primer inciso de la definición (3.6.6) entonces $\alpha(X) = \emptyset$ y la desigualdad se obtiene de la primera condición de la relación de orden. En otro caso, si $\alpha(X)$ se obtiene por el segundo inciso de la definición, claramente $x_i = x_1$ y la desigualdad se obtiene de la segunda condición de la relación de orden. Supóngase ahora que la desigualdad $X \preceq \alpha(X)$ se cumple para todo X con n elementos y sea ahora $X = \{x_1, \ldots, x_{n+1}\}$; si $\alpha(X) = \emptyset$ no hay más que hacer, y tampoco si $x_i = x_1$; finalmente, si $x_i \neq x_1$ entonces puede comprobarse

fácilmente que $\alpha(X) = \{x_1\} \cup \alpha(X')$ con $X' = \{x_2, \ldots, x_{n+1}\}$ y en este caso la desigualdad $X \preceq \alpha(X)$ se sigue de la tercera condición de la relación de orden y de la hipótesis de inducción. q.e.d

3.6.9.Corolario. *$X \prec \varphi(X)$ para cualquier $X \in P(\mathcal{R})$ con $X \neq \emptyset$.*

Dem: Se sigue directamente de la definición de φ y de la proposición anterior. **q.e.d**

La siguiente proposición está encaminada hacia demostrar que φ efectivamente encuentra a todos los testores mínimos de una muestra de aprendizaje dada.

3.6.10.Proposición. *Si X es un subconjunto propio de $\mathcal{R}$ tal que $X = \varphi^n(\mathcal{R})$ para alguna $n \in \mathbb{N}$ y r_0 es el máximo de $\mathcal{R} - X$, entonces la función e_X definida sobre X es inyectiva en el subconjunto $Z = \{ x \in X \mid x < r_0 \}$.*

Dem: La demostración se hará por inducción. Si $n = 1$ entonces, como $\mathcal{R}$ es un testor:

a) si la función de error no es inyectiva en $\mathcal{R}$ entonces $X = \sigma(\mathcal{R})$ y e_X es inyectiva sobre todo X, por la proposición (3.6.4);

b) si la función de error es inyectiva en $\mathcal{R}$ entonces $X = \beta(\mathcal{R}) = \mathcal{R}_{N-1}$ y e_X es inyectiva sobre todo X, por la proposición (3.5.3).

Supóngase que la proposición es cierta hasta n y sea $X = \varphi^{n+1}(\mathcal{R}) = \varphi(Y)$ con $Y = \{y_1, \ldots, y_s\} = \varphi^n(\mathcal{R})$:

a) si Y es un testor y e_Y no es inyectiva entonces $X = \sigma(Y)$ y e_X es inyectiva sobre todo X, por la proposición (3.6.4);

b) si Y es un testor y e_Y es inyectiva entonces

$$X = \beta(Y) = Y_{s-1} \cup \{ r \in \mathcal{R} \mid r > y_s \};$$

es claro que $r_0 = y_s$, $Z = Y_{s-1}$ y e_X es inyectiva sobre Y_{s-1} por la proposición (3.5.3);

c) si Y no es un testor, como X es un subconjunto propio de $\mathcal{R}$ entonces

$$X = \alpha(Y) = Y_{j-1} \cup \{ r \in \mathcal{R} \mid r > y_j \}$$

con

$$y_j = \max\{ y \in Y \mid \text{existe } r \in \mathcal{R} - Y \text{ con } r > y \};$$

por inducción y la definición de y_j se tiene que e_Y es inyectiva en Y_{j-1}; finalmente, por la proposición (3.5.3) se tiene que e_X es inyectiva en $Z = Y_{j-1}$ con $r_0 = y_j$. **q.e.d**

La proposición anterior indica que la función de error de cada miembro de Φ es inyectiva en un segmento inicial de cada uno de ellos. Entre mayor sea ese segmento para cada candidato a testor mínimo se tienen mejores posibilidades de que sea un testor mínimo, siempre y cuando su error total sea igual a cero.

3.6.11.Proposición. *No existe ningún testor mínimo $T \subset \mathcal{R}$ tal que*

$$\varphi^n(\mathcal{R}) \prec T \prec \varphi^{n+1}(\mathcal{R})$$

para alguna $n \in \mathbb{N}$.

Dem: Supóngase que la proposición es falsa y denótese por X a $\varphi^n(\mathcal{R})$:

a) Si e_X no es inyectiva entonces $\varphi(X) = \sigma(X)$ y sea $t_0 = \min\{\, t \in T \mid t \notin \sigma(X) \,\}$, el cual existe pues T es menor que $\sigma(X)$. Por la definición de t_0 se tiene que para cualquier $t \in T$ con $t < t_0$ se cumple que $t \in \sigma(X)$; además de la proposición (3.4.5) se tiene que para cualquier $x \in \sigma(X)$ con $x < t_0$ se cumple que $x \in T$. Por lo tanto, T y $\sigma(X)$ coinciden en todos sus elementos menores que t_0.

   Por otra parte, como $\sigma(X) \subset X$, todos los elementos de T menores que t_0 pertenecen a X. Si $t_0 \in X$ entonces fue eliminado de X por σ, y debe existir $x_0 \in \sigma(X)$ con $x_0 < t_0$ —y, por lo tanto, $x_0 \in T$— y $e_X(x_0) = e_X(t_0)$, lo cual significa que e_T no es inyectiva, contradiciendo la proposición (3.5.4). Por consiguiente, $t_0 \notin X$, y en este caso se tiene que e_X es inyectiva en el subconjunto $Z = \{\, x \in X \mid x < t_0 \,\}$, por la proposición anterior, y entonces se tiene que los tres conjuntos X, T, y $\sigma(X)$ coinciden en sus elementos menores a t_0; por lo tanto, $T \prec X$, pues $t_0 \notin X$, lo cual contradice la hipótesis inicial.

b) Si e_X es inyectiva y $X = \{x_1, \ldots, x_n\}$ es un testor, entonces $\varphi(X) = \beta(X)$ que coincide con X en los $n-1$ primeros rasgos —definición (3.6.6). Por consiguiente, $T_{n-1} = X_{n-1}$ también, pues en otro caso T no podría ser mayor que X y menor que $\varphi(X)$ al mismo tiempo. Si $x_n \notin T$ entonces $T \subset \varphi(X)$, contradiciendo la hipótesis de que $T \prec \varphi(X)$. Por lo tanto, debe cumplirse que $x_n \in T$; pero entonces $X \subset T$ y T no puede ser un testor mínimo[3].

c) Si X no es un testor entonces $\varphi(X) = \alpha(X)$ y se tienen dos casos, según $\varphi(X)$ es o no vacío. Si $\varphi(X) = \emptyset$ entonces, por la definición de α, X está compuesto por los últimos elementos de $\mathcal{R}$ y en este caso es fácil ver que $T \subset X$, por lo que T no puede ser un testor. En otro caso, sea x_j como en la definición (3.6.7);

---

[3] Puede probarse que no existe ningún subconjunto de rasgos Y tal que $X \prec Y \prec \beta(X)$, dado $X \subset \mathcal{R}$. Esto significa que el orden definido sobre $P(\mathcal{R})$ puede considerarse inducido por la función β del algoritmo *back-track*.

si $x_j \notin T$ entonces $T \subset \varphi(X)$, lo cual no es posible; debe tenerse entonces que $x_j \in T$ y en ese caso

$$X = X_j \cup \{x_{j+1}, \ldots, x_n\}$$
$$T = X_j \cup \{t_{j+1}, \ldots, t_m\}$$

donde $\{x_{j+1}, \ldots, x_n\}$ son los últimos elementos de $\mathcal{R}$ y la relación $X \prec T$ implica $\{t_{j+1}, \ldots, t_m\} \subset \{x_{j+1}, \ldots, x_n\}$, obteniendose $T \subset X$ y, por consiguiente, T no puede ser un testor.

Por lo tanto, dado que se han analizado los tres casos posibles, la hipótesis de que existe un testor mínimo entre X y $\varphi(X)$ es falsa, y la proposición enunciada es cierta. **q.e.d**

3.6.12.Corolario. $TM \subset \Phi$

Dem: Se sigue inmediatamente de la proposición anterior aplicando inducción. **q.e.d**

3.7. Comparaciones

En las secciones anteriores se presentó el algoritmo desarrollado para encontrar todos los testores mínimos de una muestra de aprendizaje. Se puede observar que dicho algoritmo cumple con tres de las cuatro condiciones planteadas al principio de su diseño, a saber:

a) trabaja únicamente con la muestra de aprendizaje —y con la lista de los testores más pequeños encontrados;

b) toma en cuenta la estructura interna de la muestra de aprendizaje —en el momento de construir la función e_X para cada subconjunto de rasgos;

c) efectua, generalmente, modificaciones en más de un rasgo para construir un candidato a testor mínimos a partir del anterior.

Falta verificar la última condición de diseño, muy importante: que el algoritmo sea eficiente en la práctica y pueda ser utilizado en microcomputadoras para resolver problemas de buen tamaño —véase la sección (3.3).

El algoritmo aquí desarrollado consiste, esencialmente, de una función φ que recorre el conjunto de todos los subconjuntos de rasgos escogiendo los testores mínimos, la cual se definió utilizando tres funciones básicas a las que se ha denominado β,

α y σ, siendo β la función propia del algoritmo *back-track*, α una función que "salta" sobre subconjuntos de rasgos que ciertamente no son testores, y σ, que permite simplificar testores eliminando algunos rasgos superfluos. De las tres funciones, únicamente σ toma en cuenta la estructura interna de la muestra de aprendizaje y constituye la aportación más representativa de este trabajo.

La necesidad de introducir la función σ —y la función de error e_X, para cada subconjunto de rasgos X— debe justificarse mostrando que el algoritmo resultante es más eficiente, especialmente en términos del tiempo de procesamiento, que un algoritmo basado solamente en las funciones α y β. Para ésto, considérese una nueva función de recorrido definida como sigue:

3.7.1.Definición.

$$\psi : P(\mathcal{R}) \rightarrow P(\mathcal{R})$$

tal que

a) si X es un testor entonces

$$\psi(X) = \beta(X)$$

b) en otro caso

$$\psi(X) = \alpha(X).$$

Si X es un subconjunto de rasgos cualquiera, para calcular $\psi(X)$ es necesario, antes que nada, verificar si ese subconjunto de rasgos es o no un testor. Suponiendo que no se dispone de más información sobre X además de los rasgos que lo conforman y los valores de dichos rasgos en cada una de las descripciones en la muestra de aprendizaje, la verificación de si X es un testor o no es de complejidad $O(m^2n)$ donde m es el número total de descripciones en la muestra de aprendizaje y n es la cardinalidad de X. Esto resulta de que es necesario comparar unas contra otras a las descripciones en la muestra hasta encontrar dos descripciones semejantes con respecto a X en clases distintas, o bien hasta agotarlas a todas, y cada comparación entre dos descripciones puede llevarse hasta comparar los valores de ambas en cada uno de los rasgos que contiene X.

Por otra parte, el número de operaciones requerido para calcular $\varphi(X)$ depende directamente del necesario para calcular la función de error e_X. Ahora bien, bajo las mismas suposiciones que en el análisis anterior, la complejidad de esos cálculos es también $O(m^2n)$ ya que pueden realizarse comparando las descripciones entre sí y anotando si son o no semejantes con respecto a X.

Por lo tanto, la dos funciones ψ y φ son igualmente fáciles y rápidas —o difíciles y lentas— de calcular y, por consiguiente, la introducción de las funciones e_X y σ no aumenta el orden de la complejidad de φ sobre el de ψ.

Es fácil comprobar que el algoritmo basado en la función ψ encuentra todos los testores mínimos de una muestra de aprendizaje dada —de hecho, tal resultado se obtiene inmediatamente de la demostración de la proposición (3.6.11) eliminando el primer inciso que corresponde a la aplicación de σ. Se denotará por $\mathcal{B}_\varphi$ al algoritmo que utiliza la función φ y por $\mathcal{B}_\psi$ al que usa la función ψ para recorrer a $P(\mathcal{R})$.

Dado que los dos algoritmos son idénticos salvo por la función de recorrido que utilizan, para compararlos deben tenerse en cuenta dos aspectos esenciales:

i) el número de operaciones requerido para calcular φ y ψ, y

ii) la cardinalidad de los conjuntos

$$\Phi = \{ X \in P(\mathcal{R}) \mid X = \varphi^n(\mathcal{R}) \text{ con } n \in \mathbb{N} \}$$

y

$$\Psi = \{ X \in P(\mathcal{R}) \mid X = \psi^n(\mathcal{R}) \text{ con } n \in \mathbb{N} \}.$$

Del análisis realizado de la complejidad de cálculo de las funciones ψ y φ se desprende que $\mathcal{B}_\varphi$ será más eficiente que $\mathcal{B}_\psi$ solamente si el número de veces que tiene que aplicarse la función de recorrido correspondiente es menor; más aún, dado que el algoritmo $\mathcal{B}_\varphi$ es más complicado, la diferencia entre los dos algoritmos debe ser lo suficientemente amplia para justificar el trabajo.

Considérese la siguiente definición de la *precisión* de un algoritmo:

3.7.2.Definición. *Sean TM el conjunto de los testores mínimos de una muestra de aprendizaje dada, $\mathcal{B}$ un algoritmo para el cálculo de los testores mínimos y*

$$C(\mathcal{B}) = \{ X \subset \mathcal{R} \mid \mathcal{B} \textit{ elige a } X \textit{ como candidato a testor mínimo} \}$$

con $TM \subset C(\mathcal{B})$. La precisión del algoritmo $\mathcal{B}$ sobre la muestra de aprendizaje se define como el número de testores mínimos de la misma entre el número de elementos de $C(\mathcal{B})$

$$\text{prec}(A) = \frac{\textit{cardinalidad de } TM}{\textit{cardinalidad de } C(\mathcal{B})}.$$

Dicho de otro manera, la precisión de un algoritmo no es más que la proporción de testores mínimos entre el total de candidatos a testores mínimos generados por él.

Se realizaron pruebas para medir la precisión de los algoritmos $\mathcal{B}_\psi$ y $\mathcal{B}_\varphi$ y determinar la importancia de modificar a la función ψ añadiendo la función σ. Por

cuestiones de tiempo, las muestras de aprendizaje se redujeron a muestras generadas aleatoriamente, de acuerdo al siguiente esquema:

i) se determinó el número de rasgos, de clases y de descripciones por clase (todas las clases tuvieron el mismo número de descripciones de objetos);

ii) para cada clase se generó aleatoriamente el número de rasgos relevantes y los valores representativos de dichos rasgos;

iii) para cada clase se generaron aleatoriamente las descripciones necesarias, propiciando la aparición de los valores esperados en los rasgos relevantes de la clase.

El número de rasgos en las muestras de aprendizaje varió entre diez y quince, mientras que el número de clases estuvo entre tres y seis; el número total de patrones fue siempre de sesenta. Los resultados obtenidos se presentan en la tabla 1.

Tabla 1

Algoritmo	Precisión mínima	Precisión máxima	Precisión promedio
B_ψ	0.0329	0.1592	0.0987
B_φ	0.2140	0.2822	0.2703

Como puede observarse, la modificación de la función ψ con la introducción de la función de error y la función σ resulta en un algoritmo en promedio aproximadamente tres veces más preciso sobre las muestras generadas, aunque en algunas ocasiones llego a ser hasta casi diez veces más rápido. Por supuesto, habrá que comparar el desenvolvimiento de los dos algoritmos sobre muestras de aprendizaje más grandes y obtenidas a partir de datos reales, a fin hacer una evaluación más exacta de su eficiencia relativa.

CAPITULO 4

El Sistema

Como se mencionó en la introducción, uno de los objetivos de este trabajo fue construir un pequeño sistema para reconocimiento de patrones y clasificación, que pudiera ser utilizado en una microcomputadora con tiempos de ejecución razonables. En este capítulo se describe el sistema desarrollado, sus posibilidades y sus limitaciones.

4.1. Descripción general.

El Sistema de Reconocimiento de Patrones y de Clasificación (RECLA) está compuesto actualmente por dos programas:

- Programa para Reconocimiento de Patrones (REC).
- Programa para Clasificación de Objetos (CLA).

La comunicación entre los dos se realiza a través de archivos intermedios. De hecho, prácticamente toda la entrada y la salida de estos programas se realiza mediante archivos, sumando un total de cinco:

1) archivo con la muestra de aprendizaje (MA),
2) archivo con los testores mínimos (TM),
3) archivo con los pesos diferenciantes de los rasgos y los ideales de cada una de las clases (PD),
4) archivo con los objetos por clasificar (OB), y
5) archivo con la clasificación de los objetos (CL),

más un archivo temporal utilizado por REC cuando el número de subconjuntos de rasgos que se deben revisar para encontrar los testores mínimos es muy grande.

El diagrama general del sistema se presenta en la figura 1. Se decidió utilizar archivos por la gran cantidad de datos que en un momento dado pueden estar involucrados en la ejecución de los programas —particularmente la muestra de aprendizaje y los objetos por clasificar—, lo cual hace engorroso un funcionamiento interactivo. Ciertamente,la comunicación mediante archivos no es atractiva cuando se maneja poca información y una de las mejoras pendientes a los programas consiste en permitir tanto la comunicación mediante archivos como una relación más estrecha con

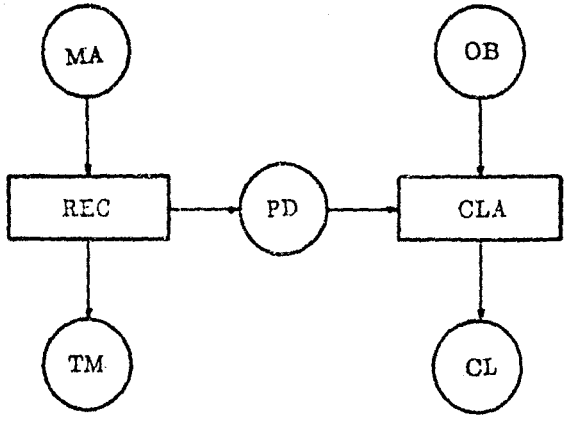

figura 1

el usuario. Por otra parte, se utilizan archivos de texto, siendo la razón principal de su uso las facilidades que proporcionan para la edición de datos.

1.2. Programa para Reconocimiento de Patrones (REC).

El objetivo de este programa es, dada una muestra de aprendizaje, calcular:

- los testores mínimos,
- los pesos diferenciantes de los rasgos utilizados para describir a los objetos, y
- los ideales (descripciones de objetos más representativos) de cada una de las clases.

Desde el punto de vista operativo, puede considerarse a REC compuesto de tres partes, cada una con una función específica:

- Comunicación con el usuario.
- Manejo de archivos.
- Reconocimiento de patrones.

Comunicación con el usuario. Es muy simple y consiste esencialmente de las preguntas y respuestas indispensables para la inicialización del sistema: inicio de sesión o continuación de una anterior y nombre del archivos con la muestra de aprendizaje.

Manejo de archivos. Se realizan las operaciones necesarias para leer los datos contenidos en los archivos de entrada y escribir los resultados en los archivos de salida, según el formato preestablecido:

- Archivo con la muestra de aprendizaje:

  ```
  <comentarios opcionales>
  <número de objetos>
  <número de rasgos>
  <valor máximo del primer rasgos>
  ⋮
  <valor máximo del último rasgo>
  <descripción del primer objeto>
  <clase del primer objeto>
  ⋮
  <descripción del último objeto>
  <clase del último objeto>.
  ```

- Archivo con los testores mínimos:

  ```
  <comentarios opcionales>
  <número de testores mínimos>
  <número de rasgos>
  <primer testor mínimo> <marca de fin de testor>
  ⋮
  <último testor mínimo> <marca de fin de testor>
  ```

- Archivo con los pesos diferenciantes y los ideales de las clases:

  ```
  <comentarios opcionales>
  <número de rasgos>
  <valor máximo del primer rasgo>
  <peso diferenciante del primer rasgo>
  ⋮
  <valor máximo del último rasgo>
  <peso diferenciante del último rasgo>
  ```

```
<número de clases>
<número de ideales de la primera clase>
<peso informativo de los ideales de la primera clase>
<ideales de la primera clase>
⋮
<número de ideales de la última clase>
<peso informativo de los ideales de la última clase>
<ideales de la última clase>
```

Reconocimiento de patrones. Constituye el propósito del programa y su parte central es la ejecución del algoritmo para el cálculo de los testores mínimos expuesto en el capítulo anterior, implantado de la siguiente manera:

$X \leftarrow \mathcal{R}$;
mientras $X \neq \emptyset$
  { Calcula la función c_X };
  si error de $X \neq 0$ **entonces**
  $X \leftarrow \alpha(X)$
  sino
    $X \leftarrow \sigma(X)$;
    { Guarda el testor X en la lista de testores,
      eliminando de ella todos los testores que contiene a X. };
    $X \leftarrow \beta(X)$

donde $\mathcal{R}$ es el conjunto de todos los rasgos en términos de los cuales se describen los objetos.

Dado que el tiempo de ejecución del programa puede ser relativamente largo, se optó por guardar regularmente en disco el estado de las operaciones, en un archivo temporal; la copia a disco se realiza cada vez que se han revisado un número fijo de candidatos a testores mínimos, contando a partir de la última copia a disco o, en su caso, a partir del inicio del programa. Por consiguiente, al principio de cada ejecución, el programa pregunta al usuario si se trata de la continuación de alguna sesión anterior y, en caso de que la respuesta sea afirmativa, busca el archivo temporal y recupera el estado de la ejecución anterior en el momento en que se realizó la última copia a disco, prosiguiendo las operaciones a partir de ese punto.

4.3. Programa para Clasificación de Objetos (CLA).

El objetivo de este programa es determinar la pertenencia o no pertenencia de cada uno de los objetos de una colección dada a cada una de las clases que conforman el universo de objetos; es decir, obtener la clasificación de los objetos dados. Para llevar a cabo su cometido, CLA necesita conocer los ideales de cada una de las clases y los pesos diferenciantes de los rasgos en términos de los cuales se describen tanto los objetos por clasificar como los ideales de las clases.

Como sucede con REC, CLA puede considerarse compuesto de tres partes, cada una de ellas con una función propia:

- Comunicación con el usuario.
- Manejo de archivos.
- Clasificación de objetos.

Comunicación con el usuario. Es muy simple y se reduce esencialmente a preguntar y obtener el nombre del archivo que contiene las descripciones de los objetos por clasificar.

Manejo de archivos. Se realizan las operaciones necesarias para leer los datos contenidos en los archivos de entrada y escribir los resultados en los archivos de salida, según el formato preestablecido:

- Archivo con los pesos diferenciantes y los ideales de las clases:

  ```
  Mismo formato que el archivo producido por REC
  ```

- Archivo con los objetos por clasificar:

  ```
  <comentarios opcionales>
  <número de objetos> <número de rasgos>
  <descripción del primer objeto>
  ⋮
  <descripción del último objeto>
  ```

- Archivo con la clasificación de los objetos:

  ```
  <comentarios opcionales>
  <número de objetos> <número de rasgos>
  <clasificación del primer objeto>
  ⋮
  <clasificación del último objeto>
  ```

Clasificación de objetos. Se realiza mediante el siguiente algoritmo:

para { cada uno de los objetos por clasificar }
{ Obtiene la cercanía del objeto a cada una de las
clases que componen el universo (véase la sección (2.4)). };
{ Determina la cercanía máxima. };
{ Normaliza los valores de cercanía. };
{ Depósita los valores obtenidos en el archivo que
contendrá la clasificación de los objetos. }

1.4. Detalles de programación.

El sistema RECLA, como ya se dijo, está compuesto actualmente por dos programas, REC y CLA. Ambos programas estan escritos en Pascal y fueron desarrollados en microcomputadoras IBM-PC y compatibles utilizando el sistema TURBO PASCAL versión 3.0. Mientras la elección de la computadora fue determinada por las características del equipo disponible —*hardware* y *software*—, la decisión de utilizar el lenguaje de programación Pascal tiene muchos motivos:

- Es un lenguaje de alto nivel con muchas facilidades para escribir programas claros, bien estructurados y sencillos de depurar.
- Es de uso extendido, especialmente en el ámbito académico.
- Es el lenguaje utilizado en los cursos básicos y en muchos otros más avanzados en la Facultad.
- Existen compiladores que permiten la traducción de Pascal a código de máquina muy eficiente.

Por comodidad, se utilizaron algunas de las extensiones que hace el sistema TURBO PASCAL al lenguaje Pascal estándar, como son las declaraciones de constantes estructuradas, algunas instrucciones para el manejo de archivos de texto y para el control del flujo del programa. El texto de los programas y un ejemplo de su ejecución se encuentran disponibles en el apéndice.

Conclusiones

Como se señaló en la introducción, los objetivos fundamentales al iniciar este trabajo fueron dos:

- Diseñar un buen algoritmo para calcular los testores mínimos de una muestra de aprendizaje cualquiera.
- Construir un pequeño sistema para reconocimiento de patrones y clasificación.

Ahora bien, en el capítulo tercero de la presente tesis se desarrolló un algoritmo para el cálculo de los testores mínimos, probando además que es correcto y señalando sus posibilidades de ser eficiente, dadas las características de los candidatos a testores mínimos que propone. Dado que el algoritmo es sencillo de formular, resulta una alternativa digna de tomar en cuenta en el momento de diseñar sistemas para reconocimiento de patrones basados en la Teoría de Test. Por consiguiente, considero alcanzado completamente el primer objetivo, salvo en un aspecto importante: se estimó la complejidad del algoritmo para proporcionar *un* candidato a testor mínimo, mas no se estimó la complejidad para calcular *todos* los testores mínimos de una muestra de aprendizaje dada. Este último análisis no es sencillo, debido a que el número de testores mínimos depende fuertemente de la estructura interna de la muestra de aprendizaje.

En lo que respecta al segundo objetivo, se diseño un sistema para reconocimiento de patrones y clasificación que utiliza el algoritmo desarrollado, el cual está disponible y funcionando en microcomputadoras. Por esta razón, considero que se cumplió también el segundo objetivo.

Por supuesto, quedan aún muchas cosas por hacer, señalándose las siguientes:

- Estimar la complejidad del algoritmo desarrollado para encontrar todos los testores mínimos de una muestra de aprendizaje cualquiera.
- Obtener condiciones suficientes para que un conjunto de rasgos sea un testor mínimo, mismas que sean fáciles de calcular y de incorporar al algoritmo.
- Añadir la posibilidad de comportamiento interactivo al Sistema para Reconocimiento de Patrones y Clasificación (RECLA).
- Ofrecer la opción, dentro del sistema, de elegir entre varios criterios de clasificación.
- Evaluar el comportamiento del sistema en la solución de problemas concretos.

Una vez expuestos los motivos por los que considero que se han alcanzado los

objetivos planteados al inicio de la tesis, así como algunas extensiones posibles, quiero expresar las razones que pienso justifican el trabajo realizado:

- Uno de los problemas más importantes a la hora de poner en práctica la Teoría de Test lo constituye la dificultad de calcular los testores mínimos. Hallar algoritmos eficientes para resolver este problema conducirá a la aplicación de la teoría a un mayor número de problemas concretos, lo que permitirá una mejor evaluación de la misma. En este sentido la tesis no depende de la efectividad de la Teoría de Test: el algoritmo diseñado y el sistema correspondiente servirán para poner a prueba la teoría y decidir, sobre una mayor experiencia con ella, si es o no adecuada para el tipo de problemas que pretende resolver.
- La tesis es un trabajo completo que abarca desde el diseño del algoritmo hasta su implementación en un lenguaje de programación, pasando por su expresión matemática y su prueba formal. Esto constituye una experiencia invaluable que influirá positivamente en mi trabajo posterior a cualquier nivel.
- Esta tesis se inició como un proyecto de trabajo entre un profesor del Departamento de Matemáticas —Pedro Miramontes Vidal— y un estudiante de los últimos semestres de la carrera —yo. El motivo por el cual me decidí a trabajar en dicho proyecto fue que me permitiría aplicar los conocimientos teóricos y prácticos adquiridos a lo largo de la carrera a la resolución de un *problema real*; de esa manera, al mismo tiempo que podría evaluar la solidez de mi preparación académica, obtendría la experiencia necesaria para el desarrollo de proyectos más importantes.

  A fin de cuentas, el hecho de que este proyecto haya terminado en una tesis es irrelevante: considero que este tipo de trabajos es muy importante para la educación de los estudiantes de la Facultad, en cualquiera de las carreras. Los proyectos a mediano y largo plazo involucran aspectos que no pueden ser desarrollados en las tareas usuales realizadas a lo largo de un semestre; además, el hecho de comprometerse en la realización de un proyecto "importante" proporciona muchas veces los incentivos necesarios para que el estudiante comience a *realizar investigación por su cuenta* y desarrolle así una capacidad indispensable en el área científica y deseable en cualquier profesional. Creo que este único punto, entendido con responsabilidad, justificaría la presentación de cualquier trabajo como una tesis profesional.

Bibliografía

(Aguila,81)

Aguila Feros, Luis y José Ruíz Shulcloper: "Algoritmo MB para la elaboración de la información k-valente en problemas de reconocimiento", *Revista de Ciencias Matemáticas*, Cuba, 1981.

(Bravo,82)

Bravo Martínez, Adrián y José Ruíz Shulcloper: "Un algoritmo natural para la elaboración de la información en problemas de reconocimiento", *Revista de Ciencias Matemáticas*, vol. III, 3, Cuba, 1982.

(Bravo,83)

Bravo Martínez, Adrián: "Algoritmo CT para el cálculo de los test típicos de una matriz k-valente", *Revista Ciencias Matemáticas*, vol. IV, 2, Cuba, 1983.

(Diukova,76)

Diukova, E. V.: "Acerca de un algoritmo para la construcción de los test típicos", *Colección de trabajos en Cibernética Matemática*, vol. I, Moscú, 1976.†

(Meisel,70)

Meisel, W. S.: *Computer-Oriented Approaches to Pattern Recognition*, Academic Press, 1970.

(Ruíz,84)

Ruíz Shulcloper, José: *Introducción a la Teoría de Test (Materiales para un curso de posgrado)*, 1984.

(Soto,80)

Soto Villaverde, Andrés, Alberto Fuentes Rodríguez y José Ruíz Shulcloper: "Una caracterización del concepto de test típico en términos de un subcojunto notable de columnas", *Revista de Ciencias Matemáticas*, vol. I, 23, Cuba, 1980.

(Wand,80)

Wand, M.: *Induction, Recursion and Programming*, North Holland, 1980

† Este es el trabajo original cuya traducción aparece en (Aguila,81)

(Watanabe,1969)
Watanabe, S.: *Methodologies of Pattern Recognition*, Academic Press, 1969.

(Wirth,76)
Wirth, Niklaus: *Algorithms + Data Structures = Programs*, Prentice-Hall, 1976.

Apéndice

```
{       Sistema de Reconocimiento de Patrones y de Clasificación
        Programa para el Reconocimiento de Patrones (REC)

   Los objetivos al diseñar este programa fueron los siguientes:
 • Poner en práctica los conceptos de la Teoría de Test relacionados con el reco-
   nocimiento de patrones.
 • Comprobar la eficiencia del algoritmo desarrollado para el cálculo de los testores
   mínimos de una muestra de aprendizaje.
 • Construir un pequeño programa de reconocimiento de patrones de propósito
   general (que formará parte de un sistema muchos más completo que incluirá
   reconocimiento de patrones y clasificación).

   Especificaciones:

 1. El sistema recibe como entrada a la muestra de aprendizaje en forma de matriz
    ("matriz de aprendizaje") a través de un archivo y produce como resultado dos
    archivos: uno con todos los testores mínimos, y otro con los pesos diferenciantes
    de los rasgos y los ideales de cada clase.
 2. Todos los archivos son de texto.
 3. Los valores de un rasgo en la matriz de aprendizaje mayores que los valores
    máximos correspondientes se consideran como indicación de ausencia de infor-
    mación.
 4. Los comentarios consisten de una o varias líneas que comienzan con ";".
 5. La marca de fin de testor en el archivo con los testores mínimos es un cero (0).
 6. El sistema genera archivos con los siguiente nombres: matriz.tm para el archivo
    con los testores mínimos y matriz.pd para el archivo con los pesos diferenciantes
    y los ideales de las clases.

                programó  : Rafael Morales Gamboa
                fecha     : 6 de Enero de 1987
                compilador : Turbo-Pascal versión 3.02
}

-------------------------------------------------------------------------------

{$G256,P512}                { Directivas para el compilador.        }
```

```
program REC;

const
  Ver = '2.20';                          { Número de la versión del sistema y  }
  Fecha = '23 de Noviembre de 1987';     { fecha de creación de la misma.      }
  Null = #0;                             { Definición de los nombres de        }
  Tab = #9;                              { algunos caracteres.                 }
  LF = #10;
  CR = #13;
  Space = #32;

  {Caracteres que se consideran como "blancos" y sirven para separar los datos en
  los archivos.}
  Blancos : set of char =
    [Null, Tab, LF, CR, Space];

  TM = 'matriz.tm';                      { Nombres asociados a los archivos    }
  PD = 'matriz.pd';                      { que produce el sistema.             }
  Temporal = 'temporal';
  MComentario = ';';                     { Marca del principio de comentario   }
                                         { en los archivos que maneja el       }
                                         { sistema.                            }
  SinErrores = 0;                        { Códigos correspondientes a los      }
  NoExisteArch = 1;                      { posibles estados y errores de los   }
  SinActividad = 2;                      { archivos.                           }
  ErrorDim = 4;
  NoHayDatos = 8;
  NoHayTestores = 16;
  EdoIncorrecto = 32;
  FinProceso = 64;
  LongNomArch = 64;                      { Longitud máxima para los nombres    }
                                         { de los archivos.                    }
  MaxRasgos = 50;                        { Número máximo de rasgos más uno.    }
  MaxObjetos = 200;                      { Número máximo de objetos más uno.   }
  Desconocido = 255;                     { Indicador de ausencia de información. }
  MaxTestores = 3000;                    { Número máximo de testores en la     }
                                         { lista.                              }
  Nulo = 0;                              { Apuntador nulo.                     }
  FinTestor = 0;                         { Marca de fin de testor.             }
  CodActivo = 1;                         { Códigos de rasgo activo y rasgo     }
  CodInactivo = 0;                       { inactivo en el archivo temporal.    }
```

```
type
  file_name = string[LongNomArch];    { Cadena para nombres de archivos. }

  {Estructura con la información concerniente a un archivo.}
  file_state =
    record
      Nombre : file_name;             { Nombre del archivo.              }
      Archivo: text;                  { Apuntador al archivo.            }
      Estado : integer                { Código que indica el estado      }
    end {record};                     { del archivo.                     }

  value = 0..Desconocido;             { Rango de valores para los rasgos. }
  f_index = 0..MaxRasgos;             { Indice para los rasgos.          }
  o_index = 0..MaxObjetos;            { Indice para los objetos.         }

  {Representación de un objeto.}
  object = array [1..MaxRasgos] of value;

  {Matriz de aprendizaje.}
  matriz = array [1..MaxObjetos] of object;

  {Vector con la partición del conjunto de objetos en clases.}
  vector_of_classes =
    record
      NumClases : o_index;            { Contiene el número de clases de la }
                                      { partición.                       }
      Clases : array [1..MaxObjetos]  { Contiene el índice del primer    }
        of o_index                    { objeto de cada clase.            }
    end {record};

  {Estructura con el estado y nivel de error de un rasgo.}
  feature_s_state =
    record
      Activo : boolean;               { Indica la condición del rasgo.   }
      Error : integer;                { Contiene el nivel de error.      }
      ValorMax: value;                { Contiene el valor máximo para    }
                                      { el rasgo.                        }
      PesoDif : real                  { Peso diferenciante del rasgo.    }
    end {record};

  {Vector con el estado y nivel de error de cada rasgo. Incluye un rasgo con índice
  cero para propósitos de cómputo.}
  vector_of_features = array [f_index] of feature_s_state;
```

```
{Estructura con la información correspondiente a un testor.}
testor =
  record
    Rasgos : set of f_index;        { Conjunto de índices de los rasgos  }
                                    { del testor.                        }
    Sig : integer;                  { Apuntador a otra estructura igual. }
  end {record};

{Bolsa de testores.}
bag = array [1..MaxTestores] of testor;
var
  ArchMA,                           { Arch. con la matriz de aprendizaje. }
  ArchTM,                           { Arch. con los testores mínimos.     }
  ArchPD,                           { Arch. con los pesos diferenciantes  }
                                    { de los rasgos y los ideales de clase. }
  ArchTP : file_state;              { Arch. temporal para guardar el      }
                                    { estado de las operaciones.          }
  NumRasgos : f_index;              { Número de rasgos.                   }
  NumObjetos : o_index;             { Número de objetos.                  }
  MatAprend : matriz;               { Matriz de aprendizaje.              }
  VectClasif : vector_of_classes;   { Vector con la partición en clases   }
                                    { del conjunto de objetos.            }
  VectRasgos : vector_of_features;  { Vector con el estado de cada rasgo  }
                                    { y el nivel de error correspondiente. }
  Bolsa : bag;                      { Espacio para la bolsa de los        }
                                    { testores mínimos.                   }
  TestoresMin,                      { Apuntador a la lista de testores.   }
  Libres : integer;                 { Apuntador al espacio                }
                                    { no utilizado.                       }

  {Las siguientes variables se utilizan únicamente para rastrear el comportamiento
  del programa.}
  Candidatos,                       { Número total de candidatos a testor }
                                    { mínimo revisados.                   }
  NumTestores,                      { Número total de testores revisados. }
  Permanecen : integer;             { Número de testores que permanecen   }
                                    { en la lista de testores mínimos.    }

-------------------------------------------------------------------------

{Rutinas para el manejos de archivos.}
```

```
{Verifica si un archivo existe en el directorio.}
function Existe ( Nombre : file_name ) : boolean;
var
  Archivo : file;
begin
  Assign(Archivo,Nombre);
  {$I-}
  Reset(Archivo);
  Close(Archivo);
  {$I+}
  Existe := (IOresult = 0);
end {Existe};

{Abre un archivo nuevo, le asigna un nombre y lo prepara para escritura;
además inicializa el código de error.}
procedure AbreEsc ( var EA : file_state; Nom : file_name );
begin
  with EA do
    begin
      Nombre := Nom;
      Assign(Archivo,Nombre);
      Rewrite(Archivo);
      Estado := SinErrores;
    end;
end {AbreEsc};

{Abre un archivo para lectura e inicializa el código de error.}
procedure AbreLec ( var EA : file_state; Nom : file_name );
begin
  with EA do
    begin
      Nombre := Nom;
      if not Existe(Nombre) then
        Estado := NoExisteArch
      else
        begin
          Assign(Archivo,Nombre);
          Reset(Archivo);
          Estado := SinErrores
        end {else}
```

```
    end {with};
end {AbreLec};

{Libera los archivos}
procedure LiberaArchivo ( var EA : file_state );
begin
  with EA do
    begin
      if (Estado and SinActividad) = 0 then
        Close(Archivo);
      Estado := Estado or SinActividad;
    end {with};
end {LiberaArchivo};

{Inicializa el estado de los archivos que maneja el sistema y las variables para el
seguimiento del programa.}
procedure Inicializa;
begin
  ArchMA.Estado := SinActividad;
  ArchTM.Estado := SinActividad;
  ArchPD.Estado := SinActividad;
  ArchTP.Estado := SinActividad;
  Candidatos := 0;                { Candidatos a testor revisados.      }
  NumTestores := 0;               { Testores encontrados.               }
  Permanecen := 0;                { Número de testores en la lista.     }
end {Inicializa};

{Devuelve el nombre de un archivo contenido en su "file_state".}
function DaNombre ( EA : file_state ): file_name;
begin
  DaNombre := EA.Nombre;
end {DaNombre};

{Da valor al nombre de un archivo.}
procedure AsignaNombre (var EA : file_state; Nom : file_name);
begin
  EA.Nombre := Nom
end {AsignaNombre};

{Devuelve el código de estado del archivo.}
```

```
function DaEstado (EA : file_state): integer;
begin
  DaEstado := EA.Estado
end {DaEstado};

{Da valor al código de estado del archivo.}
procedure AsignaEstado (var EA : file_state; Estado : integer);
begin
  EA.Estado := EA.Estado or Estado
end {AsignaEstado};

{Verifica si hay alguna señal de error en el estado del archivo.}
function HayError ( EA : file_state ) : boolean;
begin
  HayError := (DaEstado(EA) and (not SinActividad)) <> SinErrores;
end {HayError};

{Explica el tipo de error encontrado en el archivo correspondiente.}
procedure ExplicaError ( EA : file_state );
var
  Estado : integer;

function TipoError ( Cod , Error : integer ): boolean;
begin
  TipoError := (Cod and Error) <> 0;
end {TipoError};

begin
  if HayError(EA) then
    begin
      Estado := DaEstado(EA);
      WriteLn;
      if TipoError(Estado,FinProceso) then
        WriteLn('Acción suspendida'^G);
      if TipoError(Estado,NoExisteArch) then
        WriteLn('No existe el archivo "',DaNombre(EA),'"'^G);
      if TipoError(Estado,ErrorDim) then
        WriteLn('Error en las dimensiones en "',DaNombre(EA),'"'^G);
      if TipoError(Estado,NoHayDatos) then
        WriteLn('NoHayDatos en "',DaNombre(EA),'"'^G);
```

```
      if TipoError(Estado,NoHayTestores) then
        WriteLn('La matriz de aprendizaje no tiene testores mínimos'^G);
      if TipoError(Estado,EdoIncorrecto) then
        WriteLn('La información en "',DaNombre(EA),'" no es correcta'^G);
      WriteLn;
    end {if}
end {ExplicaError};

{Salta los comentarios al principio de un archivo. Cada comentario está precedido
por el caracter "MComentario" y dura hasta el fin de la línea.}
procedure SaltaComentarios ( var Archivo : text; var Estado : integer);
var
  NumComent : integer;
  C : char;
  Final : boolean;
begin
  Estado := SinErrores;
  Final := false;
  NumComent := 0;
  {Cuenta el número de líneas en blanco u ocupadas por comentarios al principio
  del archivo.}
  repeat
    C := Space;
    while (C in Blancos) and (not EoLn(Archivo)) do
      Read(Archivo,C);
    if (not (C in Blancos)) and (C <> MComentario) then
      Final := true
    else if EoF(Archivo) then
      begin
        Estado := NohayDatos;
        Final := true;
      end {else}
    else
      begin
        NumComent := NumComent + 1;
        ReadLn(Archivo);
    end {else};
  until Final;
```

```
  if Estado = SinErrores then
    begin
      Reset(Archivo);
      while (NumComent > 0) do
        begin
          ReadLn(Archivo);
          NumComent := NumComent - 1;
        end {while};
    end {if};
end {SaltaComentarios};
```

```
{Rutinas para la comunicación entre el usuario y el sistema.}

{Regresa "True" si el caracter leído fue "S" o "s", y "False" si fue}
{"N" o "n".}
function SiONo : boolean;
var
  C : char;
begin
  repeat
    Read(Kbd,C);
  until C in ['S','s','N','n'];
  WriteLn(C);
  SiONo := (Upcase(C) = 'S');
end {SiONo};

{Escribe el letrero inicial que anuncia el programa.}
procedure LetreroInicial;
begin
  WriteLn;
  WriteLn('Sistema de Reconocimiento de Patrones y Clasificación');
  WriteLn('Programa de Reconocimiento de Patrones (REC)');
  WriteLn('versión ',Ver,'  ',Fecha);
  WriteLn;
end {LetreroInicial};

{Verifica si se desea reiniciar una sesión suspendida.}
function EsReinicio : boolean;
```

```
begin
  Write('(Es reinicio de alguna sesión interrumpida? (s/n) ');
  EsReinicio := SiONo;
  WriteLn;
end {EsReinicio};
```

{Guarda en el archivo "temporal" el estado actual de los calculos, para que éstos puedan ser continuados si existe alguna interrupción.}

```
procedure GuardaEstadoActual (var Temp : file_state);
const
  Espacio = 5;
var
  j : f_index;
  t : integer;
begin
  WriteLn('Guardando el estado actual...'^G);
  AbreEsc(Temp,Temporal);
  if not HayError(Temp) then
    begin
      with Temp do
        begin
          Write(Archivo,NumObjetos:Espacio,NumRasgos:Espacio);
          Write(Archivo,Candidatos:Espacio,NumTestores:Espacio,
            Permanecen:Espacio);
          for j := 0 to NumRasgos do
            if VectRasgos[j].Activo then
              Write(Archivo,CodActivo:Espacio)
            else
              Write(Archivo,CodInactivo:Espacio);
          {Recorre la lista de testores.}
          t := Bolsa[TestoresMin].Sig;
          while ( t <> Nulo ) do
            begin
              with Bolsa[t] do
                begin
                  for j := 1 to NumRasgos do
                    if (j in Rasgos) then
                      Write(Archivo,j:Espacio);
```

```
                Write(Archivo,FinTestor:Espacio);
              end {with};
            t := Bolsa[t].Sig
          end {while};
        end {with};
      LiberaArchivo(Temp);
    end {if};
  WriteLn('Terminé de guardar el estado actual...'^G);
end {GuardaEstadoActual};
```

{Inicializa el vector con los estados de los rasgos. El primer candidato a testor mínimo es el conjunto de rasgos completo, por lo que inicialmente todas los rasgos estan activos. Pero aún no se calcula el grado de información proporcionado por cada uno de ellos.}

```
procedure IniEstadoRasgos ( var Archivo : text; var Estado : integer);
var
  n : f_index;
begin
  with VectRasgos[0] do
    begin
      Activo := true;
      Error := 0;
      PesoDif := 0;
    end {with};
  for n := 1 to NumRasgos do
    with VectRasgos[n] do
      begin
        Activo := true;
        Error := 0;
        PesoDif := 0;
        Read(Archivo, ValorMax);
        if Desconocido <= ValorMax then
          Estado := ErrorDim;
      end {with};
end {IniEstadoRasgos};
```

{Subrutinas para el procesamiento de la información relacionado con el reconocimiento de patrones.}

```
{Lee las dimensiones de la matriz, la matriz de aprendizaje y el vector de clasifi-
cación.}
procedure IniMatriz ( var Archivo : text; var Estado : integer );
const
  Espacio = 10;
var
  i : o_index;
  j : f_index;
  Rasgo : value;
  ClaseActual, ClaseAnterior : o_index;
begin
  Estado := SinErrores;
  {Lee el número de objetos, el de rasgos por objeto y el valor máximo de los
  rasgos.}
  Read(Archivo,NumObjetos);
  Read(Archivo,NumRasgos);
  WriteLn;
  WriteLn('Número de rasgos: ',NumRasgos:Espacio);
  WriteLn('Número de objetos: ',NumObjetos:Espacio);
  if (NumObjetos < 1) or (MaxObjetos <= NumObjetos) then
    Estado := ErrorDim
  else if (NumRasgos < 1) or (MaxRasgos <= NumRasgos) then
    Estado := ErrorDim
  else
    begin
      IniEstadoRasgos(Archivo,Estado);
      if Estado = SinErrores then
        begin
          {Lee la matriz y el vector de clasificación.}
          with VectClasif do
            begin
              ClaseAnterior := 0;
              NumClases := 0;
              for i := 1 to NumObjetos do begin
                for j := 1 to NumRasgos do with VectRasgos[j] do
                  begin
                    Read(Archivo,Rasgo);
                    if (Rasgo > ValorMax) then
                      MatAprend[i,j] := Desconocido
```

```
                    else
                      MatAprend[i,j] := Rasgo;
                  end {with};
                ReadLn(Archivo,ClaseActual);
                if ClaseActual <> ClaseAnterior then
                  begin
                    NumClases := NumClases + 1;
                    Clases[NumClases] := i;
                    ClaseAnterior := ClaseActual;
                  end {if};
                end {for};
              Clases[NumClases+1] := NumObjetos + 1;
            if (NumClases <= 1) then
              Estado := ErrorDim;
          end {with};
        WriteLn('Número de clases: ',VectClasif.NumClases:Espacio);
      end {if};
    end {else};
end {IniMatriz};

{Hace una verificación simple de las dimensiones de la matriz de aprendizaje comparandolas con las dimensiones contenidas en el archivo temporal.}
procedure VerificaDimensiones (var Archivo : text; var Estado : integer);
var
  NObjetos : o_indez;
  NRasgos : f_indez;
begin
  WriteLn('Verificando las dimensiones...');
  Estado := SinErrores;
  Read(Archivo,NObjetos,NRasgos);
  if (NObjetos<> NumObjetos) or (NRasgos <> NumRasgos) then
    Estado := ErrorDim
  else
    Read(Archivo,Candidatos,NumTestores,Permanecen);
end {VerificaDimensiones};

{Actualiza el vector con los estados de los rasgos a partir del archivo temporal.}
procedure ActEstadoRasgos (var Archivo : text; var Estado : integer);
var
  j : f_index;
```

```
  Cod : integer;
begin
  WriteLn('Actualizando el estado de los rasgos...');
  Estado := SinErrores;
  for j := 0 to NumRasgos do
    with VectRasgos[j] do
      begin
        Read(Archivo, Cod);
        if Cod = CodActivo then
          Activo := true
        else if Cod = CodInactivo then
          Activo := false
        else
          Estado := EdoIncorrecto
      end {with};
end {ActEstadoRasgos};

{Obtiene espacio para un testor.}
procedure Obtiene (var Nuevo : integer);
begin
  if Libres = Nulo then
    begin
      WriteLn('Obtiene. Error: No queda espacio para más testores.');
      Halt(1);
    end
  else
    begin
      Nuevo := Libres;
      Libres := Bolsa[Libres].Sig;
    end;
end {Obtiene};

{Libera el espacio ocupado por un testor.}
procedure Libera (var Obsoleto : integer);
begin
  Bolsa[Obsoleto].Sig := Libres;
  Libres := Obsoleto;
  Obsoleto := Nulo;
end {Libera};
```

```
{Inicializa la lista de testores encontrados. Inicialmente la lista contiene solamente
un testor falso (vacío).}
procedure IniTestoresMin ( var TM : integer );
var
  i : integer;
begin
  TM := 1;
  Bolsa[TM].Rasgos := [];
  Bolsa[TM].Sig := Nulo;
  Libres := 2;
  for i := Libres to MaxTestores - 1 do
    Bolsa[i].Sig := i + 1;
  Bolsa[MaxTestores].Sig := Nulo;
end {IniTestoresMin};

{Actualiza la lista de los testores encontrados a partir del archivo temporal.}
procedure ActTestoresMin (var Archivo : text; var Estado : integer);
var
  Ultimo, Nuevo : integer;

procedure LLenaTestor (var Archivo : text; var T : testor;
  var Estado : integer);
var
  R : Index;
begin
  with T do
    begin
      T.Rasgos := [];
      repeat
        Read(Archivo,R);
        if R <> FinTestor then
          Rasgos := Rasgos + [R];
      until (R = FinTestor) or EoF(Archivo);
    end {with};
  if R <> FinTestor then
    Estado := EdoIncorrecto
end {LlenaTestor};

begin
  WriteLn('Actualizando la lista de testores...');
  Estado := SinErrores;
```

```
  IniTestoresMin(TestoresMin);
  Ultimo := TestoresMin;
  while (not EoF(Archivo)) and (Estado = SinErrores) do
    begin
      Obtiene(Nuevo);
      LLenaTestor(Archivo,Bolsa[Nuevo],Estado);
      Bolsa[Ultimo].Sig := Nuevo;
      Bolsa[Nuevo].Sig := Nulo;
      Ultimo := Nuevo;
    end {while};
end {ActTestoresMin};

{Calcula que tan iguales son los objetos "x" e "y" con respecto a los rasgos activos.
Devuelve el índice del último rasgo para el que los objetos son iguales.}
function GradoIgualdadObj (x, y : o_index ) : f_index;
var
  Iguales : boolean;
  n : f_index;
begin
  Iguales := true;
  n := 1;
  while ( n <= NumRasgos ) and Iguales do
    begin
      if VectRasgos[n].Activo then
        Iguales := (MatAprend[x,n] = MatAprend[y,n])
          or (MatAprend[x,n] = Desconocido) or (MatAprend[y,n] = Descono-
        cido);
      n := n + 1;
    end {while};
  if Iguales then
    GradoIgualdadObj := n - 1
  else
    GradoIgualdadObj := n - 2;
end {GradoIgualdadObj};

{Verifica si el conjunto de rasgos activos constituye un testor.}
procedure AplicaRasgosActivos (var VP : vector_of_features;
  R0 : f_index; var Distingue : boolean );
var
  i, k, n : o_index;
```

```
  j : f_index;
  MaxIgualdad : f_index;
begin
  for j := R0 to NumRasgos do
    VP[j].Error := 0;
  with VectClasif do
    for n := 1 to NumClases do
    for i := Clases[n] to (Clases[n+1]-1) do
    for k := Clases[n+1] to NumObjetos do
      begin
        MaxIgualdad := GradoIgualdadObj(i,k);
        j := R0;
        while (j <= MaxIgualdad) do
          begin
            with VP[j] do
              Error := Error + 1;
            j := j + 1;
          end {while};
      end {for};
  Distingue := (VP[NumRasgos].Error = 0);
end {AplicaRasgosActivos};
```

{Elimina del testor los rasgos intrascendentes; esto es, aquellos rasgos que no aportan mayor información que la dada por los rasgos anteriores.}

```
procedure ReduceTestor ( var VP : vector_of_features );
const
  ErrorInicial = -1;
var
  n : f_index;
  GradError : integer;
begin
  GradError := ErrorInicial;
  for n := 1 to NumRasgos do
    with VP[n] do
      if GradError = Error then
        Activo := false
      else if Activo then
        GradError := Error;
end {ReduceTestor};
```

{Elimina el último rasgo del testor anterior. En caso que el conjunto de los rasgos activos no sea un testor, se modifica ese conjunto de la siguiente manera:
1. Se activan los últimos rasgos desactivados consecutivos.
2. Se desactiva el último rasgos activo anterior a éstos.}

```
procedure SigCandidTestor (var VP : vector_of_features;
  var R0 : f_index; Distingue : boolean; var Fin : boolean );
var
  n : f_index;
begin
  if Distingue then
    begin
      n := NumRasgos;
      while (not VP[n].Activo) do
        begin
          VP[n].Activo := true;
          n := n - 1;
        end {while};
      if n = 0 then
        Fin := true
      else
        begin
          VP[n].Activo := false;
          R0 := n;
        end {else};
    end {if}
  else
    begin
      n := NumRasgos;
      while VP[n].Activo and (n > 0) do
        n := n - 1;
      while (not VP[n].Activo) do
        begin
          VP[n].Activo := true;
          n := n - 1;
        end {while};
      if n = 0 then
        Fin := true
      else
        begin
          VP[n].Activo := false;
```

```
            R0 := n;
          end {else};
      end {else};
end {SigCandidTestor};

{Incluye el nuevo testor en la lista de testores.}
procedure GuardaTestor ( VP : vector_of_features;
   TM : integer; var Permanecen : integer );
var
  i : o_index;
  Nuevo, Viejo, Aux, Temp : integer;

procedure LlenaTestor ( Nuevo : integer; VP : vector_of_features );
var
  n : f_index;
begin
  with Bolsa[Nuevo] do
    begin
      Rasgos := [];
      for n := 1 to NumRasgos do
        if VP[n].Activo then
          Rasgos := Rasgos + [n];
      Sig := Nulo;
    end {with};
end {LlenaTestor};

begin
  Obtiene(Nuevo);
  LlenaTestor(Nuevo, VP);
  Viejo := TM;
  Aux := TM;
  {Recorre la lista de testores.}
  while ( Bolsa[Viejo].Sig <> Nulo ) do
    begin
      if Bolsa[Nuevo].Rasgos <= Bolsa[Viejo].Rasgos then
        begin
          Viejo := Bolsa[Viejo].Sig;
          Temp := Bolsa[Aux].Sig;
          Bolsa[Aux].Sig := Viejo;
          Libera(Temp);
          Permanecen := Permanecen - 1;
```

```
        end {if}
      else
        begin
          Aux := Viejo;
          Viejo := Bolsa[Viejo].Sig;
        end {else};
    end {while};
  {Compara con el último testor de la lista}
  if Bolsa[Nuevo].Rasgos <= Bolsa[Viejo].Rasgos then
    begin
      Bolsa[Aux].Sig := Nuevo;
      Libera(Viejo);
      Permanecen := Permanecen - 1
    end {if}
  else
    Bolsa[Viejo].Sig := Nuevo;
  Permanecen := Permanecen + 1;
end {GuardaTestor};

{Imprime la lista de testores.}
procedure EscribeTestores ( var TM : integer; var Archivo : text );
const
  PorRenglon = 16;
  Espacio = 5;
var
  Num_Testores : integer;
  Contador : integer;
  i : o_indez;
  EnRenglon : integer;
begin
  {Cuenta los testores mínimos.}
  Contador := Bolsa[TM].Sig;
  Num_Testores := 0;
  while ( Contador <> Nulo ) do
    begin
      Contador := Bolsa[Contador].Sig;
      Num_Testores := Num_Testores + 1;
    end {while};
  WriteLn(Archivo,Num_Testores);
  WriteLn(Archivo,NumRasgos);
```

```
  Contador := Bolsa[TM].Sig;
  {Recorre la lista de testores}
  while ( Contador <> Nulo ) do
    with Bolsa[Contador] do
      begin
        EnRenglon := 0;
        for i := 1 to NumRasgos do
          if (i in Rasgos) then
            begin
              Write(Archivo,i:Espacio);
              EnRenglon := EnRenglon + 1;
              if EnRenglon = PorRenglon then
                begin
                  WriteLn(Archivo);
                  EnRenglon := 0;
                end {if};
            end {if};
        WriteLn(Archivo,FinTestor:Espacio);
        Contador := Bolsa[Contador].Sig
      end {with};
  Contador := TM;
  TM := Bolsa[TM].Sig;
  Libera(Contador);                    { Elimina el testor falso (vacío).      }
end {EscribeTestores};

{Muestra las condiciones de cada uno de los rasgos; esto es, su actividad y el nivel
de error.}
procedure MuestraRasgos;
var
  j: f_index;
begin
  for j := 1 to NumRasgos do
    with VectRasgos[j] do
      WriteLn('Indice: ',j:5,'Activo: ',Activo:10,'Error: ',Error:5);
  WriteLn;
end {MuestraRasgos};

{Imprime, en binario, el número máximo de candidatos a testores mínimos que
faltan por revisar.}
procedure MuestraRasgosActivos;
```

```
var
  j: f_index;
begin
  WriteLn;
  Write('Quedan por revisar (en binario): ');
  for j := 1 to NumRasgos do
    with VectRasgos[j] do
      if Activo then
        Write('1')
      else
        Write('0');
  WriteLn;
end {MuestraRasgosActivos};

procedure GeneraTestores ( var ArchTM, ArchTP : file_state );
const
  PresentaRev = 100;
  PresentaRasgos = 100;
  CopiaEnDisco = 500;
  Espacio = 10;
var
  EsTestor,                          { Indica si el conjunto de rasgos   }
                                     { escogidos es un testor.           }
  Fin : boolean;                     { Indica si ya no existen más testor. }
  R0 : f_index;                      { Primer rasgo para el que se       }
                                     { recalcula el error.               }
begin
  WriteLn;
  WriteLn('Calculando los testores mínimos...');
  WriteLn;
  Fin := false;
  R0 := 1;
  repeat
    if (Candidatos MOD PresentaRev) = 0 then
      begin
        WriteLn('Candidatos: ',Candidatos:Espacio);
        WriteLn('Testores: ',NumTestores:Espacio);
        WriteLn('Permanecen: ',Permanecen:Espacio);
        WriteLn;
```

```
        if (Candidatos MOD PresentaRasgos) = 0 then
          MuestraRasgosActivos;
      end;
    {Genera la información correspondiente al conjunto de rasgos activos.}
    AplicaRasgosActivos(VectRasgos,R0,EsTestor);
    if EsTestor then
      begin
        {Reduce el testor eliminando rasgos que no aportan más información.}
        ReduceTestor(VectRasgos);
        NumTestores := NumTestores + 1;
        {Incluye el testor en la lista.}
        GuardaTestor(VectRasgos,TestoresMin,Permanecen);
      end {if};
    {Genera el siguiente conjunto de rasgos activos.}
    SigCandidTestor(VectRasgos,R0,EsTestor,Fin);
    Candidatos := Candidatos + 1;
    if (Candidatos MOD CopiaEnDisco) = 0 then
      GuardaEstadoActual(ArchTP);
  until Fin;
  WriteLn;
  WriteLn('Subconjuntos revisados: ',Candidatos:Espacio);
  WriteLn('Testores encontrados: ',NumTestores:Espacio);
  WriteLn('Testores mínimos: ',Permanecen:Espacio);
  WriteLn;
  WriteLn('Almacenando los testores mínimos en "',TM,'"...'G);
  AbreEsc(ArchTM,TM);
  with ArchTM do
    EscribeTestores(TestoresMin,Archivo);
  LiberaArchivo(ArchTM);
end {GeneraTestores};

{Vacía la lista de testores mínimos.}
procedure LiberaTestores ( var TM : integer );
var
  Libertador : integer;
begin
  Libertador := TM;
  while (Libertador <> Nulo) do
    begin
      TM := Bolsa[TM].Sig;
```

```
      Libera(Libertador);
      Libertador := TM;
    end {while};
end {LiberaTestores};

{Genera los pesos diferenciantes de cada uno de los rasgos.}
procedure GeneraPesosDif ( var ArchPD : file_state );
const
  Espacio = 18;
  PorRenglon = 4;
var
  Aux : integer;
  NumTestores, Apariciones : integer;
  j : f_index;
  EnRenglon : integer;
begin
  if Permanecen = 0 then
    AsignaEstado(ArchPD,NoHayTestores)
  else
    begin
      Aux := TestoresMin;
      while Aux <> Nulo do
        begin
          for j := 1 to NumRasgos do
            if j in Bolsa[Aux].Rasgos then
              with VectRasgos[j] do
                PesoDif := PesoDif + 1;
          Aux := Bolsa[Aux].Sig;
        end {while};
      for j := 1 to NumRasgos do
        with VectRasgos[j] do
          PesoDif := PesoDif/Permanecen;
      {Escribe los pesos calculados en el archivo.}
      WriteLn('Almacenando los pesos diferenciantes en ',PD,'...'G);
      AbreEsc(ArchPD,PD);
      with ArchPD do
        begin
          WriteLn(Archivo,NumRasgos);
```

```
            for j := 1 to NumRasgos do
              with VectRasgos[j] do
                WriteLn(Archivo, ValorMax:Espacio,PesoDif:Espacio);
            WriteLn(Archivo);
          end {with};
      end {else};
    LiberaTestores(TestoresMin);
  end {GeneraPesosDif};

{Calcula el peso informativo de un objeto relativo a su clase.}
procedure CalculaPesoInformativo (i, k : o_index; var PesoInf : real);
var
  j : f_index;
  l, Iguales : o_index;
begin
  PesoInf := 0;
  with VectClasif do
    begin
      for j := 1 to NumRasgos do
        begin
          Iguales := 0;
          if MatAprend[i,j] <> Desconocido then
            for l := Clases[k] to Clases[k+1] - 1 do
              if MatAprend[i,j] = MatAprend[l,j] then
                Iguales := Iguales + 1;
          PesoInf := PesoInf + VectRasgos[j].PesoDif*Iguales;
        end {for};
      PesoInf := PesoInf/(Clases[k+1] - Clases[k]);
    end {with};
end {CalculaPesoInformativo};

{Genera los ideales de cada una de las clases que componen a la población.}
procedure GeneraIdeales (var ArchPD : file_state);
const
  Espacio = 5;
  EspReal = 18;
type
  set_of_objets = set of o_index;
var
  i, k, NObjetos, Desplazamiento : o_index;
```

```
  PesoMax, PesoInf : real;
  Ideales : set_of_objets;

{Escribe los ideales de una clase en el archivo correspondiente, encabezando con el
número de ideales y el peso informativo de los mismos.}
procedure EscribeIdeales (var Archivo : text;
  Ideales : set_of_objets; Desp, NObj : o_index; PesoInf : real);
var
  i, NumIdeales : o_index;
  j : f_index;
begin
  NumIdeales := 0;
  for i := 0 to NObj - 1 do
    if i in Ideales then
      NumIdeales := NumIdeales + 1;
  WriteLn(Archivo,NumIdeales:Espacio,PesoInf:EspReal);
  for i := 0 to NObj - 1 do
    if i in Ideales then
      begin
        for j := 1 to NumRasgos do
          Write(Archivo,MatAprend[i+Desp,j]:Espacio);
        WriteLn(Archivo);
      end {if};
end {EscribeIdeales};

begin
  WriteLn('Almacenando los ideales de clase en ',PD,'...'G);
  with VectClasif do
    with ArchPD do
      begin
        WriteLn(Archivo);
        WriteLn(Archivo,NumClases);
        for k := 1 to NumClases do
          begin
            PesoMax := 0;
            Ideales := [];
            NObjetos := Clases[k+1] - Clases[k];
            Desplazamiento := Clases[k];
            for i := 0 to NObjetos - 1 do
              begin
                CalculaPesoInformativo(Desplazamiento+i,k,PesoInf);
```

```
                    if PesoMax < PesoInf then
                      begin
                        PesoMax := PesoInf;
                        Ideales := [i];
                      end {if}
                    else if PesoMax = PesoInf then
                      Ideales := Ideales + [i];
                  end {for};
                EscribeIdeales(Archivo,Ideales,Desplazamiento,NObjetos,PesoMax);
              end {for};
            end {with};
  LiberaArchivo(ArchPD);
end {GeneraIdeales};
```

{Rutinas para obtener los valores iniciales del sistema.}

```
{Lee el nombre del archivo con la matriz de aprendizaje.}
procedure IniArchMat ( var ArchMA : file_state);
var
  Fin : boolean;
  Nom : file_name;
begin
  Fin := false;
  repeat
    WriteLn;
    WriteLn('¿Cuál es el nombre del archivo');
    Write (' con la matriz de aprendizaje? ');
    ReadLn(Nom);
    if Length(Nom) = 0 then
      begin
        {No se desea continuar con el reconocimiento.}
        AsignaEstado(ArchMA,SinActividad);
        AsignaEstado(ArchMA,FinProceso);
        Fin := true;
      end {if}
    else
      begin
        AbreLee(ArchMA,Nom);
```

```
        If not HayError(ArchMA) then
          Fin := true
        else
          begin
            WriteLn;
            WriteLn('No existe ningún archivo llamado ',Nom);
          end {else}
      end {else}
  until Fin;
end {IniArchMat};

{Asigna los valores iniciales a las variables que almacenaran:
  • la matriz de aprendizaje,
  • el vector de clasificación,
  • el estado de los rasgos,
  • el número de objetos,
  • el número de rasgos, y
  • la lista de testores encontrados.}
procedure InicializaDatos (var ArchMA, ArchTP : file_state);
var
  Estado : integer;
begin
  Writeln('Inicializando los datos...');
  IniArchMat(ArchMA);
  if not HayError(ArchMA) then
    begin
      SaltaComentarios(ArchMA.Archivo,Estado);
      AsignaEstado(ArchMA,Estado);
    end {if};
  if not HayError(ArchMA) then
    begin
      IniMatriz(ArchMA.Archivo,Estado);
      AsignaEstado(ArchMA,Estado);
    end {if};
  LiberaArchivo(ArchMA);
  if not HayError(ArchMA) then
    IniTestoresMin(TestoresMin);
end {InicializaDatos};
```

```
{Recupera los valores de las variables después de la suspensión de la ejecución del
programa.}
procedure ActualizaDatos (var ArchMA, ArchTP : file_state);
var
  Estado : integer;
begin
  Writeln('Actualizando los datos...'G);
  IniArchMat(ArchMA);
  if not HayError(ArchMA) then
    begin
      SaltaComentarios(ArchMA.Archivo,Estado);
      AsignaEstado(ArchMA,Estado);
    end {if};
  if not HayError(ArchMA) then
    begin
      IniMatriz(ArchMA.Archivo,Estado);
      AsignaEstado(ArchMA,Estado);
    end {if};
  LiberaArchivo(ArchMA);
  if not HayError(ArchMA) then
    begin
      AbreLec(ArchTP,Temporal);
      if not HayError(ArchTP) then
        begin
          SaltaComentarios(ArchTP.Archivo,Estado);
          AsignaEstado(ArchTP,Estado);
        end {if};
      if not HayError(ArchTP) then
        begin
          VerificaDimensiones(ArchTP.Archivo,Estado);
          AsignaEstado(ArchTP,Estado);
        end {if};
      if not HayError(ArchTP) then
        begin
          ActEstadoRasgos(ArchTP.Archivo,Estado);
          AsignaEstado(ArchTP,Estado);
        end {if};
      if not HayError(ArchTP) then
        begin
          ActTestoresMin(ArchTP.Archivo,Estado);
```

```
        AsignaEstado(ArchTP,Estado);
      end {if}
    end {if};
  LiberaArchivo(ArchTP);
end {ActualizaDatos};
```

{La subrutina "EstudiaMatriz" es la encargada de estudiar la matriz de aprendizaje, generar los testores mínimos, calcular los pesos diferenciantes de los rasgos y los ideales de las clases.}

```
procedure EstudiaMatriz (Reinicio : boolean);
var
  Estado : integer;
begin
  if Reinicio then
    ActualizaDatos(ArchMA,ArchTP)
  else
    InicializaDatos(ArchMA,ArchTP);
  if not (HayError(ArchMA) or HayError(ArchTP)) then
    begin
      GeneraTestores(ArchTM,ArchTP);
      if not (HayError(ArchTM) or HayError(ArchTP)) then
        begin
          GeneraPesosDif(ArchPD);
          GeneraIdeales(ArchPD);
        end {if};
    end {if};
  ExplicaError(ArchMA);
  ExplicaError(ArchTP);
  ExplicaError(ArchTM);
  ExplicaError(ArchPD);
end {EstudiaMatriz};
```

{Cuerpo del programa principal.}

```
Begin
  Inicializa;
  LetreroInicial;
  EstudiaMatriz(EsReinicio);
End.
```

{ Sistema de Reconocimiento de Patrones y de Clasificación
Programa para Clasificación de Objetos (CLA)

El objetivo de este programa es utilizar la información obtenida con el Programa para el Reconocimiento de Patrones (REC), para clasificar objetos, representados mediante vectores de longitud fija. Así, este programa constituye la parte complementaria del Sistema de Reconocimiento de Patrones basado en la Teoría de Test.

El programa recibe como información inicial (datos de entrada) los ideales de cada una de las clases que componen el universo de objetos y la descripción de los objetos que se desean clasificar, produciendo como resultado la clasificación de dichos objetos.

Especificaciones:

1. Todos los archivos son de texto, para facilitar la revisión y edición de los datos.
2. Los valores de un rasgo en la matriz de ideales mayores que los valores máximos correspondientes se consideran como indicación de ausencia de información.
3. Los comentarios consisten de una o varias líneas que comienzan con ";".
4. El sistema genera archivos con los siguiente nombres: *objetos.cl* para el archivo con la clasificación de los objetos.

programó : Rafael Morales Gamboa
fecha : 1 de Diciembre de 1987
compilador : Turbo-Pascal versión 3.02

}

```
{$G256,P512}
program CLA;

const
  Ver = '1.00';                    { Número de la versión del sistema y  }
  Fecha = '1 de Diciembre de 1987'; { fecha de creacion de la misma.     }

  Null = #0;
  Tab = #9;
  LF = #10;
  CR = #13;
```

```
  Space = #32;
  Blancos : set of char =
    [Null, Tab, LF, CR, Space];

  LongNomArch = 64;          { Longitud máxima para los nombres }
                             { de los archivos.                 }
  LongMensaje = 80;          { Longitud máxima de los mensajes  }
                             { de error.                        }
  MComentario = ';';         { Marca del principio de comentario }
                             { en los archivos que maneja el    }
                             { sistema.                         }
  ArchPD = 'matriz.pd';      { Nombre del archivo con los ideales. }
  ArchCL = 'objetos.cl';     { Nombre del archivo con los objetos. }
  MaxRasgos = 50;            { Número máximo de rasgos más uno. }
  CadMaxRasgos = '50';
  MaxIdeales = 200;          { Número máximo de objetos más uno. }
  CadMaxIdeales = '200';
  Desconocido = 255;         { Indicador de ausencia de información. }

type
  file_name = string[LongNomArch];   { Cadena para nombres de archivos. }
  message = string[LongMensaje];     { Cadena para mensajes de error.   }
  value = 0..Desconocido;            { Rango de valores para los rasgos. }
  f_index = 0..MaxRasgos;            { Indice para los rasgos.          }
  o_index = 0..MaxIdeales;           { Indice para los objetos.         }

  {Representación de un objeto. }
  object = array [1..MaxRasgos] of value;

  {Matriz con los ideales de clase. }
  matrix = array [1..MaxIdeales] of object;

  {Vector con la partición del conjunto de objetos en clases. }
  vector_of_classes =
    record
      NumClases : integer;       { Contiene el número de clases de la }
                                 { partición.                         }
      Clases : array [1..MaxIdeales]  { Contiene el índice del primer }
        of o_index;              { objeto de cada clase.              }
      Cercan : array [1..MaxIdeales]  { Contiene la cercanía de un objeto }
        of real;                 { dado a la clase.                   }
    end {record};
```

```
  {Estructura con el estado y nivel de error de un rasgo. }
  feature_s_state =
    record
      Activo : boolean;          { Indica la condición del rasgo.        }
      Error : integer;           { Contiene el nivel de error.           }
      ValorMax: value;           { Contiene el valor máximo para         }
                                 { el rasgo.                             }
      PesoDif : real             { Peso diferenciante del rasgo.         }
    end {record};

  {Vector con el estado y nivel de error de cada rasgo. Incluye un rasgo con índice
  cero para propósitos de cómputo. }
  vector_of_features =
    array [f_index] of feature_s_state;

var
  ArchOB : file_name;
  NumRasgos : integer;           { Número de rasgos.                     }
  NumObjetos : integer;          { Número de objetos.                    }
  MatIdeales : matriz;           { Matriz de aprendizaje.                }
  VectClasif : vector_of_classes; { Vector con la partición en clases    }
                                 { del conjunto de objetos.              }
  VectRasgos : vector_of_features; { Vector con el estado de cada rasgo. }
```

```
{Procedimientos para el manejo de los errores. }

procedure Asume (Cond : boolean; Mensaje : message);
begin
  if not Cond then
    begin
      WriteLn;
      WriteLn(Mensaje, ^G);
      Halt(1);
    end {if};
end {Asume};

procedure AsumeOCierra (Cond: boolean;
  var Archivo : text; Mensaje : message);
```

```
begin
  if not Cond then
    begin
      Close(Archivo);
      WriteLn(Mensaje, ^G);
      Halt(1);
    end {if};
end {AsumeOCierra};
```

```
{Rutinas para el manejos de archivos. }

{Verifica si un archivo existe en el directorio. }
function Existe ( Nombre : file_name ) : boolean;
var
  Archivo : file;
begin
  Assign(Archivo,Nombre);
  {$I-}
  Reset(Archivo);
  Close(Archivo);
  {$I+}
  Existe := (IOresult = 0);
end {Existe};

{Abre un archivo y lo prepara para lectura. }
procedure AbreLec (var Archivo : text; Nombre : file_name);
begin
  Assign(Archivo,Nombre);
  Reset(Archivo);
end {AbreLec};

{Abre un archivo y lo prepara para escritura. }
procedure AbreEsc (var Archivo : text; Nombre : file_name);
begin
  Assign(Archivo,Nombre);
  Rewrite(Archivo);
end {AbreEsc};

{Libera un archivo de uso. }
```

```
procedure Cierra (var Archivo : text);
begin
  Close(Archivo);
end {Cierra};

{Salta los comentarios al principio de un archivo. Cada comentario está precedido
por el caracter "MComentario" y dura hasta el fin de la línea. }
procedure SaltaComentarios (var Archivo : text; Nombre : file_name);
var
  NumComentarios : integer;
  C : char;
  Final : boolean;
begin
  {Cuenta el número de líneas en blanco u ocupadas por comentarios al principio
  del archivo. }
  NumComentarios := 0;
  repeat
    C := Space;
    while (C in Blancos) and (not EoLn(Archivo)) do
      Read(Archivo,C);
    if (not (C in Blancos)) and (C <> MComentario) then
      Final := true
    else if EoF(Archivo) then
      AsumeOCierra(false,Archivo,
        'No hay datos en el archivo "'+Nombre+'".')
    else
      begin
        NumComentarios := NumComentarios + 1;
        ReadLn(Archivo);
      end {else};
  until Final;
  Cierra(Archivo);
  AbreLec(Archivo,Nombre);
  while (NumComentarios > 0) do
    begin
      ReadLn(Archivo);
      NumComentarios := NumComentarios - 1;
    end {while};
end {SaltaComentarios};
```

```
{Rutinas para la comunicación entre el usuario y el sistema. }

{Regresa "True" si el caracter leído fue "S" o "s", y "False" si fue }
{"N" o "n". }
function SiONo : boolean;
var
  C : char;
begin
  repeat
    Read(Kbd,C);
  until C in ['S','s','N','n'];
  WriteLn(C);
  SiONo := (Upcase(C) = 'S');
end {SiONo};

{Escribe el letrero inicial que anuncia el programa. }
procedure LetreroInicial;
begin
  WriteLn;
  WriteLn('Sistema de Reconocimiento de Patrones y Clasificación');
  WriteLn('Programa para la Clasificación de Objetos (CLA)');
  WriteLn('versión ',Ver,' ',Fecha);
  WriteLn;
end {LetreroInicial};

{Lee el nombre del archivo que contiene los objetos por clasificar. }
procedure IniArchivoObjetos (var Nombre : file_name);
var
  Fin : boolean;
begin
  Fin := false;
  repeat
    WriteLn;
    WriteLn('¿Cuál es el nombre del archivo');
    Write (' con los objetos por clasificar? ');
    ReadLn(Nombre);
    Asume(Length(Nombre) > 0,'Acción suspendida.');
```

```
    if not Existe(Nombre) then
      begin
        WriteLn;
        WriteLn('No existe ningún archivo llamado ',Nombre,'');
      end {if}
    else
      Fin := true;
  until Fin;
end {IniArchivoObjetos};
```

{Procedimientos y funciones relacionadas con el cálculo de las funciones de cercanía.
}

```
{Devuelve la distancia entre dos valores de un rasgo. }
function Distancia (V1, V2, ValorMax : value): real;
begin
  if ValorMax = 0 then
    Distancia := 0
  else if (V1 = Desconocido) or (V2 = Desconocido) then
    Distancia := 0
  else
    Distancia := abs(V1 - V2)/ValorMax
end {Distancia};

{Calcula la cercanía entre dos objetos. }
function CercanObjetos (Obj1, Obj2 : object): real;
var
  Dist : real;
  j : f_index;
begin
  Dist := 0;
  for j := 1 to NumRasgos do
    with VectRasgos[j] do
      Dist := Dist + Distancia(Obj1[j],Obj2[j], ValorMax);
  CercanObjetos := 1 - (Dist/NumRasgos);
end {CercanObjetos};
```

{Calcula la cercanía de un objeto a una clase, misma que se define como el máximo de las cercanías del objeto a los ideales de la clase. }

```
function CercanClase (Objeto : object; k : o_index): real;
var
  Max, CO : real;
  i : o_index;
begin
  Max := 0;
  CO := 0;
  with VectClasif do
    for i := Clases[k] to Clases[k+1] - 1 do
      begin
        CO := CercanObjetos(Objeto,MatIdeales[i]);
        if Max < CO then
          Max := CO
      end {for};
  CercanClase := Max;
end {CercanClase};

{Calcula las cercanías de los objetos a cada una de las clases. }
procedure CalculaCercan (Objeto : object; var Archivo : text);
const
  Espacio = 18;
  PorReng = 4;
var
  Max : real;
  k : o_index;
  EnReng : integer;
begin
  Max := 0;
  with VectClasif do
    begin
      for k := 1 to NumClases do
        begin
          Cercan[k] := CercanClase(Objeto,k);
          if Max < Cercan[k] then
            Max := Cercan[k];
        end {for};
      EnReng := 0;
```

```
      for k := 1 to NumClases do
        begin
          if Max > 0 then
            Write(Archivo,(Cercan[k]/Max):Espacio)
          else
            Write(Archivo,Max:Espacio);
          EnReng := EnReng + 1;
          if EnReng = PorReng then
            begin
              WriteLn(Archivo);
              EnReng := 0;
            end {if};
        end {for};
      WriteLn(Archivo);
      WriteLn(Archivo);
    end {with};
end {CalculaCercan};
```

{Procedimientos y funciones relacionados con la clasificación de los objetos. }

{Verifica que las dimensiones contenidas en el archivo de objetos correspondan con las obtenidas del archivo con los ideales. }

```
procedure VerificaDimensiones (var Archivo : text);
var
  NRasgos : integer;
begin
  Read(Archivo,NumObjetos);
  WriteLn('Número de objetos: ',NumObjetos);
  AsumeOCierra(NumObjetos > 0,Archivo,
    'Número incorrecto de objetos en el archivo.');
  Read(Archivo,NRasgos);
  AsumeOCierra(NRasgos = NumRasgos,Archivo, 'Número incompatible de rasgos.');
end {VerificaDimensiones};
```

{Lee un objeto del archivo. }

```
procedure LeeObjeto (var Archivo : text; var Objeto : object );
var
  j : f_index;
```

```
  Temp : integer;
begin
  for j := 1 to NumRasgos do
    with VectRasgos[j] do
      begin
        Read(Archivo, Temp);
        if (Temp < 0) or (ValorMax < Temp) then
          Objeto[j] := Desconocido
        else
          Objeto[j] := Temp;
      end {with};
end {LeeObjeto};

{Realiza la clasificación de los objetos y la deposita en un archivo. }
procedure Clasifica (ArchOB, ArchCL : file_name);
const
  Espacio = 5;
var
  Entrada, Salida : text;
  Objeto : object;
  i : integer;
begin
  WriteLn;
  WriteLn('Clasificando...');
  AbreLec(Entrada,ArchOB);
  VerificaDimensiones(Entrada);
  AbreEsc(Salida,ArchCL);
  WriteLn(Salida,NumObjetos:Espacio,NumRasgos:Espacio);
  with VectClasif do
    WriteLn(Salida,NumClases:Espacio);
  for i := 1 to NumObjetos do
    begin
      LeeObjeto(Entrada,Objeto);
      CalculaCercan(Objeto,Salida);
    end {for};
  Cierra(Entrada);
  Cierra(Salida);
  WriteLn;
  WriteLn('La clasificación de los objetos se guardó en el archivo ',
    ArchCL,'.'^G);
```

```
end {Clasifica};
```

{Procedimientos y funciones relacionados con la adquisición de la información contenida en los archivos. }

{Lee el número de rasgos que describen a los objetos y, para cada rasgo, el valor máximo permitido. }
```
procedure LeeDatosRasgos (var Archivo : text);
var
  j : f_index;
begin
  WriteLn('Inicializando variables...');
  WriteLn;
  Read(Archivo,NumRasgos);
  WriteLn('Número de rasgos: ',NumRasgos);
  AsumeOCierra(NumRasgos < MaxRasgos,Archivo,
    'El número de rasgos debe ser menor a '+CadMaxRasgos);
  for j := 1 to NumRasgos do
    with VectRasgos[j] do
      Read(Archivo,ValorMax,PesoDif);
end {LeeDatosRasgos};
```

{Lee los ideales de las clases. }
```
procedure LeeIdeales (var Archivo : text);
var
  TotalIdeales, NumIdeales : integer;
  PesoInfIdeales : real;
  N, i : o_index;
  j : f_index;
  Menor : boolean;
begin
  with VectClasif do
    begin
      Read(Archivo,NumClases);
      WriteLn('Número de clases: ',NumClases);
      WriteLn;
      WriteLn('Cargando los ideales de cada clase...');
      WriteLn;
```

```
      Menor := (0 < NumClases) and (NumClases < MaxIdeales);
      TotalIdeales := 0;
      N := 1;
      while Menor and (N <= NumClases) do
        begin
          Clases[N] := TotalIdeales + 1;
          Read(Archivo,NumIdeales,PesoInfIdeales);
          Menor := (TotalIdeales + NumIdeales) < MaxIdeales;
          if Menor then
            for i := TotalIdeales+1 to TotalIdeales+NumIdeales do
              LeeObjeto(Archivo,MatIdeales[i]);
          TotalIdeales := TotalIdeales + NumIdeales;
          N := N + 1;
        end {while};
      AsumeOCierra(Menor,Archivo,
      quad 'El número de ideales debe ser menor que '+CadMaxIdeales);
      Clases[N] := TotalIdeales + 1;
      WriteLn('Total de ideales: ',TotalIdeales);
    end {with};
end {LeeIdeales};

{Lee el archivo con los datos necesarios para la clasificación. }
procedure IniBaseDatos (Nombre : file_name);
var
  Archivo : text;
begin
  Asume(Existe(Nombre),'No existe el archivo "'+Nombre+'" con los ideales.');
  AbreLee(Archivo,Nombre);
  SaltaComentarios(Archivo,Nombre);
  LeeDatosRasgos(Archivo);
  LeeIdeales(Archivo);
  Cierra(Archivo);
end {IniBaseDatos};
```

{Procedimientos y funciones que dirigen el flujo de ejecución del programa. }

{Inicializa los datos a partir de un archivo, lee el nombre del archivo con los objetos por clasificar y efectúa la clasificación. }

```
procedure ClasificaObjetos;
begin
  IniBaseDatos(ArchPD);
  IniArchivoObjetos(ArchOB);
  Clasifica(ArchOB,ArchCL);
end {ClasificaObjetos};
```

```
{Cuerpo del programa principal }

begin
  LetreroInicial;
  ClasificaObjetos;
end.
```

```
ejrec

C:\TMP>echo off
Ejemplo de ejecución del Sistema de Reconocimiento de Patrones
y de Clasificación (RECLA), en su parte de reconocimiento.


rec

Sistema de Reconocimiento de Patrones y Clasificación
Programa de Reconocimiento de Patrones (REC)
versión 2.20, 23 de Noviembre de 1987

Es reinicio de alguna sesión interrumpida (s/n)? n

Inicializando los datos...

Cúal es el nombre del archivo
con la matriz de aprendizaje? matriz.ma

Número de rasgos:          10
Número de objetos:         15
Número de clases:           3

Calculando los testores mínimos...

Candidatos:             0
Testores:               0
Permanecen:             0


Quedan por revisar (en binario): 1111111111
Candidatos:           100
Testores:              38
Permanecen:             9


Quedan por revisar (en binario): 1000111011
Candidatos:           200
Testores:              69
Permanecen:            11


Quedan por revisar (en binario): 0001011110

Subconjuntos revisados:       224
Testores encontrados:          74
Testores mínimos:               9

Almacenando los testores mínimos en "matriz.tm"...
Almacenando los pesos diferenciantes en "matriz.pd"...
Almacenando los ideales de clase en "matriz.pd"...

C:\TMP>
```

```
type matriz.ma
15
10
   1   1   1   1   1   1   1   1   1   1
   1   1   0   0   1   0   0   0   1   0      1
   0   1   1   2   1   0   2   0   1   1      1
   2   1   1   0   2   0   1   0   1   1      1
   0   1   1   0   1   0   1   2   0   0      1
   0   1   1   0   1   0   1   0   0   0      1
   1   0   0   1   0   0   1   0   0   1      2
   1   0   0   1   2   1   2   1   0   1      2
   1   1   0   1   0   1   1   2   0   1      2
   0   0   0   1   0   1   1   1   0   1      2
   1   0   0   1   0   0   1   0   0   1      2
   0   2   1   1   0   1   0   0   1   1      3
   0   0   0   1   0   1   1   1   1   1      3
   1   2   1   1   1   1   1   0   1   1      3
   0   0   1   2   0   1   0   2   0   0      3
   0   0   1   0   0   1   0   0   0   1      3

A:\P>
A:\P>type matriz.tm
9
10
    1    3    4    5    9    0
    1    4    5    7    9    0
    1    4    5    9   10    0
    1    6    7    9    0
    1    6    8    9   10    0
    2    6    8    9   10    0
    3    6    9    0
    4    6    9   10    0
    6    7    8    9   10    0

A:\P>
A:\P>type matriz.pd
10
                1   5.5555555555E-01
                1   1.1111111111E-01
                1   2.2222222222E-01
                1   4.4444444444E-01
                1   3.3333333333E-01
                1   6.6666666667E-01
                1   3.3333333333E-01
                1   3.3333333333E-01
                1   1.0000000000E+00
                1   5.5555555555E-01


3
    1  3.1111111111E+00
    0    1    1    0    1    0    1    0    0    0
    2  3.6888888889E+00
    1    0    0    1    0    0    1    0    0    1
    1    0    0    1    0    0    1    0    0    1
    1  3.2666666667E+00
    0  255    1    1    0    1    0    0    1    1

A:\P>
```

```
ejcla

C:\TMP>echo off
Ejemplo de ejecución del Sistema de Recocimiento de Patrones
y de Clasificación (RECLA), en su parte de clasificación.


cla

Sistema de Reconocimiento de Patrones y Clasificación
Programa para la Clasificación de Objetos (CLA)
versión 1.00, 1 de Diciembre de 1987

Inicializando variables...

Número de rasgos: 10
Número de clases: 3

Cargando los ideales de cada clase...

Total de ideales: 4

Cúal es el nombre del archivo
 con los objetos por clasificar? objetos.ob

Clasificando...
Número de objetos: 6

La clasificación de los objetos se guardó en el archivo "c:objetos.cl".

C:\TMP>
```

```
type objetos.ob
6
10
  1  1  1  1  1  1  1  1  1  1
  0  1  2  1  1  2  2  0  0  1
  1  1  0  0  1  0  1  0  1  1
  1  1  0  1  0  2  1  1  1  0
  0  0  0  1  2  2  2  0  0  1
  0  0  0  0  0  1  2  0  1  1
  0  1  2  2  1  1  1  0  0  1

A:\P>
A:\P>type objetos.cl
     6    10
     3
  6.6666666667E-01   6.6666666667E-01   1.0000000000E+00

  1.0000000000E+00   8.7500000000E-01   1.0000000000E+00

  1.0000000000E+00   1.0000000000E+00   6.6666666667E-01

  6.6666666667E-01   1.0000000000E+00   8.3333333333E-01

  6.6666666666E-01   1.0000000000E+00   8.8888888889E-01

  5.0000000000E-01   7.5000000000E-01   1.0000000000E+00

A:\P>
```