13
24

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

A C A T L A N

SELECCION DE VARIABLES CON EL ALGORITMO REC

T E S I S

QUE PARA OBTENER EL GRADO DE

LICENCIADA EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A

JIMENEZ JACINTO VERONICA

MARZO 1995

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

A mi madre
Por la toda la confianza y el apoyo brindado
Por tener siempre su aplomo de mujer de verdad

A mi Padre
En Memoria
Luchador incansable
Hombre Honesto
Quien nos enseño que la palabra Patria
guarda mucho más que 6 letras

A mis Hermanos y Hermanas
Una lista enorme
mas que en número de elementos
en experiencias, motivaciones y apoyos
Los quiero un ch...orro

Agradezco:

A Shul
por venir a planificarnos la vida.
por su apoyo y su amistad.

Y a el conjunto A, donde

$A=\{x \cup z \mid x$ es toda aquella persona que de una u otra manera permitió el feliz término de este trabajo; z es toda aquella Institución que me ha permitido aprender y formarme, bien o mal, en ella$\}$

Tratar de enumerar todos los elementos de este conjunto sería un trabajo fallido.

INDICE

INTRODUCCIÓN

El reconocimiento de patrones es un atributo de cualquier organismo vivo, incluyendo al hombre. El proceso de reconocer patrones hecho por el hombre, surge con él, cuando éste aún habitaba las cavernas, y debía reconocer entre las plantas que podía comer y las que no por ser venenosas; a sus depredadores y a sus cohabitantes pacíficos.

El proceso de reconocer un patrón puede dividirse en dos grandes vertientes. En la primera, el reconocimiento involucra alguno de nuestros sentidos para captar las características del patrón a reconocer, por ejemplo el hecho de identificar y diferenciar una manzana, un coche o un perro. En la segunda, se involucra el reconocimiento de conceptos más abstractos como al realizar una demostración matemática.

La complejidad del reconocimiento de patrones se ha incrementado en la medida de la cantidad de información que ha tenido que manejar el hombre. "De hecho el grado de Desarrollo de una sociedad puede ser medido por la cantidad de información y conocimiento que genera y puede manejar" [1].

En las sociedades antigüas, los procesos llevados a cabo de forma cotidiana en la comunidad eran de todos conocidos, y cualquiera podía explicarlos y comprenderlos. A medida que el conocimiento ha crecido, el número de procesos se ha incrementado así como su grado de complejidad . Esto empieza a desenbocar en una mayor dificultad de comprender todos los procesos que se llevan a cabo de manera diaria dentro de un mismo grupo social. A su vez provoca un mayor interés en el manejo de los datos y en métodos más efectivos para convertir esos datos en información. Lo importante ahora es poder extraer nueva información de los

datos, extraer información que no era evidente, que no se puede percibir fácilmente.

Dentro del campo de la computación uno de los máximos retos es poder hacer que una máquina sea capaz de percibir su medio tal como lo hace un ser humano, esto involucra tratar de almacenar y recuperar la información como lo hace nuestro cerebro . Los logros aún son incipientes.

El reconocimiento de patrones es una disciplina que permite llevar a cabo este tipo de procesos al recabar información y analizarla para llevar a cabo clasificación, diagnóstico y predicciones.

Su aplicación ha producido excelentes resultados tanto en el procesamiento de imágenes como en los diagnósticos médicos, en la localización de zonas petroleras, análisis de electrocardiogramas, clasificación de frutas, y una enorme lista más de aplicaciones.

Para llevar a cabo la modelación de un sistema de Reconocimiento de Patrones (R.P.), una de las partes más importantes es la selección de las variables que describan los objetos a clasificar. Una mala selección puede llevar a resultados erróneos, a tener un número demasiado grande de rasgos que pueden resultar ser engorrosos para su manejo, incluso ser redundantes. Esta selección de variables ó rasgos permite entre otras cosas, la comprensión de la información a partir de los valores de las variables, así como determinar el conjunto de rasgos que permiten una descripción diferencial de un conjunto de objetos.

La selección de las variables es una etapa intermedia entre

el espacio real y el espacio de clasificación. Es la una transformación original a uno mas pequeño pero que guarda las mismas propiedades que el espacio original.

Existen varios métodos matemáticos para elegir el conjunto de variables "ideal" para describir al modelo. Un conjunto de estos métodos son los que se basan en la Teoría de Testores, apoyandose en ella se han creado una serie de algoritmos, entre los que se encuentra el algoritmo REC, que es el tema principal de este trabajo. El algoritmo REC fué propuesto por Rafael Morales Gamboa para obtener el título de Licenciado en Matemáticas en la Facultad de Ciencias de la Universidad Nacional Autónoma de México, en 1988. El objetivo de la presente tésis es el de mejorar este algoritmo.

En los dos primeros capítulos, se hará una breve introducción al planteamiento formal de problemas dentro del R.P. y la teoría de testores. En el tercer capítulo se describe el algoritmo tal como fué propuesto por Morales Gamboa. En los tres últimos capítulos se desarrollan los mejoramientos.

CAPITULO I
¿ QUE ES EL RECONOCIMIENTO DE PATRONES?.

I.1 ANTECEDENTES

Formalmente el orígen de la Teoría de Reconocimiento de Patrones hecho de manera automática se sitúa a los inicios de 1950, cuando las computadoras empiezan a demostrar su poder en el manejo de información y se llevan a cabo lo primeros intentos por reconocer de manera automática. Estos primeros intentos por "reconocer" o "percibir" patrones se basaron en la teoría de la decisión y en los umbrales de la lógica.

A finales de 1950 [2], Rosenblatt propone un algoritmo llamado Perceptron, que intentó ser un modelo que almacenara y organizara la información de manera similar a como lo hace el cerebro. Su artículo fue publicado en una revista de psicología en 1958, bajo el titulo " The perceptron: a probabilistic model for information storage and organization in the brain ". Este artículo inicia con tres cuestionamientos ¿Como es sensada o detectada la información del mundo físico por los sistemas biológicos? ¿En qué forma es almacenada o recordada esa información? ¿Como hace la información contenida en memoria para influir en el reconocimiento y comportamiento?. La primera pregunta, desde la óptica de Rosenblatt estaba ya bastante contestada. Las otras dos cuestionantes son la guía del resto del artículo y reciben respuesta bajo afirmaciones tales como que "El sistema nervioso central actúa como una intrincada red de conmutación, donde la retención toma la forma de nuevas conecciones o caminos entre centros de actividad...El desarrollo de la lógica simbólica, las computadoras digitales y la teoría del diseño digital han originado muchas teorías con la similitud computacional entre una neurona y una simple unidad de *on-off* de

las cuales las computadoras están construidas y han provisto métodos analíticos necesarios para representar funciones lógicas altamente complejas en términos de estos elementos." [2]. El escrito se ocupa entonces de describir un algoritmo que trata de simular el comportamiento de las neuronas, y ser capaz de este modo de percibir para clasificar o reconocer patrones.

Existen por otro lado intersecciones entre el reconocimiento de patrones y otras áreas de aplicación como la inteligencia artificial y el procesamiento de imágenes.

I.2 IMPORTANCIA DEL R.P.

Los conceptos y el interés en el Reconocimiento de Patrones se han incrementado rápidamente en las últimas decadas, debido a los buenos resultados en sistemas para el manejo de la infomación dentro de estudios interdisciplinarios o en investigaciones dentro de varios campos como la ingeniería, las ciencias de la computación, sicología, biología, fisiología, medicina, etc.

Uno de los problemas de los que se ocupa el R.P. es el de clasificación. La clasificación es la habilidad de poder definir la pertenencia de un elemento a un conjunto o clase.

Conjunto, elemento y pertenencia son los tres axiomas de la Teoría de Conjuntos. Los conjuntos no se definen, se determinan. Existen dos formas de determinar un conjunto. Una por extensión y otra por intención.

Cuando se determina un conjunto por extensión se reune a todos los elementos que constituyen este conjunto, es decir, se enlista a todos sus miembros y es inmediato pensar que este tipo de conjuntos son finitos y su cardinal no es muy grande. Por

ejemplo {lunes, martes, miércoles, jueves, viernes, sábado, domingo}. Cuando se determina por intención se describen las características que deben cumplir sus elementos para pertenecer al conjunto. Por ejemplo {x| x es un día de la semana}(Que se lee x tal que x es un día de la semana), ó {x| x $\in$ N}(x tal que x pertenece a N).

Vistas así las cosas, cuando se trata de determinar si un objeto pertenece o no a un conjunto lo único que debemos hacer es verificar si se encuentra en la lista de elementos, en el primer caso, o si cumple con las condiciones impuestas, para el segundo caso. El problema surge cuando la lista de elementos no es completa, o cuando no se pueden determinar todas las características que permitirían decidir qué objetos pertenencen al conjunto y cuáles no. También puede ocurrir que existe la propiedad que describe a los objetos que deben pertenecer al conjunto, pero esta propiedad es poco precisa o ambigüa, o requiere de conocimientos amplios en una aréa determinada, por ejemplo {x|x es una persona honesta}, {x|x es un fruto no muy grande}, {x|x tiene bronquitis}, {x|x es una zona perspectiva para el petróleo}. Y este tipo de situaciones se presentan de manera común en los problemas de clasificación reales, sobre todo en las ciencias como la Medicina, Biología, Sociología, Psicología, Geología, etc.. Es entonces cuando el R.P. adquiere su importancia dado que ofrece una solución a problemas como los descritos en estos párrafos.

El Reconocimiento de Patrones se aplica en aquellas ciencias donde el conocimiento no se presenta siempre de manera, es decir, donde ciertas consideraciones están en función directa del subjetivismo del experto. Donde las conclusiones a las que se llega al solucionar ciertos problemas no tiene una explicación única, no es universal y en mucha ocasiones ni siquiera existe.

Donde no siempre se conocen expresiones analíticas que describan la interacción entre los diferentes factores que influyen en el comportamiento del fenómeno. Cuando no se tiene idea precisa del grado de influencia de los distintos hechos que inciden en el problema.

Por ejemplo, el diagnóstico de un enfermo acerca de si tiene X ó Y enfermedad, no puede formularse a partir de una regla establecida, fija, ni universal y depende del criterio del médico que lo atienda. Otro ejemplo se presenta si se debe decidir cuándo un jovencito de 10 años tiene un perfil que indique si es un criminal potencial, aqui no se sigue un patrón fijo, pues la determinación esta basada en el subjetivismo del criminalísta que analice el problema.

La teoría de Reconocimiento de Patrones ha desempeñado un papel de suma importancia en la resolución de este tipo de problemas de clasificación dentro de estas ciencias.

El proceso de R.P. tiene que ver con los problemas de clasificación, de diagnóstico, y de pronóstico. Por ejemplo, del diagnóstico del estado mental de un paciente dentro de la Psicología; del diagnóstico de falla de un equipo mecánico; del pronóstico de la ocurrencia de un movimiento telúrico o cualquier otro fenómeno natural; la clasificación de horizontes geológicos en yacimientos de petróleo o de algún tipo de mineral; la clasificación de medidores de luz dentro de 100 clases distintas de fallas; la determinación de precios de predios, la evaluación de la calidad en la producción de una fábrica; la lectura diagnóstica de señales biomédicas como electrocardiogramas, electroencefalogramas y otros; el pronóstico de erupciones de violencia en un grupo social; etc.

La teoría de conjuntos, la teoría de lenguajes formales y la teoría de optimización sobre las que descansaba el R.P. originalmente, es el punto de partida para los diferentes enfoques de la aplicación de esta disciplina. Entre los cuales se encuentran desarrollos cimentados en :

- las estadísticas
- la probabilidad
- la teoría de conjuntos difusos
- las funciones potenciales
- la lógica clásica
- la lógica difusa
- la lógica polivalente
- la lingüística matemática
- la combinatoria
- la teoría de gráfos
- las ecuaciones diferenciales
- etc.

En función de las disciplinas en que descansa un desarrollo dentro del reconocimineto de patrones se pueden distinguir 4 grandes enfoques:

- R.P. estadístico probabilístico
- R.P. sintáctico estructural
- R.P. con redes neuronales
- R.P. lógico combinatorio

Cada uno de estos enfoques tiene sus propias restricciones, sus ventajas y desventajas. El decidir cuál de ellos (o sus combinaciones) es el más adecuado esta en función directa del tipo de problema que tengamos en frente, y de qué características cumplen sus variables. Lo anterior, no por obvio, deja de ser

importante y pilar del tratamiento que en lo sucesivo se dará en esta tesis.

Se dice que un problema de R.P. es con aprendizaje si se cuenta con una muestra para cada una de las clases en que se divide el universo de objetos; con aprendizaje parcial si se carece de muestra para una de las clases; y sin aprendizaje si no se sabe a que clase pertenencen los objetos de la muestra.

I.3 CONCEPTOS BÁSICOS.

Desde el punto de vista lógico combinatorio un problema de Reconocimiento de Patrones con aprendizaje se describe como sigue:

Se tiene un universo M el cual está dividido en clases no necesariemente disjuntas y se cuenta con una muestra de objetos para cada clase, los objetos son descritos en términos de un conjunto de rasgos, variables ó características (Fig 1). El problema consiste en determinar a qué clase pertenece un nuevo objeto. En los siguientes párrafos formalizamos estos conceptos.

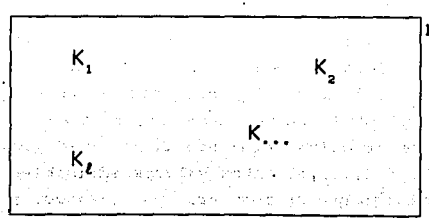


fig. 1. Un universo cualquiera M, dividido en ℓ clases, de cada una de las cuales poseemos una muestra de objetos.

Se considera un conjunto de objetos denotados por $O_1,...O_n$,

divididos en ℓ clases denominadas por $K_1,\ldots K_\ell$. Este conjunto de objetos estan descritos por un conjunto de rasgos $x_1,\ldots x_n$, cada una de estas descripciones se llama descripción estandar del objeto O_i, y se denota $I(O_i)=(x_1(O_i),\ldots,x_n(O_i))$ y cuya definición formal se da en la definición 1.1 que se encuentra más adelante. Esta información la podemos representar por una matriz llamada Matriz de Aprendizaje (M.A.). El objetivo es poder decidir a qué clase pertence un nuevo objeto, que está descrito por medio de estas características, es decir por su n-uplo informacional.(Fig. 2)

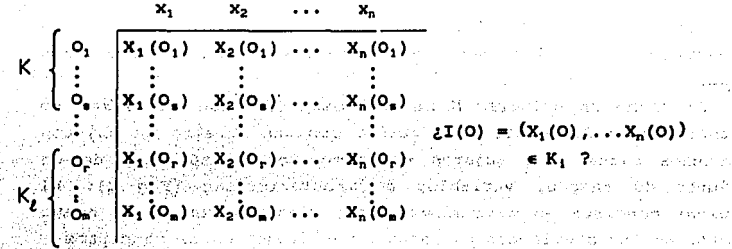


fig. 2. MATRIZ DE APRENDIZAJE EN EL PLANTEAMIENTO DE R.P.

Como se dijo antes, los objetos $O_1,\ldots,O_s$ no son todos los que conforman la clase K_1, sino los objetos de los cuales se tiene información en la clase K_1, esta situación se repite para las ℓ clases. Por otra parte, cada rasgo tiene asociado un conjunto M_i que se denomina "conjunto de valores admisibles del rasgo x_i", donde $i = 1,\ldots,n$. Estos valores admisibles pueden ser lo sencillo o complejo que se requiera, por ejemplo, pueden tomar valores dentro del conjunto $\{0,1\}$ ó estar en intervalos $[a,b]$, $[a,b)$, (a,b) , con a y b números cualesquiera, incluso $(-\infty,\infty)$; pueden tomar valores dentro de un conjunto $\{a_1,\ldots a_s\}$ donde a_i no necesariamente es un valor numérico, aunque debe permitir la

comparación entre ellos. También se puede considerar la ausencia de información que es denotada con el símbolo " * ".

DEFINICION 1.1: Una descripción estandar de un objeto O será un n-uplo $I(O) = (x_1(O),\ldots,x_n(O))$ donde $x_i(O) \in M_i$, $i = 1,\ldots n$, es el valor del rasgo x_i para el objeto O, M_i es el conjunto de valores admisibles de x_i. Si $x_i(O) \neq *$ para $i = 1,\ldots n$, diremos que I(O) es una descripción completa de O en términos de $x_1,\ldots x_n$.[3]

DEFINICION 1.2: Llamaremos l-uplo de pertenencia de O a $\bar{\alpha}(O) = (\alpha_1(O),\ldots,\alpha_\ell(O))$, donde $\alpha_i(O) =$ "$O \in K_i$" y se cumple que $\alpha_i(O) \in \{0,1,*\}$. $\alpha_i(O) = 0$ significará que $O \notin K_i$, $\alpha_i(O) = 1$, que $O \in K_i$ y $\alpha_i(O) = *$, que no se conoce si pertenece o no el objeto en cuestión a la clase K_i. Si $\alpha(O) \neq *$, $\forall\ i = 1,\ldots n$, lo llamaremos ℓ-uplo de pertenencia completo y si se cumple que $\alpha_i(O) = P_i(O)$, siendo $P_i(O)$ el predicado que describe correctamente la pertenencia de O a K_i, $\bar{\alpha}(O)$ se denominará ℓ-uplo de pertencia correcto. Esta última denominación obedece al hecho de que estos procesos son heurísticos y como veremos más adelante los algoritmos de reconocimiento producen sus ℓ-uplos de pertenencia. Finalmente, denominaremos ℓ-uplo de pertenencia verdadero a $\bar{\alpha}(O)$ si éste es completo y correcto.

DEFINCION 1.3: Por información estandar de las clases $K_1,\ldots,K_\ell$ entenderemos

$$I_o(K_1,\ldots,K_\ell) = (I(O_1), \bar{\alpha}(O_1),\ldots,I(O_m), \bar{\alpha}(O_m))$$

donde $I(O_i)$ es la descripción de O_i y $\bar{\alpha}(O)$ su ℓ-uplo de pertenencia. $I_o(K_1,\ldots,K_\ell)$ será una información estandar correcta (completa, verdadera), en dependencia de que todos sus ℓ-uplos de pertencia sean correctos(completos, verdaderos).

En lo sucesivo supondremos que estamos trabajando con

información estandar verdadera de las clases $K_1,\ldots,K_\ell$.

Denominaremos criterio de comparación δ_i a la función que nos permite comparar dos valores distintos del mismo rasgo. Esta δ_i puede ser cualitativa(booleano o k-valente) o cuantitativa, en dependencia de el conjunto de valores admisibles M_i de la variable x_i, (fig 3). En el caso de los δ_i booleanos, estos denotan la "semejanza" o no del par de valores. Los δ_i k-valentes sirven para dar una gradación de la "semejanza" entre los dos valores y los δ_i aritméticos proporcionan una magnitud de la coincidencia de los valores de x_i. Es por demás decir que el tipo de los criterios de comparación está en dependencia del problema específico que se tenga enfrente.

valores de x_i admisibles		nominal		ordinal	aritmética
δ_i		boolena	k-valente		
$c_{u_{a_{1_i}}}$	booleano	X	X	X	X
$t_{a_{t_{i_{v_o}}}}$	k-valente		X	X	X
cuantitativo					X

FIG. 3. VALORES DE LOS CRITERIOS DE COMPARACIÓN.

Algunos ejemplos de criterios de comparación considerando $X_i(O_h) \in M_i$, $X_i(O_j) \in M_i$, son los siguientes:

$$1)\,\delta_i(X_i(O_h),X_i(O_j)) = \begin{cases} 0 \text{ si } X_i(O_h)=X_i(O_j) \cdot \text{v } X_i(O_h)=* \text{ v } X_i(O_j)=* \\ 1 \text{ si } X_i(O_h) \neq X_i(O_j) \end{cases}$$

Este es el criterio de comparación más común llamado de igualdad.

$$2)\delta_i(X_i(O_h),X_i(O_j))=\begin{cases}0 \text{ si } (X_i(O_h)-X_i(O_j))\leq c \text{ v } X_i(O_h)=* \text{ v } X_i(O_j)=* \\ 1 \text{ en otro caso.}\end{cases}$$

En este criterio de comparación se definen dos valores como semejantes siempre y cuando su diferencia sea menor que un valor predefinido c. Por ejemplo, sea un rasgo llamado "calificación" en la cual los valores admisibles están en el intervalo [0,10] de valores reales, definimos que un par de calificaciones cuya diferencia es menor a 5 centésimas son semejantes; de tal forma que $X_i(O_h) = 9.12$ y $X_i(O_j) = 9.15$ son dos calificaciones semejantes.

$$3)\text{Sea } M_i = \bigcup_{t=1}^{s} [a_t,\ a_{t+1}),\ a_j \in M_i,\ j = 1,\ldots,s+1$$

$$\delta_i(X_i(O_h),X_i(O_j))=\begin{cases}0 \text{ si } (X_i(O_h) \in [a_p,a_{p+1}) \wedge (X_i(O_j) \in [a_p,a_{p+1}) \\ \qquad\qquad \text{v } X_i(O_h) = * \text{ v } X_i(O_j)=* \\ 1 \text{ en otro caso.}\end{cases}$$

En este criterio de comparación dos valores son semejantes si estan dentro del mismo intervalo, sea x_i "temperatura" de los pacientes y $M_i = [35.8,\ 37.0) \cup [37.0,38.2)$ entonces si $X_i(O_h)=36.5$ y $X_i(O_j) = 36.0$, se puede afirmar que son semejantes porque ambos pertenecen al mismo intervalo [35.8,37.0).

4)Sea M_i como en el ejemplo 3).

$$\delta_i(X_i(O_h),X_i(O_j))=\begin{cases}0 \text{ si } ((X_i(O_h) \in [a_p,a_{p+1}) \wedge X_i(O_j) \in [a_p,a_{p+1}) \\ \quad \wedge |X_i(O_h)-X_i(O_j)|)\leq c_i \text{ v } X_i(O_h)=* \text{ v } X_i(O_j)=* \\ 1 \text{ en otro caso.}\end{cases}$$

En este criterio para que dos valores sean semejantes, deben,

además de estar en el mismo intervalo, tener entre ellos una diferencia menor o igual a c_1.

A partir de estos criterios de comparación, se formulan las funciones de semejanza que permiten cotejar cada par de objetos, son una medida de la semejanza entre las descripciones de 2 objetos. Un ejemplo de una función de semejanza es el siguiente:

$$\underset{\sim}{\delta} = \begin{cases} 1 & \text{si para todo } i = 1,\ldots n,\ \delta_i(X_i(O_h), X_i(O_j)) = 0. \\ 0 & \text{en otro caso.} \end{cases}$$

Planteamiento formal del R.P.

Supongamos un conjunto de objetos M descritos por $x_1,\ldots,x_n$ rasgos que toman valores admisibles en $M_1,\ldots,M_n$ respectivamente, y también que se han definido criterios de comparación para cada uno de estos rasgos $\delta_1,\ldots,\delta_n$. Consideremos también un cubrimiento finito de subconjuntos propios de M, denotados por $K_1',\ldots,K_\ell'$. Una matriz o muestra de aprendizaje, será una información estandar verdadera $I_o(K_1,\ldots,K_\ell)$, donde $K_i \subset K_i'$, con $i=1,..\ell$, y una matriz o muestra de control estará conformada por un conjunto de descripciones estandar, posiblemente completas $I(O_1'),\ldots,I(O_q')$ de un conjunto de objetos admisibles $O_1',\ldots,O_q'$.

El problema consiste en hallar un algoritmo A tal que :

$$A(I_o(K_1,\ldots,K_\ell),\ I(O_1'),\ldots,I(O_q') = \|\alpha_j^i(O_i')\|_{q\times\ell}$$

siendo $\alpha_j^i(O_i) \in \{0,1,*\}$ la respuesta del algoritmo A en cuanto a la pertenecia de O_i' a la clase K_j,por medio de una matriz $\|a_{ij}\|_{q\times\ell}$ que denota una matriz de q filas y ℓ columnas, donde * denota la abstención del algoritmo A a clasificar O_i' [4].

Cabe hacer algunas observaciones a este planteamiento. El algoritmo A puede equivocarse, es decir, $\alpha_j^i(O_i^l) \neq P_j(O_i^l)$ donde $P_j(O_i^l)$ es la potencia real del objeto O_i. En otras palabras, los ℓ-uplos de pertenencia que produce A para cada objeto O_i a clasificar pudieran no ser completos e incluso no ser correctos. Las clases pueden ser disjuntas ($K_i \cap K_j = \emptyset$, para $i \neq j$) o tener intersección ($K_i \cap K_j \neq \emptyset$, para $i \neq j$), es decir, pueden existir elementos multiclasificados. Estas clases pueden ser difusas, dicho de otro modo, cada objeto puede tener asociado un grado de pertenencia a las clases. Pueden no existir clases, o carecer de información de los objetos de alguna de ellas. Los criterios de comparación entre los valores admisibles de los rasgos pueden ser booleanos, finito-valentes, infinito-valentes, ó difusos. Y los propios rasgos pueden ser boolenos, k-valentes, ∞-valentes ó lingüísticos fig(3).

I.4 EL PROBLEMA DE LA SELECCIÓN DE VARIABLES

El espacio de representación es el producto cartesiano de los valores admisibles para cada variable, $M_1 \times M_2 \times \ldots M_n$.

Dentro del R.P., el problema de la selección de variables que determinarán el espacio de representación, genera por sí sólo toda una fuente de problemas aún no resueltos completamente.

Básicamente hay dos grandes vertientes: una que intenta reducir el espacio de representación, es decir el número de rasgos en términos de las cuales se describe el problema; y otra que pretende evidenciar la importancia de cada uno de estos rasgos o variables.

En la primera vertiente se trata de optimizar el modelo de representación. Resulta inmediato que es mucho más económico

trabajar con un grupo reducido de rasgos que con enormes cantidades de ellos, que antes que resultar más completos pueden caer en la redundancia, o incluso en una nula aportación de información por parte de ciertos rasgos en particular o combinación de ellos.

Lo ideal es poder describir todos los objetos con el conjunto mínimo de rasgos que nos permita distinguir unívocamente a cada objeto de la clase K_i de las restantes clases, para que la clasificación de los nuevos objetos sea lo más inmediata y exacta posible.

La segunda vertiente versa en el sentido de localizar cuáles variables aportan más información, ya sea para efectos de clasificación, pronóstico o predicción, dicho de otro modo, cuáles son más importantes. Esta "importancia" que se denominará de aquí en lo sucesivo *peso informacional del rasgo*, se puede medir desde distintos enfoques, algunos de los cuales se basan fundamentalmente en el cálculo de los testores típicos, concepto que será definido en el capítulo siguiente, pero que de manera informal se debe pensar en combinaciones de rasgos que diferencian entre todos los objeto de distintas clases, es decir en su capacidad diferenciante.

En ocasiones un rasgo no tiene importancia por si sólo, sino por su combinación con otros rasgos. Estas combinaciones de rasgos pueden ser de suprema utilidad en el diagnóstico médico, por ejemplo, o en la clasificación de zonas geológicas ó en una lista interminable de problemas prácticos.

Es por ello que uno de los objetivos primarios de este trabajo es el análisis de un algoritmo que calcule éstas combinaciones de rasgos y su optimización para hallarlos. Para

ello se requiere formalizar una serie de conceptos en el capítulo siguiente, que están relacionados con estos subconjuntos de características. Luego la descripción del algoritmo en el capítulo III, y su extensión en los capítulos IV, V y VI.

CAPÍTULO II: ¿ QUÉ ES LA TEORÍA DE TESTORES?

II.1 ORÍGENES.

Una de las ramas de la Lógica Matemática que se utilizará como herramienta fundamental en la solución de los problemas de Reconocimiento de Patrones bajo el enfoque lógico combinatorio, és la Teoría de Testores.

La Teoría de Test (como se denominó en sus inicios) se formuló como una de las direcciones científicas independientes de la Cibernética Matemática a mediados de los años 60. Su origen en 1954-1955, está vinculado a la utilización de métodos lógicos matemáticos para la localización de desperfectos en los circuitos eléctricos que realizan funciones boolenas. Las primeras investigaciones en esta línea fueron llevadas a cabo por los especialistas soviéticos S. V. Yablonskii, I.A. Cheguis y Yu I. Zhuravliov[4]. En los trabajos de los primeros especialistas se formula por primera vez el concepto de TEST, algunas derivaciones de ese concepto, algunas propiedades y se describe el primer algoritmo para el cálculo de todos los test que cumplen una cierta propiedad de optimalidad. Todo ello en el marco de la Lógica Matemática clásica, haciendo uso del aparato de las formas normales disyuntivas boolenas. En estos dos primeros trabajos se dieron soluciones a algunos problemas particulares relacionados con los circuitos lógicos (para esquemas particulares que realizan funciones simétricas elementales, funciones lineales para esquemas de comparación, para un sumador binario.) aunque las formulaciones iniciales están hechas para "esquemas" que realizan funciones booleanas, siendo los circuitos lógicos el caso más simple y conocido de los "esquemas lógicos".

Los trabajos de Cheguis y Yablonskii abrieron una

importante línea de investigacion que pudieramos denominar "Teoría de Test para esquemas lógicos" y que consiste en la elaboración de modelos y métodos matemáticos para la solución de problemas que conllevan al análisis de esquemas conformados por elementos que pueden cambiar su funcionamiento ante el surgimiento de "desperfectos". De la definición de test dada por Yablonskii en términos de las posibles funciones de desperfectos que pudieran ser construidas bajo deteminadas hipótesis, se llega a una matriz (de desperfectos) que posee la peculiaridad de tener sus filas dos a dos diferentes.

En esta línea surgen dos problemas que motivan una cierta subdivisión de la inicial, a saber, la necesidad de estimar las "longitudes" de los test y los aspectos algorítmicos de la cuestión de hallar los test y con ellos procedimientos diagnósticos del estado de los esquemas.

En 1969 se celebró un evento sobre diagnóstico técnico en Moscú donde se expusieron una serie de importantes trabajos en el área de la segunda subdivisión. Posteriormente y en forma anual S.V. Yablonskii realiza un evento que abarca a toda la ex-URSS sobre los problemas relacionados con el desarrollo y las aplicaciones de la Teoría de Testores en el que participan especialistas de otros países ex-socialistas principalmente.

II.2 CONCEPTOS BÁSICOS DE TESTOR.

En este subcapítulo se verán algunas de las definiciones que se han dado de testor.

El concepto de test fué cambiado por el de testor, para evitar que se piense en el concepto en términos de una "prueba", y con esto se hagan otras interpretaciones, aunque esta fue la

idea original de Chegüis y Yablonskii, por lo que exceptuando la primera definición, se hablará de aquí en adelante de testor.

·Sea M una conjunto de funciones $\{f \mid f:E^n \longrightarrow G\}$, E y G, conjuntos dados y E^n denota el producto cartesiano n veces de E. Sea $N = \{(f,g) \mid f,g \in M, f \neq g\} \subseteq M^2$. Que los elementos de N sean diferentes dos a dos, quiere decir que para cada par de funciones de M existe al menos un vector $\bar{\alpha} \in E^n$ sobre el cual las funciones del par toman valores diferentes en G.

DEFINICION 2.1 (Chegüis y Yablonskii).- El conjunto $T \subseteq E^n$ se denomina testor (relativo a E, M, y N) si no existe par alguno (f,g) de funciones de N tal que $f|_T = g|_T$, donde $f|_T$ denota la restricción del dominio de f a T.

Las funciones a las que hace referencia la definición 2.1 son las llamadas "funciones de desperfectos" del esquema lógico estudiado. En cada caso el conjunto de dichas funciones origina lo que ellos también denominaron "tabla de desperfectos"[4].

·Sea M un conjunto finito de objetos sobre los que se definen un conjunto de predicados $x_1,\ldots x_n$, x_{n+1}. Este último se denomina "predicado fundamental" y su función es determinar la pertenencia de un objeto a una de las clases dadas. Es de hecho, un predicado ficticio ya que justamente su formulación es desconocida y en la práctica o se parte de definiciones dadas por el experto o se aplican técnicas de agrupamiento (cluster analysis). Los predicados restantes los denominaremos "rasgos" (o cualesquiera de sus sinónimos). Esto es, dado un conjunto de objetos, estos pueden ser puestos en correpondencia con un conjunto de n-uplos, sus descripciones en términos de los valores de los rasgos, este conjunto puede conformar una tabla (de "objeto-propiedad -TOP-) o matriz cuyas filas pudieran agruparse

sobre la base del predicado fundamental. En pocas palabras, partimos de una información estandar de las clases $K_1,\ldots K_\ell$ (definición 1.3), si el problema tiene definido el predicado fundamental para algunos objetos y partimos de la descripción estandar(definición 1.1) en caso que no se conozcan los ℓ-uplos de pertenencia (defincion 1.2) de los objetos.

Considere T_0 y T_1 como 2 clases disjuntas:

DEFINICION 2.2 (Zhuravliov): El subconjunto $\mathcal{T} = \{i_1,\ldots i_s\}$ de columnas de la tabla T (y sus respectivos rasgos $x_{i_1},\ldots x_{i_s}$) se denomina testor para $(T_0,T_1) = T$, si después de eliminar de T todas las columnas excepto las de $\mathcal{T}$ no existe fila alguna en T_0 igual a una de T_1.

Aún cuando Zhuravliov habla exclusivamente de 2 clases (T_0 y T_1) es fácil observar que la definición es extendible a ℓ clases sin violarla.

Tanto la definición 2.1 como la definición 2.2 pueden ser complementadas con la siguiente definición:

DEFINICION 2.3(J.R. Shulcloper) Un testor T se llama irreducible (típico) si al eliminar cualquiera de dichas columnas deja de ser testor, al cardinal del conjunto T se denominará longitud del testor.

El término irreducible, de carácter eminentemente matemático, habla de obtener la representación más pequeña que nos permite distinguir unívocamente entre cualquier par de renglones de la matriz. Los testores típicos no son necesariamente únicos.

Por ejemplo, considere una matriz de aprendizaje de 3 clases y 4 rasgos:

		x_1	x_2	x_3	x_4
K_1	O_1	0	0	1	0
	O_2	1	1	0	0
K_2	O_3	0	0	0	1
K_3	O_4	1	1	1	1
	O_5	1	0	1	1

esta matriz de aprendizaje, tiene cinco testores, $\{x_1, x_2, x_3, x_4\}$, $\{x_1, x_4\}$, $\{x_2, x_3, x_4\}$, $\{x_3, x_4\}$, de los cuales, tres son testores típicos, $\{x_1, x_3\}$, $\{x_1, x_4\}$, $\{x_3, x_4\}$.

$\{x_1, x_3\}$ es testor típico, porque, como se observa en la siguiente submatriz que se define a partir de este conjunto:

		x_1	x_3
K_1	O_1	0	0
	O_2	1	0
K_2	O_3	0	1
K_3	O_4	1	1
	O_5	1	1

no existe ningún renglón de una clase que sea igual a uno de otra. Con esto el conjunto puede considerarse testor, pero además ocurre que si eliminamos cualquiera de los rasgos que forman el conjunto, este presenta renglones iguales, si eliminamos x_1 el renglón correspondiente al objeto 1 resulta igual a los dos renglones de la clase K_3, y el renglón correspondiente al objeto 2, resulta igual al objeto 3 de la clase K_2. Por esta última característica $\{x_1, x_3\}$ es un testor típico.

Con la submatriz definida por el subconjunto $\{x_1, x_4\}$,

ocurre una situación semejante:

		x_1	x_4
K_1	O_1	0	0
	O_2	1	0
K_2	O_3	0	1
K_3	O_4	1	1
	O_5	1	1

ya que distingue perfectamente entre todos los objetos de clases distintas y no se puede eliminar ninguno de los rasgos sin obtener renglones indistinguibles de clases distintas en la submatriz.

Por otra parte, $\{x_1, x_2, x_3, x_4\}$, es testor pero no típico, porque se pueden eliminar rasgos, por ejemplo x_1 y x_2, y seguir distinguiendo entre todos los elementos de clases distintas.

El término típico, va más acorde con el proceso de modelación matemática e indudablemente es mejor para el trabajo interdisciplinario con los especialistas no matemáticos.

Sea

$$P_t(O_i,O_j)=\begin{cases}0 \text{ si } x_t(O_i) = x_t(O_j)\\ 1 \text{ en otro caso}\end{cases} \qquad \text{para } t = 1,\ldots,n \qquad \ldots(1)$$

donde el predicado $P_t(O_i,O_j)$ es igual a 0 si las respectivas coordenadas de O_i y O_j son iguales, cuando el valor es 1, entonces son diferentes.

Este predicado o su negación es extensamente usado en todos los trabajos dedicados al tema, permite realizar la comparación dos a dos entre todas las descripciones estandar de la muestra, que pertenecen a diferentes clases, obteniendose una nueva matriz a la que se llamará Matriz de Diferencias (M.D.) .

DEFINICION 2.4(Aizenberg y Tsipkin).- Un conjunto testor de predicados $\tilde{P}_{i_1}(O_l,O_j),\ldots,\tilde{P}_{i_\ell}(O_l,O_j)$ $(i_1 < \ldots < i_\ell)$ para la muestra de aprendizaje $\{A_1,\ldots,A_r\}$ es un conjunto tal que

$\forall\, O_l \in A_p\ \forall\, O_j \in A_q\ ((p \neq q) \rightarrow \overline{\tilde{P}_{i_1}(O_l,O_j),\ldots,\tilde{P}_{i_\ell}(O_l,O_j)})$ ...

es una tautología.

El símbolo "~" encima de los predicados significa que pueden aparacer con negación. La barra sobre del predicado que aparece a continuación del símbolo implicación denota la negación del mismo. En otras palabras, el conjunto de predicados es testor si y sólo si para cualesquiera dos vectores O_l y O_j de clases diferentes de la muestra de aprendizaje, existe un i_t, $t \in \{1,\ldots\ell\}$ tal que $x_t(O_l) \neq x_t(O_j)$ si $P_{i_t}(O_l,O_j)$ aparece sin negación y la igualdad en caso contrario.

Un testor en el sentido de Zhuravliov (definición 2.3) coincide con los conjuntos testores en los que no aparecen predicados con negaciones, enuncian Aizenberg y Tsinpkin.

Un conjunto testor es denominado primo por Aizenberg y Tsipkin si ningún subconjunto propio es testor. La conjunción correspondiente la denominan testor primo.

Es por demás decir que estas son sólo algunas de las diferentes extensioneš que existen de testor y testor típico. Pero para los objetivos de esta tesis resultan suficientes para

entender la esencia de los testores y la utilidad y sentido de los testores típicos. Si se desea conocer las distintas extensiones de testor desde diferentes enfoques se puede encontrar un tratamiento amplio de ellas en [5].

Cabe mencionar que la Teoría de Testores ha encontrado una barrera importante en la complejidad de los algoritmos que hasta el momento se emplean para hallarlos.

I.3 VINCULACIÓN CON EL R.P.

La primera vinculación que hubo entre el Reconocimiento de Patrones y la incipiente en aquél momento teoría de test según [3], ocurre cuando Zhuravliov lleva el concepto de test a un terreno como el de la Geología, publicando sus resultados en 1966.

Si se ve hacia los orígenes de la Teoría de Testores es evitente que aún cuando la intención de Chegüis y Yablonskii no era dar solución a problemas de Reconocimiento de Patrones, sus primeros trabajos de algún modo lo dan.

Luego, en 1966 Zhuravliov aplica el concepto de testor en la solución de un problema de Geología, con lo que se da la primera aproximación formal de este concepto y se presentan los primeros algoritmos para el cálculo de todos los testores típicos de una matriz. Este es un nuevo enfoque desde la matemática discreta dado por Dmitriev, Zhuravliov y Krendeleiev. Y de aquí en adelante la Teoría de Testores abre toda una rama de investigación en la que hasta el momento, un gran número de investigadores trabaja, de modo tal que soluciona problemas de Reconocimiento de Patrones dentro de ciencias como la Geología y la Medicina, entre otras.

CAPÍTULO III.
EL ALGORITMO REC

III.1 CONCEPTOS BÁSICOS.

Consideremos $\mathcal{R}=\{\alpha_1,\alpha_2,\ldots,\alpha_n\}$, el conjunto de todos los rasgos donde por convención diremos que i < j. Calcular los testores típicos puede ser equivalente a generar todos los subconjuntos posibles de $\mathcal{R}$, lo que sería computacionalmente irrealizable en la medida de que el conjunto crece.

Hay dos formas generales de calcular todos los testores típicos:

a) construyendo la matriz identidad, basándose sobre la estructura interna de la matriz de diferencias(definición 2.4) y persiguiendo reducirla a una matriz identidad. Los algoritmos que trabajan bajo esta forma se llaman de escala interior.

b) auxiliandose de un n-uplo booleano $\bar{\omega} = (\omega_1,\ldots,\omega_n)$, tal que si $w_j = 0$ significa que la columna correspondiente al rasgo j, no aparece en el conjunto de columnas a considerar. Todas las combinaciones que puede tener este n-uplo booleano se pueden obtener a través de el conjunto potencia de $\mathcal{R}$, y estas combinaciones pueden ser recorridas en distinto orden en dependencia del algoritmo en turno. El algoritmo también define cómo "saltar" entre estos n-uplos $\bar{\omega}$. Los algortimos que trabajan de este manera se denominan de escala exterior.

El algoritmo REC es un algoritmo de escala exterior, por lo tanto iniciemos definiendo un orden para los subconjuntos posibles de $\mathcal{R}$ y una función de recorrido

$$\rho : \mathcal{P}\{\alpha_1,\ldots,\alpha_n\} \longrightarrow \mathcal{P}\{\alpha_1,\ldots,\alpha_n\}$$

tal que ρ^n denota la n-ésima composición de la función y ρ^o la identidad.

$\Phi = \{X \subset \{\alpha_1,\ldots,\alpha_n\} \mid X = \rho^n(\{\alpha_1,\ldots,\alpha_n\}),\ n \in \mathbb{N}\}$
y TM=$\{X \subset \{\alpha_1,\ldots,\alpha_n\} \mid X$ es un testor$\}$ entonces $TM \subseteq \Phi$.

Se requiere ahora establecer un orden sobre $\mathcal{P}(\{\alpha_1,\ldots,\alpha_n\})$. Sea $X = \{\alpha_{i_1},\ldots,\alpha_{i_s}\} \subseteq \{\alpha_1,\ldots,\alpha_n\} = \mathcal{R}$.

Denotamos

$X = [\alpha_{i_1}|X']$ con $X' = \{\alpha_{i_2},\ldots,\alpha_{i_s}\}$. Además $[] = \varnothing$.

donde α_{i_1} denota el primer elemento de X y X' el resto del conjunto X (al estilo de prolog).

DEFINICION 3.1(Morales Gamboa): Sea $\succ\cdot$ una relación en $\mathcal{P}(\mathcal{R})$ tal que

a) $X \succ\cdot \varnothing$ para cualquiera que sea $X \in \mathcal{P}(\mathcal{R})$;

b) si $X = [\alpha|X']$, $Y = [y|Y']$ y $\alpha < y$, recordando que $\alpha_i < \alpha_j$, $\forall\ i < j$, entonces $X \succ\cdot Y$
es decir, si el primer elemento del conjunto X es anterior que el primer elemento del conjunto Y entonces $X \succ\cdot Y$;

c) si $X = [\alpha|X']$, $Y = [\alpha|Y']$ y $X' \succ\cdot Y'$ entonces $X \succ\cdot Y$, es decir, si el primer elemento de X es igual al primer elemento de Y, entonces de manera recursiva hay que analizar el resto de X y el resto de Y. En el momento que se pueda decidir si alguno de los restos es anterior que el otro se puede decir si X es anterior que Y.

Esta relación, coloca cada superconjunto antes que sus

subconjuntos. Se desprende entonces la siguiente proposición:

PROPOSICION 3.1(Morales Gamboa): Si $Y \subset X$, entonces $X \succ\cdot Y$.

DEMOSTRACION: Se realizará por inducción sobre el número de elementos de X. Si $X = \emptyset$, entonces $Y = \emptyset$ y la relación se cumple en función de a). Suponga ahora que el enunciado de la proposición es válido para cualquier X con menos de s elementos y se quiere probar que se cumple para cuando X tiene s elementos:

1) si $Y = \emptyset$, entonces $X \succ\cdot Y$ en virtud de a).
2) si $y_{i_1} \neq \alpha_{i_1}$, entonces $\alpha_{i_1} < y_{i_1}$ ya que $Y = \{y_{i_1},\ldots,y_{i_t}\} \subset X = \{\alpha_{i_1},\ldots,\alpha_{i_s}\}$ y en este caso la relación se cumple en virtud de b);
3) Si $y_{i_1} = \alpha_{i_1}$, entonces tiene que ocurrir que $Y' \subset X'$, que como se dijo antes se define como $X = [\alpha_{i_1}|X']$ y $Y = [y_{i_1}|Y']$, y por la hipótesis de inducción y el inciso c) se completa la demostración. ∎

PROPOSICION 3.2(Morales Gamboa): Si $X,Y, \in \mathcal{P}(\mathcal{R})$, entonces $X \succ\cdot Y$ ó $Y \succ\cdot X$.

DEMOSTRACION:Sean $X,Y \in \mathcal{P}(\mathcal{R})$
1) Si $X = \emptyset$, entonces $Y \succ\cdot \emptyset = X$ y análogamente $X \succ\cdot Y$ si $Y = \emptyset$.
2) Si $X = [\alpha_{i_1}|X']$, $Y = [y_{i_1}|Y']$ y $\alpha_{i_1} < \alpha_{i_1}$, entonces $X \succ Y$ y de la misma manera $y_{i_1} < \alpha_{i_1}$, ocurrirá que $Y \succ X$.
3) Si $\alpha_{i_1} = y_{i_1}$, entonces se analizará de manera recursiva a X' y Y', en lugar de X, y Y respectivamnete, con lo que termina la demostración. ∎

PROPOSICION 3.3(Morales Gamboa): $\succ\cdot$ es un orden total en $\mathcal{P}(\mathcal{R})$.

DEMOSTRACION: Primero probaremos que la relación es reflexiva, antisimétrica y transitiva.

Es reflexiva: $\emptyset \succ\cdot \emptyset$ en virtud de a) de la definición 3.1, obviamente siguiendo un proceso inductivo y basándose en el hecho de que $\mathcal{R}$ está ordenado totalmente $X = [\alpha|X'] \succ\cdot [\alpha|X'] = X$.
Es antisimétrica: Sean $X,Y \in \mathcal{P}(\mathcal{R})$ tales que $X \succ\cdot Y$ y $Y \succ\cdot X$. Si $X = \emptyset$ entonces por a) de la definición 3.1 se tiene que $Y=\emptyset$. Si $X \neq \emptyset$, entonces se aplica la condición b) de la mencionada definición, obteniéndose en primera instancia que $\alpha = y$ porque $\mathcal{R}$ esta ordenado totalmente; $X = [\alpha|X']$, $Y = [y|Y']$; y además que $X' \succ\cdot Y'$ y $Y' \succ\cdot X'$ en virtud de c). Es claro que siguiendo este mismo proceso inductivamente para X' y Y'; X'',Y'', etc. se concluye que $X = Y$.

Es transitiva: Sean $X, Y, Z \in \mathcal{P}(\mathcal{R})$ tales que $X \succ\cdot Y$ y $Y \succ\cdot Z$. Si $Z = \emptyset$, es inmediato en virtud de a) que $X \succ\cdot Z$. Si $Y = \emptyset$, entonces $Z = \emptyset$ y también se cumple. Si $X = \emptyset$ entonces $Y = \emptyset$ y por el paso anterior $X \succ\cdot Z$.
Supongamos ahora que $X = [\alpha|X']$, $Y = [y|Y']$, $Z = [z|Z']$.
1) Si $\alpha < y$ y $y \leq z$, entonces $\alpha < z$ en virtud del orden en $\mathcal{R}$ y por b) $X \succ\cdot Z$;
2) Si $\alpha = y$ y $y < z$, análogamente $\alpha < z$ y $X \succ\cdot Z$;
3) Si $\alpha = y = z$, entonces $X' \succ\cdot Y'$ y $Y' \succ\cdot Z'$ y por un procedimiento recursivo se concluye que $X \succ\cdot Z$ lo que completa la demostración de que $\succ\cdot$ es un orden parcial, pero si agregamos lo demostado en la proposición 3.2, como se indica en [12] podemos concluir que existe un orden total en $\mathcal{P}(\mathcal{R})$ ∎

PROPOSICION 3.4(Morales Gamboa)(CARACTERIZACION DE $\succ\cdot$): $X \succ\cdot Y$ si y sólo si $X = Y$ ó existe $x_0 \in X$ tal que i) $x_0 \notin Y$ y ii) $\forall y \in Y$ con $y < x_0$ se tiene que $y \in X$.

DEMOSTRACION: Suponga que $X \succ\!\!\cdot\ Y$ y $X \neq Y$ (que en lo sucesivo denotaremos $X \succ Y$).

1) Si $Y = \emptyset$, por la suposición anterior $X \neq \emptyset$ y la proposición se cumple obviamente.

2) Si $X = [x|X']$, $Y = [y|Y']$ y $x < y$, entonces $x \notin Y$ y ii) se cumple de inmediato.

3) Si en las condiciones anteriores $x = y$, entonces $X' \succ Y'$ y el proceso en 2) repite cuantas veces sea necesario y se obtiene el cumplimiento de i) y ii).

Supongamos ahora que existe $x_o \in X$ tal que se cumplen las condiciones i) y ii). Si $Y = \emptyset$, por a) de la definición 3.1 $X \succ Y$. Se supone pues que $y = [A|B]$ donde $A = \{y \in Y | y < x_o\}$, $B = \{y \in Y | x_o < y\}$, recordando que en virtud de i) $x_o \notin Y$. Entonces puede ocurrir:

1) $A = \emptyset$, en este caso por ii) el primer elemento de y_{i_1} de Y cumplirá $y_{i_1} > x_o \geq x_{i_1}$, siendo este último el primer elemento de X y en virtud de b) en la definición 3.1 se tiene $X \succ Y$.

2) $A \neq \emptyset$, en este caso por ii) el primer elemento y_{i_1} de Y cumple que $y_{i_1} \in X$. Si $x_{i_1} < y_{i_1}$ por b) de la definición 3.1 queda $X \succ Y$. Si $x_{i_1} = y_{i_1}$ entonces como $A \neq \emptyset$ existe t tal que $x_{i_t} < y_{i_t}$ y aplicando b) de la definición 3.1 se tendría que $X^{t-1} \succ Y^{t-1}$, siendo

$$X = [x_{i_1}, x_{i_1}, \ldots, x_{i_{t-1}} | X^{t-1}] \text{ y } Y = [y_{i_1}, \ldots, y_{i_{t-1}} | Y^{t-1}]$$

de donde iterando t-1 veces con la condición c) de la mencionada definición se obtendría $X \succ Y$. ∎

Con las definiciones anteriores se ha establecido un orden, que como se corroborará, es el orden antinatural, entendido "antinatural" como el orden inverso del orden natural que guardan por ejemplo los números enteros, y que se visualiza fácilmente si consideramos a cada conjunto en correspondencia con el n-uplo

característico $\bar{\omega}$, que contenga un 1 en el lugar n-ésimo significa que estamos considerando al n-esimo rasgo. Por ejemplo, si tuviéramos 5 rasgos $X = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$, el conjunto potencia de este conjunto esta formado por 2^n subconjuntos, es decir $2^5 = 32$.

La lista de n-uplos característicos o números binarios sería la sucesión del 31 al 0, si se listan en orden antinatural se obtendría lo que se muestra abajo con su conjunto equivalente de rasgos a la derecha:

decimal	binario	conjunto correspondiente
31	1 1 1 1 1	$\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$
30	1 1 1 1 0	$\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$
29	1 1 1 0 1	$\{\alpha_1, \alpha_2, \alpha_3, \alpha_5\}$
28	1 1 1 0 0	$\{\alpha_1, \alpha_2, \alpha_3\}$
27	1 1 0 1 1	$\{\alpha_1, \alpha_2, \alpha_4, \alpha_5\}$
26	1 1 0 1 0	$\{\alpha_1, \alpha_2, \alpha_4\}$
25	1 1 0 0 1	$\{\alpha_1, \alpha_2, \alpha_5\}$
24	1 1 0 0 0	$\{\alpha_1, \alpha_2\}$
23	1 0 1 1 1	$\{\alpha_1, \alpha_3, \alpha_4, \alpha_5\}$
22	1 0 1 1 0	$\{\alpha_1, \alpha_3, \alpha_4\}$
21	1 0 1 0 1	$\{\alpha_1, \alpha_3, \alpha_5\}$
20	1 0 1 0 0	$\{\alpha_1, \alpha_3\}$
19	1 0 0 1 1	$\{\alpha_1, \alpha_4, \alpha_5\}$
18	1 0 0 1 0	$\{\alpha_1, \alpha_4\}$
17	1 0 0 0 1	$\{\alpha_1, \alpha_5\}$
16	1 0 0 0 0	$\{\alpha_1\}$
15	0 1 1 1 1	$\{\alpha_2, \alpha_3, \alpha_4, \alpha_5\}$
14	0 1 1 1 0	$\{\alpha_2, \alpha_3, \alpha_4\}$
13	0 1 1 0 1	$\{\alpha_2, \alpha_3, \alpha_5\}$
12	0 1 1 0 0	$\{\alpha_2, \alpha_3\}$
11	0 1 0 1 1	$\{\alpha_2, \alpha_4, \alpha_5\}$

10	0 1 0 1 0	$\{\alpha_2, \alpha_4\}$
9	0 1 0 0 1	$\{\alpha_2, \alpha_5\}$
8	0 1 0 0 0	$\{\alpha_2\}$
7	0 0 1 1 1	$\{\alpha_3, \alpha_4, \alpha_5\}$
6	0 0 1 1 0	$\{\alpha_3, \alpha_4\}$
5	0 0 1 0 1	$\{\alpha_3, \alpha_5\}$
4	0 0 1 0 0	$\{\alpha_3\}$
3	0 0 0 1 1	$\{\alpha_4, \alpha_5\}$
2	0 0 0 1 0	$\{\alpha_4\}$
1	0 0 0 0 1	$\{\alpha_5\}$
0	0 0 0 0 0	$\varnothing$

Para cualquier par de conjuntos se cumplen las definiciones descritas antes, por ejemplo, para averiguar si el conjunto correspondiente al 29 antecede al conjunto correspondiente al 27, es decir $\{\alpha_1, \alpha_2, \alpha_3, \alpha_5\} \succ \{\alpha_1, \alpha_2, \alpha_4, \alpha_5\}$ hay que aplicar el inciso c) de la definición 3.1, como el primer elemento de ambos conjutos es igual analizamos el resto, $\{\alpha_2, \alpha_3, \alpha_5\}$ y $\{\alpha_2, \alpha_4, \alpha_5\}$, pero también es igual el primer elemento así que se sigue analizando el resto, $\{\alpha_3, \alpha_5\}$ y $\{\alpha_4, \alpha_5\}$, ya en este caso, por el inciso b) de la definición 3.1 dado que el primer elemento del primer conjunto es menor que el primer elemento del segundo conjunto, podemos afirmar que el primer conjunto antecede al segundo.

Por el inciso a) de la definición 3.1, el conjunto que se corresponde con el 0 es el $\varnothing$, ya que es el último de todos los conjuntos pues es posterior a cualquiera de ellos.

La proposición 3.1, nos dice que cualquier subconjunto va después de su superconjunto; esto se ve claro en la lista, ningún subconjunto se encuentra antes de su superconjunto.

Tal como lo expresa la proposición 3.2, para todo par de conjuntos se puede decidir quién antecede a quién, es decir, no hay un sólo par para el cual no sea posible definir un orden.

Efectivamente, es un orden total como versa la proposición 3.3 y al referirnos a la proposición 3.4 podemos observar que para todo par de conjuntos (denotados por X y Y), ocurre que o son iguales o existe un elemento α_0 que pertenece a X y no pertenece a Y, y para todos los elementos de Y que son menores que α_0 pertenecen a X. Tómense por ejemplo los conjuntos correspondientes a los números 23 y 22, denotándolos

$$X = \{\alpha_1, \alpha_3, \alpha_4, \alpha_5\} \text{ y } Y = \{\alpha_1, \alpha_3, \alpha_5\}$$

en este caso $\alpha_0 = \alpha_4$ que no pertenece a Y, y todos los elementos de Y menores que α_0, es decir α_1 y α_3, están contenidos en X.

Ahora que se tiene claro el orden definido para todas las combinaciones de los rasgos que de alguna manera establecen una "dirección de búsqueda", vamos a ver de qué forma se localizan estos testores típicos, que como se definió en el capítulo anterior nos permiten distinguir entre las descripciones (o subdescripciones) de los objetos que se encuentran en clases distintas.

Se debe tener claro que estos subespacios de representación logran distinguir entre todos los pares de objetos que no pertenencen a la misma clase, si esto no ocurre quiere decir que confunden las descripciones de un cierto número de pares de objetos al considerar subconjuntos de rasgos. Cada vez que se localiza una subdescripción igual de dos objetos distintos, se dirá que ese subconjunto de rasgos está cometiendo un error que puede ser cuantificable y que ayudará a acercarse a los subconjuntos de rasgos que no cometen errores, que no confunden objetos y que por lo tanto cometen un error igual a cero.

Estos conjuntos asociados a un error igual a cero son testores y pueden entonces ser candidatos a ser testores típicos, que distinguen perfectamente cualquier par de descripciones de objetos en clases distintas, sin cometer ningún testor como subconjunto.

Formalicemos este concepto de error y la caracterización de testores desde el punto de vista del algoritmo REC:

DEFINICION 3.2: Sea $X \subseteq \mathcal{R} = \{\alpha_1,\ldots,\alpha_n\}$. Denotemos por E(X) al conjunto de todas las parejas de descripciones estandar semejantes (I(O),I(O')) con respecto a X, de objetos que pertenecen a clases diferentes, es decir:

$E(X) = \{(I(O),I(O') \mid I(O) \overset{x}{\sim} I(O'),\ I(O) \in K_i,\ I(O') \in K_j,\ i<j\}$

donde $\overset{x}{\sim}$ se lee como " son semejantes para el subconjunto X de rasgos".

Se dirá que el error de X es igual al cardinal del conjunto E(X). Teniendo ya definido el concepto de "error de un subconjunto de rasgos" se puede formalmente afirmar que :

PROPOSICION 3.5: X es un testor si y sólo si el error de X es igual a cero.

DEMOSTRACION: Es inmediata a partir de la definición 2.2 ■

Es fácil imaginar que si un subconjunto de rasgos comete cierto error, al aumentar el número de rasgos en ese subconjunto pueden ocurrir dos cosas: la primera es que el error disminuya debido a que el nuevo rasgo permite distinguir mejor entre los objetos, o dicho de otro modo, ayuda a confundir menos objetos, a

disminuir el error que el subconjunto cometía sin él; la segunda es que el error, no disminuya, que se puede interpretar como que la nueva variable introducida al subconjunto no aporta información adicional a la que ya se tenía, entonces no sirve de nada introducirla y daría lo mismo no hacerlo. En realidad sería mejor no introducirla, porque aunque el error se mantiene, el espacio de representación aumenta.

Podemos deducir entonces que para $X \subset Y$, $\|E(X)\| \geq \|E(Y)\|$. Definamos una función

$$\begin{array}{ccc} e_x : X & \longrightarrow & N \\ \alpha_{i_s} & & \|E(X_s)\| \end{array}$$

tal que podamos asociar un valor entero que denota el error que comete el subconjunto X hasta el rasgo α_{i_s}, denotado por X_{i_s} y que cumple que

$$e_{X_i}(x) = e_X(x) \quad \forall x \in X_i \qquad \ldots (1)$$

y si $\alpha_{i_j} < \alpha_{i_k}$, entonces $e_x(\alpha_{i_j}) \geq e_x(\alpha_{i_k})$. $\qquad \ldots (2)$

Entonces, cuando $e_X(\alpha_{i_j}) = e_X(\alpha_{i_{j+1}})$ se puede interpretar como que el rasgo $\alpha_{i_{j+1}}$ no aporta nada al subconjunto de rasgos, es decir, no ayuda a distinguir entre los objetos que se confunden, estando en clases distintas. Se puede concluir, tomando en cuenta la definición de testor típico lo siguiente:

PROPOSICION 3.6: Si X es un testor típico entonces e_x es inyectiva.

DEMOSTRACION: Sea $X = \{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_s}\}$ un testor típico y supongamos que existen dos rasgos $\alpha_{i_j} < \alpha_{i_k}$ tales que

$$e_X(\alpha_{i_j}) = e_X(\alpha_{i_k}).$$

Entonces, en virtud de la proposición 3.5 y de (2) se cumple la siguiente relación:

$$e_x(\alpha_{i_1}) > e_x(\alpha_{i_2}) > \ldots > e_x(\alpha_{i_j}) = e_x(\alpha_{i_{j+1}}) = \ldots = e_x(\alpha_{i_k}) > e_x(\alpha_{i_{k+1}}) > \ldots > e_x(\alpha_{i_s})$$

Como ya se razonó anteriormente, los rasgos $\alpha_{i_{j+1}},\ldots,\alpha_{i_k}$ no aportan información alguna y podemos prescindir de ellos, por lo tanto $X\setminus\{\alpha_{i_{j+1}},\ldots,\alpha_{i_k}\}$ es un testor lo que es una contradicción con el hecho de ser X un testor típico. ∎

Esta condición es necesaria, pero no suficiente para caracterizar a los testores típicos. Se podría tener un subconjunto de rasgos cuya función e_x fuese inyectiva y sin embargo ni siquiera fuera testor, como en el caso de la siguiente matriz:

	α_1	α_2	α_3
K_1	1	1	1
	1	0	1
	0	1	0
K_2	1	1	1
	0	0	0
	0	1	1

$e_X(\alpha_1) = 4$
$e_X(\alpha_2) = 2$
$e_X(\alpha_3) = 1$

como se ve e_x es inyectiva y sin embargo X no es testor. Entonces para pretender que X pueda ser candidato a testor típico, además de tener una función de error inyectiva debe ser igual a cero al considerar todo el conjunto X.

En este momento ya se definió un orden para el conjunto $\mathcal{P}(\mathcal{R})$ y un criterio para determinar quiénes pueden ser candidatos a testor típico, lo que sigue entonces es definir el modo en que

recorreremos el conjunto potencia, o mejor dicho que formaremos el conjunto Φ, para ello utilizaremos las siguientes funciones:

DEFINICION 3.3(Morales Gamboa):Sea $\rho: \mathcal{P}(\mathcal{R}) \longrightarrow \mathcal{P}(\mathcal{R})$. Entonces

a) si e_x no es inyectiva y X es testor entonces $\rho(X) = \sigma(X)$

b) si e_x es inyectiva y X es un testor entonces $\rho(X) = \beta(X)$

c) en otro caso $\rho(X) = \alpha(X)$, es decir, X no es testor.

donde:

1) $\sigma: \mathcal{P}(\mathcal{R}) \longrightarrow \mathcal{P}(\mathcal{R})$, tal que si $X = \{\alpha_{i_1},\ldots,\alpha_{i_s}\}$, entonces

$\sigma(X) = \{\alpha_i \in X \mid \forall\ \alpha \in X,\ \alpha < \alpha_i$ si entonces $e_x(\alpha) > e_x(\alpha_i)\}$

y

$\sigma(\varnothing) = \varnothing$.

2) $\beta: \mathcal{P}(\mathcal{R}) \longrightarrow \mathcal{P}(\mathcal{R})$ tal que si $X = \{\alpha_{i_1},\ldots,\alpha_{i_s}\}$, entonces

$\beta(X) = \{\alpha_{i_1},\ldots,\alpha_{i_{s-1}}\} \cup \{r \in \mathcal{R} \mid r > \alpha_{i_s}\}$ y

$\beta(\varnothing) = \varnothing$.

3) $\alpha: \mathcal{P}(\mathcal{R}) \longrightarrow \mathcal{P}(\mathcal{R})$ tal que si $X = \{\alpha_{i_1},\ldots,\alpha_{i_s}\}$, entonces

$$\alpha(X)=\begin{cases} \varnothing \text{ si } r < \alpha_{i_1} \ \forall\ r \in \mathcal{R}\backslash X \\ \{\alpha_{i_1},\ldots,\alpha_{i_{j-1}}\} \cup \{r \in \mathcal{R} \mid r > \alpha_{i_j}\} \\ \quad \text{si } \alpha_{i_j} = \max\{\alpha \in X \mid \exists\ r \in \mathcal{R}\backslash X, \text{ con } \alpha < r\} \end{cases}$$

¿Qué está haciendo exactamente cada función?. Para el inciso a), se trata de quitar los "bancos" o partes no inyectivas de la función, es decir, los rasgos que no ayudan a disminuir el error cometido por la función lo que los hace dignos de ser eliminados, este trabajo precisamente lo lleva a cabo la función σ, eliminando todas las $\alpha_{i_{j+1}},\ldots\alpha_{i_k}$ tales que

$$e_x(\alpha_{i_j}) = e(\alpha_{i_{j+1}}) = \ldots = e(\alpha_{i_k}) \text{ con } j < k.$$

Para el inciso b), se encontró un conjunto candidato a ser testor típico, entonces se quita su último elemento y a cambio de él se agregan todos aquellos elementos que sean mayores que ese último elemento en el conjunto original $\mathcal{R}$. Esto es un backtracking en realidad. Si retomamos la representación binaria de $\bar{\omega}$ esta función hace saltar siempre a al siguiente n-uplo binario inmediato, ya que al quitar el último elemento y agregar todos los mayores a él es equivalente a un decremento unitario.

Para el último inciso el conjunto X resultó no ser testor, entonces es necesario agregar más rasgos al conjunto o saltar por sus subconjuntos que podrían empeorar la situación. Esto lo hace α localizando el elemento en X más a la derecha que es menor que los elementos que pertenecen a la diferencia de los conjuntos $\mathcal{R}$ y X.

Analicemos ahora algunas de las propiedades de estas funciones:

PROPOSICION 3.7(Morales Gamboa):Sea $X \subset \mathcal{R}$ y E(X) como en la definición 3.2. Entonces

i) $E(\sigma(X)) = E(X)$ y

ii) $e_{\sigma(X)}(\alpha) = e_x(\alpha)$ para cualquier $\alpha \in \sigma(X)$.

DEMOSTRACION: Dado $X \subset \mathcal{R}$, e_x puede o no ser inyectiva. Si e_x es una función inyectiva sobre X significa que $e_x(\alpha_i) > e_x(\alpha_{i+1})$ para cualquier i y por tanto $\sigma(X) = X$, con lo que no habría nada que demostrar. Supongamos que e_x no es inyectiva. Eso significa que existen α_i, $\alpha_j \in X$ tales que $e_x(\alpha_i) = e_x(\alpha_j)$. Supongamos que $\alpha_i < \alpha_j$. Como e_x es una función no creciente se tiene que para toda α tal que $\alpha_i < \alpha < \alpha_j$, $e_X(\alpha) = e_x(\alpha_i)$. En virtud de la definición de la función σ se tendrá que $\sigma(X) = X \setminus \{\alpha_{i+1}, \ldots, \alpha_j\}$.

Es obvio por la definición de E(X) y de e_x que las descripciones que confunde α_i son las mismas que confunden los $\alpha \in \{\alpha_{i+1},\ldots,\alpha_j\}$ por lo tanto $E(\sigma(X)) = E(X)$. Por otra parte, $\forall \alpha_i \in \sigma(X)$ $\|E(\{\alpha_{i_1},\ldots\alpha_{i_s}\})\| = \|E(\sigma(\{\alpha_{i_1},\ldots\alpha_{i_s}\})\|$ de donde se sigue $e_x(\alpha_i) = e_{\sigma(X)}(\alpha_i)$ para los $\alpha_i \in \sigma(X)$. ∎

COROLARIO: $e_{\sigma(X)}$ es una función inyectiva sobre $\sigma(X)$.
La demostración es inmediata a partir de la definición de σ.

PROPOSICION 3.8: Si $X \subset \mathcal{R}$ es un testor, entonces $\sigma(X)$ es también un testor.

DEMOSTRACION: Es inmediata a partir de las proposiciones 3.5 y 3.7, ya que X testor significa $\|E(X)\| = 0$ y como $E(X) = E(\sigma(X))$ el error de $\sigma(X)$ también es cero y por lo tanto testor. ∎

PROPOSICION 3.9: Para todo $X \in \mathcal{P}(\mathcal{R})$ se cumple que

$$X \succ\cdot \sigma(X)$$
$$X \succ\cdot \beta(X)$$
$$\text{y } X \succ\cdot \alpha(X).$$

DEMOSTRACION: El caso $X = \varnothing$ es obvio por definiciones de σ, β, α, por lo que suponemos $X \neq \varnothing$.
A) De la definición de σ (elimina rasgos de X) $\sigma(X) \subseteq X$ y en virtud de la proposición 3.1, $X \succ\cdot \sigma(X)$.
B) Si $X = \{\alpha_{i_1},\ldots,\alpha_{i_s}\}$ por la definición 3.3:

$$\beta(X) = \{\alpha_{i_1},\ldots,\alpha_{i_{s-1}}\} \cup \{r \in \mathcal{R} \mid r > \alpha_{i_s}\}$$

por lo tanto existe un elemento de X que no está en $\beta(X)$, en este caso el α_{i_s} tal que para cualquier elemento $y \in \beta(X)$, $y < \alpha_{i_s}$, $y \in X$, por lo tanto en virtud de la proposición 3.4 se tiene

$$X \succ\cdot \beta(X).$$

C) Por la definición de α (inciso 3 de la definición 3.3) se

sabe que si $r < \alpha_{i_1}$ para cualquier $r \in \mathcal{R}\backslash X$ entonces $\alpha(X) = \varnothing$ y por la definición de orden en $\mathcal{P}(\mathcal{R})$ se tendría $X \rightarrowtail \alpha(X)$. Suponga ahora que $\alpha(X) = \{\alpha_{i_1},\ldots,\alpha_{i_{s-1}}\} \cup \{r \in \mathcal{R} \mid r > \alpha_{i_j}\}$, siendo $\alpha_{i_j} = \max\{\alpha \in X \mid \exists\, r \in \mathcal{R}\backslash X \text{ con } \alpha < r\}$. Queda claro que los primeros (j-1) elementos de X y de $\alpha(X)$ son los mismos y por definición de $\alpha(X)$ el primer elemento del conjunto $\{r \in \mathcal{R} \mid r > \alpha_{i_j}\}$ es mayor que α_{i_j}, por lo tanto por la proposición $X \rightarrowtail \alpha(X)$.■

Con estos últimos resultados podemos afirmar que las funciones de recorrido son monótonas crecientes. Como el objetivo es encontrar todo el conjunto de testores típicos es importante asegurar que se encontrarán todos, cosa que se logra con las siguientes proposiciónes.

PROPOSICION 3.10(Morales Gamboa): Si $X \subset \mathcal{R}$ es tal que $X = \rho^n(\mathcal{R})$, donde $\rho^n(\mathcal{R})$ indica la n-ésima aplicación a partir de $\mathcal{R}$ de ρ con $n \in \mathbb{N}$ y r_o igual al máximo de los elementos de $\mathcal{R}\backslash X$ entonces la función e_x definida sobre X es inyectiva en el subconjunto

$$Z = \{\alpha \in X \mid \alpha < r_o\}$$

DEMOSTRACION:Supongamos que $X = \rho^n(\mathcal{R})$ y $Y = \rho^{n-1}(\mathcal{R})$. Pueden ocurrir las siguientes situaciones: e_y no es inyectiva y Y es un testor; e_y es inyectiva y Y es un testor y finalmente Y no es testor siendo e_y inyectiva en un caso y no inyectiva en el otro.
A) Sea e_Y no inyectiva y Y un testor. Entonces $\rho(Y) = \sigma(Y) = X$ y por la definición 3.3 e_x es inyectiva y como

$$Z = \{\alpha \in X \mid \alpha < \max\{r \in \mathcal{R} \mid r \notin X\}\} \subseteq X$$

e_x es inyectiva sobre Z.
B) Supongamos que e_Y sea inyectiva y Y un testor. En virtud de la citada definición $X = \beta(Y) = \{y_{i_1},\ldots,\alpha_{i_{s-1}}\} \cup \{r \in \mathcal{R} \mid r > \alpha_{i_s}\}$, siendo $Y = \{\alpha_{i_1},\ldots,\alpha_{i_s}\}$. Es claro que $\alpha_{i_s} \notin X$ y es por tanto

$\alpha_{i_s} = \max\{\alpha \in \mathcal{R} | \alpha \notin X\}$. Luego podemos afirmar que $\{\alpha_{i_1}, \ldots, \alpha_{i_{s-1}}\} = Z = \{ \alpha \in X | \alpha \leq \max\{\alpha \in \mathcal{R} | \alpha \notin X\}\}$. Por otra parte e_y es inyectiva luego $e_Y(\alpha_{i_{s-1}})$ también lo es y como en virtud de (1) $e_{Y_{i_{s-1}}}(\alpha) = e_x(\alpha)$ para todo $\alpha \in \{\alpha_{i_1}, \ldots, \alpha_{i_{s-1}}\} = Z$ se cumple lo que se quería probar.

C) Suponga ahora que Y no es testor. Obviamente $Y \neq \mathcal{R}$. Como $\mathcal{R}$ es testor no se puede aplicar $\alpha(\mathcal{R})$ por la definición 3.3. Por tanto tuvo que haberse aplicado al menos una vez $\sigma(\mathcal{R})$ ó $\beta(\mathcal{R})$ antes de obtener a Y no testor. En virtud de lo demostrado en a) y b) existe $Z = \{\alpha \in Y | \alpha < \max\{r \in \mathcal{R} | r \notin Y\}\}$ y $Z \subseteq Y$, tal que e_y es inyectiva en Z. Como Y no es testor $\rho(Y) = \alpha(Y) = X$ y en virtud de la definición 3.3 se tendrá que $\alpha(Y) = \emptyset$ si $r < \alpha_{i_1}$ para todo $r \in \mathcal{R} \backslash Y$, lo que de entrada no se considerará ya que el enunciado de la proposición se refiere a subconjuntos de $\mathcal{R}$, o bien ocurre que $X = \alpha(Y) = \{\alpha_{i_1}, \ldots, \alpha_{i_{s-1}}\} \cup \{r \in \mathcal{R} | r > \alpha_{i_j}\}$ donde $\alpha_{i_j} = \max\{\alpha \in Y | \exists\, r \in \mathcal{R} \backslash Y \text{ con } \alpha < r\}$. En este caso se tiene por la propia definición de $\alpha(Y)$ que $\alpha_{i_j} \notin X$ y obviamente $\alpha_{i_j} < \max\{\alpha \in \mathcal{R} | \alpha \notin Y\}$ por lo que $\alpha_{i_j} \in Z$ de donde se puede concluir que $Z' = \{\alpha \in Y | \alpha < \alpha_{i_j}\} = \{\alpha \in X | \alpha < \alpha_{i_j}\} \subset Z$ y como e_y es inyectiva en Z se tendrá que $e_y(\alpha) = e_x(\alpha)$ para todo $\alpha \in Z'$, luego e_x es inyectiva sobre Z' como se puede apreciar esta conclusión es independiente de que e_y sea inyectiva o no sobre Y ■.

PROPOSICION 3.11: Si $X \succ Y$ y ambos conjuntos tienen en común sus primeros s elementos a lo que denotaremos $Z = \{\alpha_{i_1}, \ldots, \alpha_{i_s}\}$, entonces para todo W tal que $X \succ W \succ Y$, tiene en común ese mismo conjunto Z.

DEMOSTRACION: Por contradicción: Supongamos que existe un W tal que dentro de sus primeros s elementos cuenta con un w_{i_j} tal que

$w_{i_j} \notin X$ y $w_{i_j} \notin Y$, y es el primero con esta propiedad, o que exista un $\alpha_{i_j} \in X$, $\alpha_{i_j} \in Y$ y $\alpha_{i_j} \notin W$, y nuevamente es el primero que cumple esta propiedad, en ambos casos:

i) Si $w_{i_j} < \alpha_{i_j}$, entonces por la proposición 3.4 (caracterización de $\succ\cdot$) se tiene que $W \succ\cdot X$, lo que contradice nuestra suposición inicial.

ii) Si $w_{i_j} > \alpha_{i_j}$ y como $\alpha_{i_j} = \alpha_{i_j}$(porque estan dentro de los s primeros elementos) se deduce que $w_{i_j} > \alpha_{i_j}$. Pero entonces α_{i_j} es el α_o de la proposición 3.4 y por lo tanto $Y \succ\cdot W$, lo que nuevamente contradice nuestra suposición inicial. Por lo tanto no existe w_{i_j} que no sea igual a su correspondiente α_{i_j} ó y_{i_j}, dentro de los primeros s rasgos. ∎

¿Cómo se puede asegurar que la función ρ no saltará por encima de un testor típico impidiendo de este modo encontrarlos todos?

PROPOSICION 3.12: No existe testor típico alguno $T \subset \mathcal{R}$ tal que $\rho^n(\mathcal{R}) \succ T \succ \rho^{n+1}(\mathcal{R})$, $n \in \mathbb{N}$.

DEMOSTRACION: Denotemos por $X = \rho^n(\mathcal{R})$ y supongamos que no se cumple la proposición, es decir, que existe T testor típico que no está incluido en el recorrido definido por ρ. Pueden ocurrir tres situaciones:

A) e_X no es inyectiva y X es testor. En este caso

$$\rho(X) = \rho^{n+1}(\mathcal{R}) = \sigma(X)$$

Supóngase que $t_o = \min\{t \in T | t \notin \sigma(X)\}$ el cual existe ya que $T \succ \sigma(X)$. Por la definición de t_o se tiene que para cualquier $t \in T$ con $t < t_o$, $t \in \sigma(X)$. Por otra parte, en virtud de la proposición 3.4 (caracterización de $\succ\cdot$) existe $\alpha_o \notin \sigma(X)$, $\alpha_o \in T$ y para cualquier $\alpha \in \sigma(X)$, $\alpha < \alpha_o$ se tiene que $\alpha \in T$. Es obvio que $t_o \leq \alpha_o$, ya que t_o se exige que sea el mínimo de los que

pertenecen a T, $t_o \notin \sigma(X)$ y α_o no necesariamente es el mínimo de los que cumplen ese par de requisitos. De lo anterior se puede concluir que $\{\alpha \in T | \alpha < t_o\} = \{\alpha \in \sigma(X) | \alpha < t_o\}$. Pero como por la definición de σ se desprende que $\sigma(X) \subset X$ (e_x no es inyectiva), entonces $\{\alpha \in T | \alpha < \alpha_o\} \subset X$. Ahora pueden ocurrir dos casos:
i) $t_o \in X$, ii) $t_o \notin X$.
i) Si $t_o \in X$ entonces σ lo eliminó, ya que $t_o \notin \sigma(X)$ y entonces eso quiere decir que existe un $\alpha' \in X$ tal que $e_x(\alpha') = e_x(t_o)$, $\alpha' \in \sigma(X)$, $\alpha' < t_o$ y por lo tanto $\alpha' \in T$ y por (1) $e_T(\alpha) = e_X(\alpha)$ para todo $\alpha < t_o$, de donde concluimos que e_T no es inyectiva lo que contradice que sea testor típico (proposición 3.6).
ii) Si $t_o \notin X$ en virtud de la proposición 3.11, e_x es inyectiva sobre el conjunto $Z = \{\alpha \in X | \alpha < t_o \leq \max \{\alpha \in \mathcal{R} | \alpha \notin X\}\}$, X, T y $\sigma(X)$ coinciden en sus elementos $\alpha < t_o$ y como $t_o \in T$ y $t_o \notin X$, en virtud de la proposición 3.4, $T \succ X$ lo que contradice la hipótesis inicial.

B) Supongamos entonces que e_X es inyectiva y que $X = \{\alpha_{i_1}, \ldots, \alpha_{i_s}\}$ es un testor. En este caso por la definición 3.3, tenemos que $\rho(X) = \beta(X)$, pero entre $\beta(X)$ y X no hay ningún conjunto ya que $\beta(X)$ está siempre inmediatamente despues de X, por lo que no hay la posibilidad de que exista ningún T.

C) Supongamos pues que X no es testor, entonces

$$\alpha(X) = \begin{cases} \varnothing \text{ si } r < \alpha_{i_1} \ \forall\ r \in \mathcal{R} \setminus X \\ \{\alpha_{i_1}, \ldots, \alpha_{i_{j-1}}\} \cup \{r \in \mathcal{R} | r > \alpha_{i_j}\}. \\ \quad \text{si } \alpha_{i_j} = \max \{\alpha \in X | \exists\ r \in \mathcal{R} \setminus X, \text{ con } \alpha < r\} \end{cases}$$

Si $\alpha(X) = \varnothing$ significa que X está formado por los últimos elementos de $\mathcal{R}$ lo que implica que $T \subset X$ ya que si tuviera algún elemento diferente t tendría necesariamente que ser $t < \alpha_{i_1}$ y en

este caso por la definición de $\succ\!\cdot$ tendríamos $T \succ\!\cdot X$ en contradicción con la hipótesis de esta proposición. En este caso T no puede ser siquiera testor ya que X no lo es.

Si $\alpha(X) \neq \varnothing$ entonces puede ocurrir que $\alpha_{i_j} \notin T$. Si ocurre esto $T \subset \rho(X)$ por lo que $\rho(X) \succ T$ que contradice la hipótesis. Si $\alpha_{i_j} \in T$ entonces por el mismo razonamiento que hicimos en b) tendremos que los primeros (j-1) rasgos de X coinciden con los de T, por lo que:

$$X = \{\alpha_{i_1}, \ldots \alpha_{i_j}\} \cup \{\alpha_{i_{j+1}}, \ldots, \alpha_n\}$$

$$T = \{\alpha_{i_1}, \ldots \alpha_{i_j}\} \cup \{t_{j+1}, \ldots, t_m\}$$

pero por la definición de α_{i_j}, $\{\alpha_{i_{j+1}}, \ldots, \alpha_n\}$ son los últimos elementos de $\mathcal{R}$ luego $\{t_{j+1}, \ldots t_m\} \subset \{\alpha_{i_{j+1}}, \ldots, \alpha_n\}$, de donde se concluye que $T \subset X$, por lo tanto T no puede ser testor típico ya que X no es testor.

Por todo lo anterior la hipótesis de suponer la existencia de un testor típico fuera del recorrido definido por ρ es falsa ■.

COROLARIO: $TM \subset \Phi = \{X \subset \{\alpha_1, \ldots, \alpha_n\} \mid X = \rho^n(\{\alpha_1, \ldots, \alpha_n\}),\ n \in \mathbb{N}\}$

III.2 DESCRIPCIÓN DEL ALGORITMO REC.

Finalmente se describirá de manera formal cómo opera este algoritmo:

PASO 1: Se considera en X al conjunto de todos los rasgos del conjunto $\mathcal{R}$.

PASO 2: Si X fuera igual al conjunto vacío el algoritmo termina su ejecución.

PASO 3. Calculamos el error del conjunto de rasgos X, es decir, $e_x(\alpha_{i_s})$, donde $X = \{\alpha_{i_1}, \ldots, \alpha_{i_s}\}$.

PASO 4. Si $e_x(\alpha_{i_s}) = 0$, es decir, X es testor, entonces se averigüa si la función e_x es inyectiva. En caso de serlo, se salta al paso 7 y de lo contrario se va al paso 6. Si $e_x(\alpha_{i_s}) \neq 0$ continúa.

PASO 5. Se aplica $\alpha(X)$ y se va al paso 2 con un nuevo conjunto X.

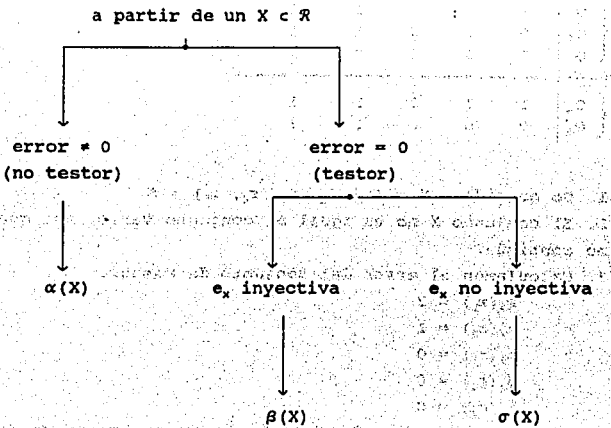
PASO 6. Se calcula $\sigma(X)$ y se va al paso 2.

PASO 7. Se anota el conjunto de rasgos X en la lista de los testores y se eliminan los superconjuntos de dicha lista.

PASO 8. Se aplica $\beta(X)$ y se regresa al paso 2 con un nuevo conjunto X.

Una observación pertinente acerca del algoritmo es el hecho de que la condición de parada dada en el paso 2 es encontrar el conjunto $\varnothing$. Este hecho se deriva de que la función de recorrido visita como última opción al conjunto vacío.

El funcionamiento del algoritmo se puede expresar mediante el siguiente árbol:



III.3 EJEMPLO.

Consideremos la siguiente matriz de aprendizaje con 5 objetos divididos en 2 clases y descritos por 5 rasgos:

		α_1	α_2	α_3	α_4	α_5
K_1	O_1	1	1	0	1	1
	O_2	0	1	0	1	1
	O_3	1	0	1	0	1
K_2	O_4	1	1	1	1	1
	O_5	0	0	0	1	1

1)PASO 1. Se considera $X = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\} = \mathcal{R}$

2)PASO 2. El conjunto X no es igual al conjunto vació, así que el algoritmo continúa.

3)PASO 3. Calculamos el error del conjunto de rasgos:

$$e_X(\alpha_1) = 3$$
$$e_X(\alpha_2) = 1$$
$$e_X(\alpha_3) = 0$$
$$e_X(\alpha_4) = 0$$
$$e_X(\alpha_5) = 0$$

4)PASO 4. Como $e_X(\alpha_{1_s}) = 0$ y la función e_X no es inyectiva se va al paso 6.

5)PASO 6. Se aplica $\rho(X) = \sigma(X) = \{\alpha_1, \alpha_2, \alpha_3\}$. Ahora $X = \{\alpha_1, \alpha_2, \alpha_3\}$.

6)PASO 2. El conjunto X es diferente del $\varnothing$. Se continúa.

7)PASO 3. Se calcula el error:

$$e_X(\alpha_1) = 3$$
$$e_X(\alpha_2) = 1$$
$$e_X(\alpha_3) = 0$$

8)PASO 4. Como $e_X(\alpha_{1_s}) = 0$ y la función es inyectiva se va al

paso 7.

9)PASO 7. Se anota el conjunto X como candidato a testor típico y se eliminan en esa lista los superconjuntos:

LISTA_TESTORES = ($\{\alpha_1, \alpha_2, \alpha_3\}$)

10)PASO 8. Se aplica $\rho(X) = \beta(X) = \{\alpha_1, \alpha_2, \alpha_4, \alpha_5\}$ y se va a 2 haciendo X = $\{\alpha_1, \alpha_2, \alpha_4, \alpha_5\}$.

11)PASO 2. El conjunto X es distinto del conjunto vació, el algoritmo continúa.

12)PASO 3. Se calcula la función de error del conjunto:

$e_X(\alpha_1) = 3$

$e_X(\alpha_2) = 1$

$e_X(\alpha_4) = 1$

$e_X(\alpha_5) = 1$ no es testor

13)PASO 5. Se aplica $\rho(X) = \alpha(X) = \{\alpha_1, \alpha_3, \alpha_4, \alpha_5\}$ y se va a 2 con X = $\{\alpha_1, \alpha_3, \alpha_4, \alpha_5\}$.

14)PASO 2. El conjunto X no es el conjunto vacío. Entonces se continúa.

15)PASO 3. Se calcula error del conjunto:

$e_X(\alpha_1) = 3$

$e_X(\alpha_3) = 2$

$e_X(\alpha_4) = 1$

$e_X(\alpha_5) = 1$ no es testor.

16)PASO 5. Se aplica $\rho(X) = \alpha(X) = \{\alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ y se va a 2 con X = $\{\alpha_2, \alpha_3, \alpha_4, \alpha_5\}$.

17)PASO 2. Se continúa ya que $X \neq \emptyset$.

18)PASO 3. Se calcula la función de error:

$e_X(\alpha_2) = 3$

$e_X(\alpha_3) = 0$

$e_X(\alpha_4) = 0$

$e_X(\alpha_5) = 0$

19)PASO 4. Como la función es igual a cero pero no es inyectiva se va al paso 6.

20)PASO 6.Se aplica $\rho(X) = \sigma(X) = \{\alpha_2, \alpha_3\}$ y se va al paso 2

haciendo $X = \{\alpha_2, \alpha_3\}$.

21)PASO 2. El conjunto X sigue siendo diferente de ø por lo tanto se continúa.

22)PASO 3. Se calcula el error:

$$e_X(\alpha_2) = 3$$
$$e_X(\alpha_3) = 0$$

24)PASO 4. Como el $e_X(\alpha_{1_8}) = 0$ y la función es inyectiva se va al paso 7.

25)PASO 7. Se anota el conjunto X en la lista de testores:

LISTA_TESTORES = $(\{\alpha_1, \alpha_2, \alpha_3\}, \{\alpha_2, \alpha_3\})$

y se eliminan los superconjuntos de la lista:

LISTA_TESTORES = $(\{\alpha_2, \alpha_3\})$

26)PASO 8. Se aplica $\rho(X) = \beta(X) = \{\alpha_2, \alpha_4, \alpha_5\}$ y se va al paso 2 con $X = \{\alpha_2, \alpha_4, \alpha_5\}$.

27)PASO 2. $X \neq \emptyset$. Continúa el algoritmo.

28)PASO 3. Se calcula el error del conjunto X_{1_8}:

$$e_X(\alpha_2) = 3$$
$$e_X(\alpha_4) = 2$$
$$e_X(\alpha_5) = 2 \quad \text{no es testor.}$$

29)PASO 5. Se aplica $\rho(X) = \alpha(X) = \{\alpha_3, \alpha_4, \alpha_5\}$ y se va al paso 2 haciendo $X = \{\alpha_3, \alpha_4, \alpha_5\}$

30)PASO 2. Se continúa con el algoritmo ya que $X \neq \emptyset$.

31)PASO 3. Se calcula la función de error:

$$e_X(\alpha_3) = 3$$
$$e_X(\alpha_4) = 2$$
$$e_X(\alpha_5) = 2 \quad \text{no es testor}$$

32)PASO 5. Se aplica $\rho(X) = \alpha(X) = \emptyset$ ya que $\mathcal{R}\backslash X = \{\alpha_1, \alpha_2\}$ y por lo tanto para todo $\alpha_1 \in X$, y $r \in \mathcal{R}\backslash X$ ocurre que $r < \alpha_1$. Luego se va al paso 2, con $X = \emptyset$.

33) $X = \emptyset$. Entonces el algoritmo termina. la lista de testores contiene en este caso un sólo conjunto $\{\alpha_2, \alpha_3\}$ y este es el único testor típico de la M.A.

En seguida se mostrará la misma corrida del algoritmo evidenciando los saltos a través de los conjuntos de $\mathcal{P}(\mathcal{R})$:

	RASGOS α_1	α_2	α_3	α_4	α_5	ERROR α_1	α_2	α_3	α_4	α_5	e_x inyect.	testor	lista de testores
	1	1	1	1	1	3	1	0	0	0	NO	SI	
α	1	1	1	1	0								
	1	1	1	0	1								
β	1	1	1	0	0	3	1	0	X	X	SI	SI	$\{\alpha_1,\alpha_2,\alpha_3\}$
	1	1	0	1	1	3	1	X	0	0	--	NO	
α	1	1	0	1	0								
	1	1	0	0	0								
	1	0	1	1	1	3	X	2	1	1	--	NO	
	1	0	1	1	0								
	1	0	1	0	1								
	1	0	1	0	0								
α	1	0	0	1	1								
	1	0	0	1	0								
	1	0	0	0	1								
	1	0	0	0	0								
	0	1	1	1	1	X	3	0	0	0	NO	SI	
σ	0	1	1	1	0								
	0	1	1	0	1								
β	0	1	1	0	0	X	3	0	X	X	SI	SI	$\{\alpha_2,\alpha_3\}$
	0	1	0	1	1	X	3	X	2	2	--	NO	
α	0	1	0	1	0								
	0	1	0	0	1								
	0	1	0	0	0								
	0	0	1	1	1	X	X	3	2	2	--	NO	
	0	0	1	1	0								
	0	0	1	0	1								
α	0	0	1	0	0								
	0	0	0	1	1								
	0	0	0	1	0								
	0	0	0	0	1								
	0	0	0	0	0	F I N .							

En este listado resulta evidente la manera de trabajar del algoritmo, en lugar de analizar las 32 posibles combinaciones de rasgos sólo analiza 8. La X indica que el rasgo correspondiente a esa columna no se considera. Cuando encuentra un candidato a testor típico salta al conjunto que sigue inmediatamente, que

seguro es una combinación de rasgos que no está contenida en él. Cuando localiza una combinación de rasgos que no es testor(entonces ni siquiera se analiza si es inyectiva su función de error, lo cual se indica con "--"), salta encima de los inmediatos subconjuntos posibles, ya que si el conjunto analizado no es testor mucho menos lo serán sus subconjuntos. Cuando el conjunto es testor pero su función no es inyectiva elimina los rasgos que no permiten la inyectividad.

III.4 LIMITACIONES.

Para la solución de problemas reales dentro del Reconocimiento de Patrones el algorimto REC presenta algunas limitaciones importantes.

Inicialmente trabaja con la Matriz de Aprendizaje, lo que puede implicar un enorme manejo de información, que puede colocarlo en desventaja desde el punto de vista de complejidad computacional con respecto a otros algoritmos.

Opera sólo para clases no disjuntas, es decir, no pueden existir elementos multiclasificados (descripciones del mismo objeto en distintas clases). Este hecho lo deja fuera en la solución de una gran cantidad de problemas, como en algunos casos del diagnóstico médico, la clasificación de medidores anómalos, etc.

La función de semejanza entre los objetos es demasiado rígida. Los objetos en los problemas reales pueden ser semejantes sin ser iguales. Por ejemplo, en la medicina es semejante tener 37 grados de temperatura que tener 37.5, aún cuando los valores de manera estricta no son iguales. En otros casos un par de

objetos pueden discrepar en el valor de algunas características o rasgos sin dejar de ser semejantes.

Pueden también presentarse objetos que se encuentran en varias clases con distintos grados de pertenencia. Esto requiere hablar de clases difusas, para lo cual el algoritmo REC no está capacitado.

A estas limitaciones se les propone solución de la manera en que se mostrará en los capítulos subsecuentes.

CAPÍTULO IV.
PLANTEAMIENTO PARA CLASES NO DISJUNTAS.

Hasta ahora se ha venido hablando de la matriz de aprendizaje con la restricción de clases disjuntas, $K_i \cap K_j = \emptyset$, para $i \neq j$. Hablar del caso no disjunto nos conduce a cuestionar la validez de que la descripción de un objeto se encuentre simultáneamente en más de una clase.

La causa de tal hecho puede ser producida básicamente por una de dos razones: o los rasgos no son suficientes para distinguir entre dos objetos que no son iguales, pero que bajo ese conjunto de rasgos lo parecen ó efectivamente un objeto pertenece a más de una clase. Este tipo de casos se han presentado en problemas de clasificación dentro de la medicina, donde un mismo objeto (pensemos en un paciente) presenta un cuadro que lo ubica en más de una enfermedad. De hecho estos casos se presentan en casi cualquier problema y no puede pasarse por alto. En general en todos los problemas en los que el predicado que caracteriza a cada clase como conjunto no niega los restantes predicados.

En el primer caso resulta obvio que el problema se pudiera solucionar agregando más rasgos para describir al objeto (cosa que no siempre es posible), o cambiando de rasgos. Para el segundo caso el tratamiento ha sido alguno de los siguientes:

a) Eliminar las descripciones iguales o "repetidas" de algunas de las clases, de manera tal que la descripción del objeto pertenezca exclusivamente a una de ellas. Esto además de

resultar agresivo y de mutilar información, no asegura eliminar la posibilidad de que un nuevo objeto a ser clasificado deba pertenecer a varias clases. Es por demás subjetivo decidir de qué clase se elimina la descripción para que deje de estar repetida. Esta opción es por tanto poco recomendable.

b) Crear una nueva clase con las descripciones que aparecen en más de una clase. Dicho de otra manera, tratar de transformar un problema con clases no disjuntas a uno con más clases disjuntas. Pero nuevamente queda la posibilidad de que un nuevo objeto a ser clasificado deba ser multiclasificado y por otra parte el número de clases puede crecer incontrolablemente, corriéndose el riesgo de no tener una muestra de aprendizaje para cada una de ellas o que dicha muestra no sea representativa.

c) Cambiar la función de semejanza mediante la cual "medimos" la analogía de los objetos, de modo tal que, los objetos que antes resultaban iguales con la nueva función resulten diferentes. Directamente estamos proponiendo en esta variante, modificar la función de semejanza, siempre y cuando no sea booleana. Esta modificación debe responder ante todo a la naturaleza propia del problema, es decir, el cambio debe ser más una respuesta natural al tratamiento del problema que una artimaña matemática que nos salve del embrollo.

d) Conservar las descripciones iguales en clases distintas y entonces trabajar con ellas ampliando nuestras definiciones y conceptos. Replantear pues, nuestro tratamiento del problema. Esta opción resulta la más saludable, y en el caso del algoritmo REC requiere ampliaciones a la definición de testor típico y de la función de Error, las cuales serán tratadas a detalle en este capítulo.

Se debe mencionar que en dependencia del tratamiento que se haga para el caso no disjunto, el conjunto de Testores Típicos varía de manera considerable, de modo tal que una mala elección puede conducirnos a resultados erróneos.

Por ejemplo: considere un par de clases no disjuntas K_1 y K_2

		x_1	x_2	x_3
K_1	O_1	1	1	0
	O_2	0	1	1
K_2	O_3	1	1	0
	O_4	1	0	1

Actuando por a) y eliminado la descripción del objeto 1 obtenemos los siguiente testores típicos $\{x_1\}$, $\{x_2, x_3\}$.

Siguiendo los lineamientos de b) creamos una nueva clase K_0

		x_1	x_2	x_3
K_0	O_1	1	1	0
K_1	O_2	0	1	1
K_2	O_4	1	0	1

obtenemos como testores típicos $\{x_1, x_2\}$, $\{x_1, x_3\}$, $\{x_2, x_3\}$
Con lo que se puede observar el cambio radical en la obtención de los testores típicos.

Planteando que trabajaremos con una matriz de aprendizaje que contiene clases no disjuntas, es decir, $K_i \cap K_j \neq \emptyset$, para $i \neq j$ y asumiendo que actuaremos como lo indica el inciso d), debemos pues ampliar el concepto de Error de la definición 3.2, quedando como sigue:

DEFINICION 4.1 : Denotemos como $ET(\mathcal{R})$, al multiconjunto(es decir

que admite la repetición de sus elementos) de parejas de descripciones estandar semejantes, que pertenecen a clases distintas con respecto al conjunto completo de rasgos, es decir:

$$ET(\mathcal{R}) = \{I(O) \mid I(O) \in K_i \wedge I(O) \in K_j, \text{ para } i<j \},$$

con $i=1,\ldots,\ell-1$, $j = i+1,\ldots\ell$ y $\mathcal{R} = \{x_1,x_2,\ldots,x_n\}$.

E(X) como antes denota al conjunto de subdescripciones semejantes en clases distintas (I(O),I(O')), es decir

$$E(X) = \{I(O),I(O') \mid I(O) \overset{x}{\sim} I(O'),\ I(O) \in K_i,\ I(O') \in K_j,\ i<j\}$$

donde $X=\{x_{i_1},\ldots x_{i_s}\}$ con $s \leq n$.

Diremos que el error de X es igual a la diferencia entre los cardinales de los conjuntos E(X) y ET($\mathcal{R}$).

La definición anterior permite encontrar en la matriz de aprendizaje el conjunto de los objetos multiclasificados y entonces el error considerará sólo aquellas descripciones de objetos que realmente confunden elementos no multiclasificados.

Entiéndase por Conjunto Irreducible de Rasgos (que es la extensión del concepto de Testor Típico para el caso de clases no disjuntas) al subconjunto mínimo de rasgos que no confunden sino exclusivamente a los objetos multiclasificados, conservando la diferenciación entre objetos distintos de clases distintas.

Ejemplo: Consideremos la siguiente matriz de aprendizaje con únicamente 2 clases y 5 rasgos:

		X_1	X_2	X_3	X_4	X_5
K_1	O_1	1	1	0	0	1
	O_2	0	1	0	0	1
	O_3	0	0	0	0	1
	O_4	1	0	0	0	1
K_2	O_1	1	1	0	0	1
	O_5	1	1	1	1	1
	O_6	0	1	1	1	1

El objeto O_1 está clasificado en ambas clases por lo tanto cualquier subconjunto de rasgos cometerá un error por lo menos igual a 1, si se toma como base la definición original de Error. Tomando en cuenta la definición aumentada $ET(\mathcal{R}) = \{I(O_1)\}$, cuya cardinalidad por tanto es 1, la cardinalidad de $E(X)$ ($|E(X)|$) cuando el conjunto $X = \{x_1, x_2, x_3\}$ es también igual a 1. por lo tanto el error $e_x(X_3) = |E(X)| - |ET(\mathcal{R})| = 1 - 1 = 0$, lo que se traduce en que el conjunto X es un Conjunto Irreducible de Rasgos.

Cuando el conjunto $X = \{x_2, x_3, x_4\}$, $|E(X)| = 2$ y por tanto $e_x(X_4) = 2 - 1 = 1$, lo que significa que ese conjunto de rasgos nos confunde además del objeto multiclasificado a un objeto más.

El hecho de contabilizar la diferencia entre las cardinalidades hace trasparentes a los objetos multiclasificados, cuando el algoritmo calcula el error que comete un conjunto de rasgos, de esta forma no tenemos que eliminar dichos objetos de la matriz.

Lo anterior no violenta ninguno de los planteamientos del algoritmo REC, conservando todas las propiedades para las funciones definidas, el recorrido y la caracterización de los

Testores Típicos, que para este caso se llaman de manera apropiada Conjuntos Irreducibles de Rasgos. Esto se concluye al observar que se sigue exigiendo un error igual a cero para caracterizar a los Conjuntos Irreducibles de Rasgos y que para constatar que es el conjunto mínimo, la función σ debe ser inyectiva, en consistencia con la definición dada por Ruiz Shulcloper y Lazo Córtes en [3].

CAPITULO V.
PLANTEAMIENTO DE REC CON LA M.B.

V.1 CONCEPTOS BÁSICOS.

A partir de la matriz de aprendizaje (M.A) se puede definir otra matriz llamada matriz de comparación o matriz de diferencia (M.D.), que explicita la igualdad(0) o diferencia(1), rasgo a rasgo (a partir de un criterio de comparación) entre cada par de objetos de clases distintas. Entonces, si en la matriz de aprendizaje existen m_1 objetos en la primera clase, m_2 objetos en la segunda clase,...,m_l en la l-ésima clase, la matriz de diferencias tendrá:

$m' = m_1(m_2 + \ldots + m_l) + m_2(m_3 + \ldots + m_l) + \ldots + (m_{l-1})m_l$, renglones

Veamos una definición más formal:

DEFINICION 5.1: Sea T_{nml} una matriz de aprendizaje con m objetos descritos con n rasgos y divididos en ℓ clases, y $\bar{\delta} = (\delta_1, \ldots, \delta_n)$ una medida de semejanza entre objetos definida en términos de los criterios de comparación booleanos δ_i, $i = 1, \ldots n$. Llamaremos <u>matriz de diferencias de</u> T_{nml} a la matriz boolena $M.D._{nm'}$ formada por las filas

$$S_{ij} = (\alpha_1^{ij}, \ldots, \alpha_n^{ij}), \text{ con } i \neq j,\ i,j = 1, \ldots, m$$

donde

$$\alpha_p^{ij} = \delta_p\,(\alpha_p^i, \alpha_p^j),\ p = 1, \ldots, n,\quad \alpha_p^i = X_p(O_i),\ \alpha_p^j = X_p(O_j)$$

i.e. el valor que toma el i'esimo y el j'esimo objeto en el rango p.

para $i \neq j$

y $$m' = \sum_{i=1}^{l-1} \sum_{t=1}^{l-i} || K_i || \, || K_{i+t} ||$$

siendo $\| K_i \|$ la cantidad de elementos de $T_{m \times l}$ que pertenencen a la clase K_i, $i = 1,\ldots,\ell$.

Ejemplo: Supóngase una matriz de aprendizaje con n = 4 rasgos, m=5 objetos, y ℓ = 2 clases:

		X_1	X_2	X_3	X_4
K_1	O_1	1	1	0	1
	O_2	1	0	1	0
	O_3	0	1	1	1
K_2	O_4	0	0	1	1
	O_5	1	0	1	1

Supóngase también un criterio de comparación en los siguientes términos:

$$\delta_p(\alpha_p^i,\ \alpha_p^j) = \begin{cases} 1 & \text{si } \alpha_p^i \neq \alpha_p^j \\ 0 & \text{si } \alpha_p^i = \alpha_p^j \end{cases}$$

donde α_p^i denota el valor de la posición (p,i) de la matriz, y α_p^j el valor de la posición (p,j).

Entonces la matriz de Diferencias queda definida como sigue

	X_1	X_2	X_3	X_4
O_{14}	1	1	1	0
O_{15}	0	1	1	0
O_{24}	1	0	0	1
O_{25}	0	0	0	1
O_{34}	0	1	0	0
O_{35}	1	1	0	0

donde el renglón O_{14}, por ejemplo, se interpreta como el resultado de la comparación del objeto 1 y el objeto 4 rasgo a rasgo.

En esta matriz un renglón que consta únicamente de ceros es equivalente a un par de descripciones semejantes en clases distintas.

DEFINICION 5.2: Sea $X \subseteq \mathcal{R} = \{X_1,\ldots,X_n\}$. Definamos el conjunto error sobre los rasgos del conjunto X y denotemos por E(X) al conjunto de renglones tales que $\forall j\ \omega_{ij} = 0$, donde ω_{ij} es el valor en la j-esima columna del i-ésimo renglón de la matriz de diferencias.

De allí que en este ejemplo en particular se puede extraer información rápidamente, como el hecho de que una combinación clave de rasgos para distinguir los objetos de clases distintas es $\{X_2, X_4\}$, dado que si eliminárамos el rasgo 2 el renglón correspondiente a O_{34} quedaría conformado por ceros únicamente, denotando la imposibilidad de distinguir entre el objeto 3 y el objeto 4. Así mismo al eliminar el rasgo 4 provoca que el renglón O_{25} carezca de por lo menos un 1, es decir se elimina el único rasgo que distingue a los objetos 2 y 5.

Se pudo haber definido un criterio de comparación de la manera siguiente:

$$\delta_p(\alpha_p^i,\ \alpha_p^j) = \begin{cases} 1 & \text{si } \alpha_p^i = \alpha_p^j \\ 0 & \text{si } \alpha_p^i \neq \alpha_p^j \end{cases}$$

Entonces un renglón conteniendo sólo 1's denotará un par de objetos de clases distintas indistinguibles, para el subconjunto de rasgos considerado.

A pesar de que la matriz de comparación o diferencias nos permite manifestar la igualdad o diferencia entre todos los pares

de objetos de clases distintas, el tamaño de ésta es en todos los casos muy grande, presenta un crecimiento de orden cuadrático con respecto a la matriz de aprendizaje T_{mal}.

Existe una matriz que sintetiza la información de la matriz de diferencias y que denominaremos Matriz Básica(M.B.). Esta nueva matriz elimina la información redundante en el sentido de que elimina los renglones que son superconjuntos de los demás, si convenimos como que los unos representan diferencia entre un par de objetos para ese rasgo, los unos que aparecen en un mismo renglón pueden ser considerados como una combinación de rasgos que diferencian el par de objetos. En resumen, la matriz básica además de reducir considerablemente las dimensiones de la matriz de diferencias en relación al número de renglones, disminuye la complejidad en relación al número de unos a considerar en cada renglón.

Para formalizar la definición de Matriz Básica requeriremos primero de los siguientes conceptos:

DEFINICION 5.3: Sean i_p, i_t filas de M.D. . Diremos que i_p es una <u>subfila</u> de i_t si y sólo si :

a) $\forall j \ (a_{i_t j} = 0 \Rightarrow a_{i_p j} = 0)$

b) $\exists j_o \ (a_{i_t j_o} = 1 \wedge a_{i_p j_o} = 0)$.

También diremos que i_t es <u>superfila</u> de i_p.

<u>Ejemplo:</u>

Supongamos $i_t = \{ 1 \ 1 \ 1 \ 0 \ 1 \}$ y $i_p = \{ 1 \ 0 \ 0 \ 0 \ 1 \}$, en la cuarta posición de i_t hay un cero, por lo tanto en la cuarta posición de i_p debe encontrarse un cero, tal como sucede. Luego los unos correspondientes al segundo y tercer elemento de i_t, cumplen la existencia de por lo menos un j_o que exige el

inciso b, ya que en esas mismas posiciones i_p tienen ceros. Entonces podemos concluir que i_p es subfila de i_t.

Supongamos ahora que i_t ={1 0 1 0 1} y i_p = {1 1 0 0 1}, en la segunda posición de i_t existe un cero sin que en la segunda posición de i_p exista un cero. Por lo tanto i_p no es subfila de i_t. Por otra parte, queriendo averiguar si i_t es subfila de i_p, observamos que en la tercera posición de i_p hay un cero, sin que en esa misma posición de i_t ocurra lo mismo. Se concluye finalmente que ni i_p es subfila de i_t ni i_t es subfila de i_p.

DEFINICION 5.4(J.R.Shulcoper): Sea i_t una fila de M.D.. La fila i_t es básica si y sólo si en M.D. no existe fila i_p alguna que sea subfila de i_t.

Cabe la observación de que cualquier fila de M.D. es básica o superfila.

Se tienen ahora los elementos necesarios para poder definir formalmente a la Matriz Básica:

DEFINICION 5.5(J.R. Shulcloper): Dada una matriz de comparación M.D. llamaremos matriz básica de M.A. a la matriz M.B. formada exclusivamente por las filas básicas de M.D., excluyendo filas repetidas.

Hasta este punto se ha tomado una matriz de aprendizaje, M.A., de ella hemos extraido una matriz de comparación llamada matriz de diferencias ó M.D., finalmente de ésta llegamos a una matriz básica ó M.B., las tres aportan la misma información desde el punto de vista de la diferenciación de los objetos, pero cada paso ha sido dado en la dirección de una representación más compacta.

V.2 VENTAJAS DE TRABAJAR CON LA M.B. EN LUGAR DE LA M.A.

Para dar una idea de las reducciones que se pueden lograr al pasar a la M.B., se retoma el comentario hecho por Ruiz Shulcloper en [4], donde dice que han trabajado con matrices de comparación de 36 y 125 filas, que fueron reducidos a una matriz básica con 4 y 15 filas, respectivamente. Es inmediato que la cantidad de unos que estas matrices contenían es mucho menor que los unos en M.D. . Esto directamente se refleja en una disminución de la complejidad al trabajar con las combinaciones de rasgos; además de hacer más eficiente la búsqueda reduce el tiempo de ejecución de la mayoría de los algoritmos que exiten para tales efectos.

El trabajo computacional que se requiere para pasar de una matriz de aprendizaje a un matriz de diferencias y luego de esta a una matriz básica es realmente sencillo.

De hecho en este momento los algoritmos de escala interior para el cálculo de todos los testores típicos (T.T.) de una M.A., que ponen especial atención en la estructura interna de la matriz de diferencia, tales como el CC, el CT [3], y los de escala exterior que ponen más atención en la conformación de los subconjuntos de rasgos, como el BT y el TB [3], han demostrado ser mucho más eficientes al trabajar con M.B.. Dicho sea de paso todos trabajaban con la matriz de diferencias y han sido adecuados para trabajar con la M.B.. En este sentido el algoritmo REC resultaba ser el único que trabaja con la matriz de aprendizaje, cosa que en cierto sentido y como se ha expresado antes lo colocaba en desventaja con respecto a los demás algoritmos.

Es natural pensar que esta reducción de la información pudiera alterar el conjunto de testores, o que evite de algún modo llegar a obtener todos los testores típicos, cosa que sería suficiente para regresar atrás y trabajar con la M.D. con todo y su complejidad.

En todos los algoritmos mencionados anteriormente, se ha demostrado que el paso de la matriz de diferencias a la matriz básica no altera el conjunto de testores y se conserva integro el conjunto de testores típicos. Además el ahorro en tiempo es notorio. La memoria utilizada es mucho menor, dado que el número de datos a almacenar es más pequeño. El número de comparaciones por realizar también disminuye y por tanto el número de operaciones.

Midiendo la complejidad del algoritmo REC al trabajar con M.A. se obtienen los siguientes resultados:

PASO	DESCRIPCION	ACCESOS	COMPARACIONES
1	$X = \{x_1,\ldots x_n\} = \mathcal{R}$	n	0
2	¿ $X = \emptyset$?	1	1
3	Calculo de $e_x(X_{1_s})$	$2\|X\|\cdot\mathcal{MD}$	$\|X\|\cdot\mathcal{F}\cdot\mathcal{MD}$
4	Si $e_x(X_{1_s}) = 0$ averigua si es inyectiva.	$\|X\|$	$\|X\|\frac{n(n+1)}{2}$
5	se aplica $\alpha(X)$	$\|X\|$	$\|X\|$
6	se aplica $\sigma(X)$	$\|X\|$	$\|X\|$
7	se anota el conjunto X	$\|X\|$	$c\|X\|$
8	se aplica $\beta(X)$	$\|X\|$	$\|X\|$

donde

$\mathcal{F}$:es la complejidad que produce aplicar el criterio de

comparación y que puede ser tan grande como se quiera.

$\mathcal{MD} := \sum_{i=1}^{\ell-1} \sum_{j=i+1}^{\ell} \|K_i\| \|K_j\|$, i.e. el número de combinaciones de objetos en clases distintas en M.A.

c: el número de comparaciones necesarias para eliminar todos los superconjuntos de X en la lista de testores.

Al trabajar con M.D. las complejidades de todos los pasos son las mismas, excepto para el paso 3, donde el número de accesos es $(\|X\| \cdot \mathcal{MD})$ y el número de comparaciones es $(\|X\| \cdot \mathcal{MD})$ y por lo tanto la complejidad en M.D. es mucho menor que en M.A., aún cuando para trabajar con M.D. se considera un "paso 0" donde se forma la matriz de diferencias que incrementa la complejidad en $\|X\| \cdot \mathcal{F} \cdot \mathcal{MD}$, que no se compara con el ahorro obtenido en el paso 3, ya que en M.A. esa cantidad se repetirá en el peor de los caso 2^n de veces que es el número de combinaciones a analizar. Resumiendo:

la complejidad del algoritimo sobre M.A. es $O(2^n(\|X\| \cdot \mathcal{F} \cdot \mathcal{MD}))$

la complejidad del algoritimo sobre M.D. es $O(2^n(\|X\| \cdot \mathcal{MD}))$

y finalmente, sobre M.B. es menor o igual a $O(2^n(\|X\| \cdot \mathcal{MD}))$.

Veamos pues ahora en el caso particular del algoritmo REC las adecuaciones necesarias para trabajar con la matriz básica.

V.3 DESARROLLO DE REC CON M.B.

Pensar en el algoritmo REC utilizando la matriz de diferencias en lugar de la de aprendizaje es un paso muy natural. Lo único que se modifica es que en lugar de que el algoritmo lleve a cabo (al calcular el error) la comparación rasgo a rasgo (del subconjunto X) para cada par de objetos, se suma el número de renglones de ceros que se producen para el subconjunto X de rasgos en cuestión.

El ahorro en tiempo y complejidad como se vió en el subcapítulo anterior es evidente, al llevar a cabo las comparaciones sólo una vez (al principio) antes de empezar a trabajar con el algoritmo, y ya trabajando con él cambiar estas comparaciones por una simple suma de renglones de ceros.

PROPOSICION 5.1: $e_{x_{M.A.}}(X_1) = e_{x_{M.D}}(X_1)$ y $\|E_{M.A}(X)\| = \|E_{M.D.}(X)\|$, donde $\|E_{M.A}(X)\|$ es el conjunto de la definición 3.2 sobre la matriz M.A. y $\|E_{M.D.}(X)\|$ el conjunto formado a partir de la matriz M.D..

DEMOSTRACION: El conjunto $E_{M.A}(X)$ se conformará por cada par de descripciones semejantes de objetos de clases distintas y $E_{M.D.}(X)$ por cada fila de ceros en M.D., pero cada par de descripciones semejantes provocará una fila de ceros en M.D. $\therefore \|E_{M.A.}(X)\| = \|E_{M.D.}(X)\|$ y $e_{x_{M.B}}(\alpha_1) = e_{x_{M.D}}$ ■

El paso del tratamiento del algoritmo con la matriz de diferencias al tratamiento del algoritmo con la matriz básica, requiere un análisis más detallado que permita tener la seguridad de que los saltos del algoritmo siguen barriendo todos los testores típicos, o en el caso no disjunto todas las combinaciones irreducibles de rasgos.

PROPOSICION 5.2: $\|E_{MD}(X)\| \geq \|E_{MB}(X)\|$.

DEMOSTRACION: Para cada fila de M.D. en la que una parte de ella provoque una subfila de ceros que incremente $E_{MD}(X)$ y por lo tanto incremente $\|E_{MD}(X)\|$, pueden ocurrir sólo dos cosas por la definición 5.3 : que sea fila básica o que no lo sea. En el primer caso $E_{MD}(X)$ se conserva igual a $E_{MB}(X)$. En el segundo caso ese renglón no formará parte de MB por lo tanto formará parte de

$E_{MD}(X)$ e incrementará $\|E_{MD}(X)\|$, pero no formará parte de $E_{MB}(X)$ y por lo tanto no incrementará $\|E_{MB}(X)\|$ entonces $\|E_{MD}(X)\| \geq \|E_{MB}(X)\|$. ■

PROPOSICION 5.3: Si X es testor en M.D. entonces X es testor en M.B.

DEMOSTRACION: Supongamos lo contrario, es decir, que existe una combinación de rasgos X que es testor sobre M.D. y no lo es sobre M.B., si X no es testor sobre M.B. es porque produce un error diferente de cero, lo que significa que en la submatriz definida por X se localizó por lo menos un renglón formado exclusivamente por ceros, pero como la M.B. esta formada por renglónes que pertenecian a M.D., este renglón también debió existir en la submatriz definida por X partiendo de M.D. lo que le impediría ser un testor para M.D., lo que es una contradicción.■

PROPOSICION 5.4: El conjunto de testores típicos encontrados en M.D por REC es igual al conjunto de testores típico encontrados en M.B. por REC.

DEMOSTRACION: Es obvio que el conjunto de testores encontrados sobre M.D. es mayor o igual que el conjunto de testores encontrados por REC sobre M.B., pero aseguramos que el conjunto de testores típicos encontrados por REC es el mismo para ambas matrices. Entonces se debe demostrar que el conjunto de testores que se pierden al trabajar sobre M.B. no eran testores típicos.

Denominemos T al conjunto de rasgos sobre el cual la función de error es igual a cero e inyectiva sobre M.D., por lo tanto se le considera en la lista de testores candidatos a testores típicos.

Si en el recorrido de ρ trabajando con M.B. T no es considerado en la lista de testores (es decir, es un testor localizado sobre M.D. pero no lo es sobre M.B.) puede ser por alguna de las siguientes 2 causas:

1) porque al llegar con la función de recorrido ρ, ese conjunto T no cubre los requisitos para ser considerado en la lista de testores. Es decir:

i) Porque T sobre M.B. no tiene una función de error inyectiva por lo tanto T en M.D. era testor pero no típico, ya que existe un subconjunto de él que es testor, a saber

$$\rho_{MB}^{n+1}(\mathcal{R}) = \rho_{MB}(T) = \sigma(T)$$

ii) Porque la función $e_T(T) \neq 0$ al trabajar sobre M.B. pero esto no puede ocurrir nunca porque sería equivalente a admitir que T es testor sobre M.D. y no lo es en M.B, lo que contradice la proposición 5.3.

2) Porque fué saltado por la función ρ_{MB}. Supongamos $X = \rho_{MB}^{n}(\mathcal{R})$ y que es falsa la proposición, es decir, que existe un T testor típico que no esta incluido en el recorrido definido por ρ_{MB}, dicho de otro modo ocurre que $\rho_{MB}^{n} \succ\cdot T \succ\cdot \rho_{MB}^{n+1}$. Pueden ocurrir 3 situaciones.

a) e_X no es inyectiva y X es testor. Por la definición 3.3

$$\rho_{MB}(X) = \rho_{MD}^{n+1}(\mathcal{R}) = \sigma(X).$$

Por la proposición 3.4 (caracterización de $\succ$) existe un $x_o \notin \sigma(X)$, $x_o \in T$ y para cualquier $x \in \sigma(X)$ con $x < x_o$ se tiene que $x \in T$. O sea que $\{x \in \sigma(X) | x < x_o\} = \{x \in T | x < x_o\}$. Como $\sigma(X) \subset X$ por la definición de σ entonces $\{x \in T | x < x_o\} \subset X$. Ahora ese elemento $x_o \in T$ y $x_o \notin \sigma(X)$ puede o no pertenecer a X. Analicemos por separado:

i) Si $x_o \in X$ entonces al aplicar la función σ fué eliminado del conjunto, pero σ sólo elimina los rasgos donde la función no es inyectiva y como estuvimos de acuerdo antes $\{x \in T | x<x_o\} \subset X$, ahora con x_o incluido, lo que nos lleva a concluir que la función de e_x no era inyectiva sobre T, lo que contradice la afirmación de que sea testor típico.

ii) Si $x_o \notin X$, como $Z = \{x \in X | x < x_o\} \in T$ tiene una función inyectiva sus elementos deben coincidir con X y $x_o \in T$ y $x_o \notin X$, entonces por la proposición 3.4 $T \succ X$, lo que contradice la hipótesis inicial.

b) Supongamos e_x inyectiva y X un testor. Por la definición 3.3 $\rho_{MB}(X) = \rho_{MB}^{n+1}(\mathcal{R}) = \beta(X)$. Como se dijo antes entre X y $\beta(X)$ no hay ningún conjunto ya que $\beta(X)$ es un decremento en 1 con respecto a X.

c) X no es testor. Entonces la definición 3.3 conlleva a aplicar $\rho_{MB}(X) = \rho_{MB}^{n+1}(\mathcal{R}) = \alpha(X)$. Pueden presentarse 2 casos, que $\alpha(X)$ sea o no igual a $\varnothing$. Si $\alpha(X) = \varnothing$ y consideramos que $X \succ T$ entonces $T \subset X$, pero X no es un testor, entonces T no puede ser testor, mucho menos típico. Si $\alpha(X) \neq \varnothing$ entonces por la definición de α y la proposición 3.12 podemos asegurar que X, T y $\alpha(X)$ son exactamente iguales en sus primeros (j-1)-ésimos elementos $\{x_{i_1},\ldots,x_{i_{j-1}}\}$, $x_{i_j} \in X$, $x_{i_j} \notin \alpha(X)$ y x_{i_j} puede o no pertenecer a T. Veamos este último punto en 2 partes:

i) Si $x_{i_j} \notin T$, entonces $T = \{x_{i_1},\ldots,x_{i_{j-1}}\} \cup T'$, donde $T' \subset \{r \in \mathcal{R} | r > x_{i_j}\}$ y $\alpha(X) = \{x_{i_1},\ldots,x_{i_{j-1}}\} \cup \{r \in \mathcal{R} | r > x_{i_j}\}$ y por lo tanto $T \subset \alpha(X)$ y por la definición 3.1 $\alpha(X) \succ T$ lo que contradice el orden supuesto al principio de la demostración.

ii) Si $x_{i_j} \in T$, entonces $X = \{x_{i_1},\ldots,x_{i_j}\} \cup \{x_{i_{j+1}},\ldots,x_{i_s}\}$

$$T = \{x_{i_1},\ldots,x_{i_j}\} \cup T'$$

definida como habíamos dicho antes, pero por la definición de x_{i_j}, $\{x_{i_{j-1}},\ldots,x_{i_s}\}$ son los últimos elementos de $\mathcal{R}$ por lo tanto $T' \subset \{x_{i_{j-1}},\ldots,x_{i_s}\}$, así que $T \subset X$, pero T no puede ser

testor típico y al mismo tiempo subconjunto de un conjunto que no es testor.

∴ No es posible que ρ_{MB} salte un testor típico en MB. Entonces sólo se puede concluir que si T era testor no era testor típico en el recorrido de REC sobre M.D.■

Veamos ahora un ejemplo. Consideremos la siguiente matriz de aprendizaje:

		x_1	x_2	x_3	x_4
K_1	O_1	0	0	1	0
	O_2	1	1	0	0
	O_3	0	0	0	1
K_2	O_4	1	1	1	1
	O_5	1	0	1	1

que produce la siguiente matriz de diferencia (M.D.):

x_1	x_2	x_3	x_4
1	1	0	1
1	0	0	0
0	0	1	1
0	1	1	1
1	1	1	0
1	0	1	0

lo que produce a su vez la matriz básica:

x_1	x_2	x_3	x_4
1	0	0	0
0	0	1	1

el recorrido del algoritmo REC sobre M.D. queda de la siguiente manera:

CONJUNTO X		ERROR DE CADA RASGO x_1	x_2	x_3	x_4	e_x. INY.	TESTOR	LISTA DE TESTORES
$\{x_1,x_2,x_3,x_4\}$	σ	2	1	0	0	NO	SI	
$\{x_1,x_2,x_3\}$	β	2	1	0	x	SI	SI	$\{\{x_1,x_2,x_3\}\}$
$\{x_1,x_2,x_4\}$	β	2	1	x	0	SI	SI	$\{\{x_1,x_2,x_3\},\{x_1,x_2,x_4\}\}$
$\{x_1,x_2\}$	α	2	1	x	x	--	NO	
$\{x_1,x_3,x_4\}$	σ	2	x	0	0	NO	SI	
$\{x_1,x_3\}$	β	2	x	0	x	SI	SI	$\{\{x_1,x_3\},\{x_1,x_2,x_4\}\}$
$\{x_1,x_4\}$	β	2	x	x	0	SI	SI	$\{\{x_1,x_3\},\{x_1,x_4\}\}$
$\{x_1\}$	α	2	x	x	x	--	NO	
$\{x_2,x_3,x_4\}$		x	3	1	1	--	NO	
$\{x_2,x_3\}$								
$\{x_2,x_4\}$								
$\{x_2\}$	α							
$\{x_3,x_4\}$								
$\{x_3\}$								
$\{x_4\}$								
$\{\ \}$		FIN.						

Este mismo recorrido de REC sobre M.B. se comporta como se aprecia en la tabla 1.

El número de conjuntos candidatos a testor típico generados en M.D. fueron 4, contra 2 que fueron generados con M.B., de los cuales en ambos casos el número de testores típicos es 2, que para ambas matrices y que como se demostró, son los mismos.

CONJUNTO X		ERROR DE CADA RASGO x_1	x_2	x_3	x_4	e_x. INY.	TESTOR	LISTA DE TESTORES
$\{x_1,x_2,x_3,x_4\}$		1	1	0	0	NO	SI	
$\{x_1,x_2,x_3\}$								
$\{x_1,x_2,x_4\}$	σ							
$\{x_1,x_2\}$								
$\{x_1,x_3,x_4\}$								
$\{x_1,x_3\}$	β	1	x	0	x	SI	SI	$\{\{x_1,x_3\}\}$
$\{x_1,x_4\}$	β	1	x	x	0	SI	SI	$\{\{x_1,x_3\},\{x_1,x_4\}\}$
$\{x_1\}$	α	1	x	x	x	--	NO	
$\{x_2,x_3,x_4\}$		x	2	1	1	--	NO	
$\{x_2,x_3\}$								
$\{x_2,x_4\}$								
$\{x_2\}$	α							
$\{x_3,x_4\}$								
$\{x_3\}$								
$\{x_4\}$								
$\{\ \}$		FIN.						

TABLA 1. RECORRIDO DE REC SOBRE M.B.

Cabe hacer la observación que pueden existir conjuntos en M.D. cuya función de error sea inyectiva e igual a cero y no lo sean en M.B. y viceversa, que sean inyectivos y con la función de error igual a cero en M.B. y no lo sean en M.D.. El recorrido de la función ρ no es la misma para M.D. que para M.B., en general hay algunos saltos en M.B. que son más grandes que en M.D.

CAPITULO VI.
c- TESTORES.

VI.1 CONCEPTOS BASICOS.

Como se vió antes en el concepto de Zhuravliov de testor se considera que un conjunto de columnas (rasgos) de T_{anl} es testor si no existen filas iguales en clases diferentes.

Al hablar de filas iguales aparece de forma implícita una función de semejanza booleana:

$$f(O_i,O_j) = \begin{cases} 1 & \text{si } s_i = n \\ 0 & \text{en otro caso} \end{cases} \qquad (1)$$

donde la función toma el valor de 1 si los objetos O_i y O_j descritos con n rasgos son semejantes y 0 si son diferentes, s_i es el número de rasgos donde O_i y O_j deben coincidir.

La matriz básica (MB) es una matriz booleana cuando se utilizan criterios de comparación booleanos, como se ha venido trabajando hasta este punto. La definición de testor a partir de la MB habla de un subconjunto de columnas (rasgos) de MB que definen un submatriz que no contiene filas formadas exclusivamente por ceros(en el caso en que todos los criterios de comparación sean de diferencia y por lo tanto cada cero represente la igualdad de los valores comparados), y así mismo, un testor típico es un conjunto de columnas al cual no podemos eliminar ninguna columna sin que deje de ser testor.

Es importante verificar cuando se lleva a cabo un proceso de clasificación que la función de semejanza que se utilice sea la misma que se utilizó para encontrar los testores típicos, ya que

de no ser así se corre el riesgo de trabajar con 2 conceptos de analogía diferentes que den como resultado una clasificación erronea.

DEFINICION 6.1. Un conjunto de columnas T de MB es un c-testor si la submatriz que este define en MB contiene todas sus filas formadas por al menos c+1 unos(c+1 rasgos con valores de diferencia).

DEFINICION 6.2 Un conjunto de columnas T de MB es un c:testor típico si cualquier subconjunto propio de él no es un c:testor.

Algunas de las propiedas relacionadas con estos conceptos consideradas en [6] son:

PROPIEDAD 6.1. la longitud mínima de un c-testor es c+1 y los c:testor de longitud mínima son típicos.

PROPIEDAD 6.2. Si un conjunto de columnas T no es un c-testor entonces para todo c' > c T no es un c':testor.

PROPIEDAD 6.3. Si un conjunto de columnas T es un c:testor típico entonces para todo c'>c T no es un c':testor.

PROPIEDAD 6.4. Si un conjunto de columnas T es un c:testor entonces para todo c'<c T es un c':testor.

PROPIEDAD 6.5. Si un conjunto de columnas T es un c:testor típico entonces para c'<c T es un c':testor pero no típico.

VI.2 EXTENSIONES TEORICAS DE REC PARA c-TESTORES.

Por las definiciones del subcapítulo anterior se puede ver que el concepto de c-testor modifica directamente la función de

semejanza entre los objetos. De manera implícita esta semejanza entre pares de objetos se ha considerado cuando los valores que tienen rasgo a rasgo son iguales. A partir de este punto retomaremos la función de semejanza definida en el subcapítulo anterior, para $s_1 \geq c$, es decir, dos objetos pueden ser semejantes si por lo menos tienen valores semejantes en s_1 rasgos. De la necesidad de tener por lo menos c+1:rasgos diferenciantes se deduce que los subconjuntos de rasgos candidatos a testor típico, tienen una cardinalidad por lo menos de c+1.

Si razonamos en el concepto de error definido en 3.2 y verificamos la manera en que se recorren los subconjuntos a partir de la definición 3.3 podemos observar que en términos de los c-testores un conjunto puede producir un error igual a cero, es decir ser c:testor y no tener una función de error inyectiva. Si procedieramos como lo indica el algoritmo REC deberíamos aplicar bajo estas circunstancias al conjunto en cuestión la función sigma(σ), esperando obtener un conjunto cuyo error sea igual a cero y su función de error se comporte de manera inyectiva. Desgraciadamente esto no sucede en los c-testores, como se ve en el siguiente ejemplo:

ejemplo: Supongamos una matriz básica:

α_1	α_2	α_3	α_4	α_5	α_6
0	0	1	1	0	1
1	1	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1

Si iniciamos el algoritmo REC pretendiendo localizar los

2:testores de la matriz con el conjunto inicial de rasgos

$$X = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6\}$$

encontraremos que el error del conjunto X es cero, así que analizamos su función de error

$$e_X(\alpha_1) = 4 \qquad e_X(\alpha_4) = 3$$
$$e_X(\alpha_2) = 4 \qquad e_X(\alpha_5) = 3$$
$$e_X(\alpha_3) = 3 \qquad e_X(\alpha_6) = 0$$

que resulta no ser inyectiva. Aplicamos entonces $\rho(X) = \sigma(\mathcal{R})$ para eliminar los rasgos que provocan esta no-inyectividad, obteniendo así $\sigma(X) = \{\alpha_1, \alpha_3, \alpha_6\}$, ¡pero entonces el error deja de ser igual a cero y el nuevo conjunto sigue sin ser inyectivo!

$$e_x(X_1) = 4, \qquad e_x(\alpha_3) = 4, \qquad e_x(\alpha_6) = 2$$

lo que contradice el corolario de la proposición 3.7.

¿Qué es lo que provoca esta situación?. El haber cometido un error conceptual al considerar que en la función de error con los c:testores, el error provocado por $e_x(\alpha_1)$ es igual al error provocado por $e_x(\alpha_2)$ y peor aún, pensar que los conjuntos X_3, X_4, y X_5 provocan el mismo error. Cuando tratamos con 0:testores, los errores eran los mismos en base a encontrar renglones exclusivamente formados por ceros y no remediar esta situación al agregar la columna correspondiente al siguiente rasgo del conjunto si el error cambiaba era signo de que en algún renglón, esa columna había agregado por lo menos un 1 en MB.

El error en los c:tesores para $e_x(\alpha_{i_j})$ y $e_x(\alpha_{i_{j+1}})$ no puede ser considerado igual debido a que pueden suceder dos cosas distintas:

i) La columna correspondiente a la característica $\alpha_{i_{j+1}}$ no aporta ningún nuevo uno para esos renglones en los que los c+1 unos no se han completado (y por tanto se consideran de error) con lo que la columna no aporta nada y daría lo mismo no considerarla.

ii) Que por lo menos en uno de los renglones de error la columna $\alpha_{i_{j+1}}$ aporta por lo menos un uno más, que aún cuando sigue sin reunir los c+1 reduce la distancia para alcanzar este número. Por este motivo esta columna sigue causando error pero va creando condiciones para que el renglón deje de cometerlo.

Para poder distinguir entre estos dos tipos de errores se propone la siguiente definición

DEFINICION 6.3: Sea $X \subseteq \mathcal{R} = \{\alpha_1,\ldots\alpha_n\}$. Denotemos por $cE(X)$ al conjunto de renglones de M.B. que en la submatriz definida por X no contengan por lo menos c+1 unos. Se dirá que <u>el error del renglón j-ésimo con respecto a</u> X_{i_k} denotado por <u>eren</u>(j,X_{i_k}) es igual a la diferencia entre c+1 y la suma de unos en el renglón j hasta el rasgo i_k, si esta diferencia es mayor o igual a cero y cero en otro caso, es decir:

$$\text{eren}(j,X_{i_k}) = \begin{cases} c+1 - \sum_{h=1}^{k} a_{ji_h} & \text{si } \sum_{h=1}^{k} a_{ji_h} \leq c+1 \\ 0 & \text{en otro caso.} \end{cases}$$

donde a_{ji_h} es el i_h-ésimo elemento, del j-ésimo renglón de la matriz básica. Se dirá que <u>el epsilón error de X</u> , denotado por $ce_x(\alpha)$ es la suma de todos los eren(j, X) de la matriz,

$$ce_x(\alpha) = \sum_{j=1}^{s} \text{eren}(j,\ X)$$

donde s es el número de renglones de la matriz en cuestión.

Esta definición como se habrá notado es un símil de la definición 3.2 y se aplica a la matriz básica cuando se consideran clases no disjuntas, la definición de $ET(\mathcal{R})$ a partir de este momento hará referencia a esta definición de $cE(X)$.

La función ce_x permanece constante siempre y cuando la nueva columna no agrege unos en alguno de los renglones que aún no han completado sus c+1 unos o si agrega unos solo en renglones donde ya existian los c+1 unos.

En el sentido en que la función de semejanza se plantea se podría pensar en modificar el orden de los subconjuntos de $\mathcal{R}$, propuesto en la definición 3.1 y las proposiciones 3.1 a la 3.4, de tal forma que aquellos subconjuntos con cardinalidad menor a c+1 ni siquiera intervinieran en el orden, es decir, no fueran considerados, de tal modo que no formarán parte del recorrido de las funciones definidas en 3.2., pero esto incrementaría la complejidad al tener que verificar la cardinalidad para cada conjunto considerado por el recorrido.

Una manera más optima de considerar la cardinalidad de los conjuntos es verificarla sólo para aquellos conjuntos candidatos a testores típicos, dicho de otro modo, el cambio que se plantea es que en el momento que se localiza un conjunto cuya función de error es igual a cero se verifique después si es c-testor, es decir, que tenga por lo menos c+1 unos.

PROPOSICION 6.1: X es un ctestor si y sólo si el epsilón error de X es igual a cero.

DEMOSTRACION: Si el error de X es igual a cero, entonces el conjunto X define una submatriz donde ningún par de descripciones de objetos de clases distintas está siendo confundida, para lo que se requiere que en la M.B. haya por lo menos c+1 rasgos diferenciantes y por la definición 6.1 es un ctestor. Por otro lado, si es un ctestor entonces por las propiedades de los ctestores debe tener una cardinalidad de por lo menos c+1, con

c+1'unos por renglón entonces diferencia a todos los objetos sin cometer un sólo error, por lo tanto tiene un ce(x) igual a cero.

COROLARIO: la cardinalidad del conjunto X es mínimamente c+1.

PROPOSICION 6.2: Si X es un ctestor típico entonces ce_x es inyectiva.

DEMOSTRACION: Sea $X = \{\alpha_{i_1}, \ldots \alpha_{i_s}\}$ un testor típico tal que existan dos rasgos $\alpha_{i_h} < \alpha_{i_j}$ que provocan que la función no sea inyectiva, es decir, $ce_x(\alpha_{i_h}) = ce_x(\alpha_{i_j})$ lo que implica que

$$ce_x(\alpha_{i_1}) > \ldots > ce_x(\alpha_{i_h}) = ce_x(\alpha_{i_{h+1}}) = \ldots = ce_x(\alpha_{i_j}) > ce_x(\alpha_{i_{j+1}}) > \ldots > ce_x(\alpha_{i_s})$$

Pero entonces se puede deducir que, desde $\alpha_{i_{h+1}}$, hasta α_{i_j} no se aporta información distinta a la que se aporta hasta α_{i_h}, en cuanto a la diferenciación entre los objetos de interés, dicho de otro modo en cuanto a completar los c+1 por renglon, por lo que podríamos eliminar estos rasgos del conjunto X sin dejar de tener un ctestor, lo que contradice el hecho de que X sea un ctestor típico.

Establecidas las condiciones al definir el epsilón error, para caracterizar a los ctestores y los candidatos a ctestores típicos reconstruyamos la función de recorrido:

DEFINICION 6.5: Sea $c\rho : \mathcal{P}(\mathcal{R}) \longrightarrow \mathcal{P}(\mathcal{R})$. Entonces

a) si ce_x no es inyectiva y X es ctestor,
   entonces $c\rho(X) = \sigma(X)$.
b) si ce_x es inyectiva y X es un ctestor,
   entonces $c\rho(X) = \beta(X)$.

c) en otro caso $c\rho(X) = \alpha(X)$, es decir, X no es ctestor.

Las definiciones de σ,β,α seguiran contruyendose de la forma establecida en la definición 3.3.

Veamos ahora algunas de las propiedades de estas funciones con los ctestores.

PROPOSICION 6.3: Sea $X \subset \mathcal{R}$ y $cE(X)$ como en la definición 6.3 Entonces

i) $cE(\sigma(X)) = cE(X)$ y

ii) $ce_{\sigma(x)}(\alpha) = ce_x(\alpha)$ para cualquier $\alpha \in \sigma(X)$.

DEMOSTRACION: Pueden darse 2 casos: que ce_x sea inyectiva o que no lo sea.

En el 1er. caso, $\sigma(X)$ y X serían iguales por lo tanto es inmediato que $cE(\sigma(X)) = cE(X)$ y $ce_{\sigma(x)}(\alpha) = ce_x(\alpha)$ para cualquier $\alpha \in \sigma(X)$.

En el segundo caso, ce_x no es inyectiva sobre X pero los elementos en común con $\sigma(X)$, están en la parte donde si es inyectiva, porque $\sigma(X) = X \setminus \{\alpha_{i_{j+1}},\ldots,\alpha_{i_k}\}$, donde $e_{x_{i_{j+1}}}(\alpha) = e_{x_{i_{j+2}}}(\alpha) = \ldots = e_{x_{i_k}}(\alpha)$ por lo tanto $e_{\sigma(x)}(\alpha) = e_x(\alpha)$.

por otra parte $cE(X)$ se forma a partir de los renglones iguales en la submatriz definida por X pero como ya dijimos antes el subconjunto no inyectivo $\{\alpha_{i_{j+1}},\ldots\alpha_{i_k}\}$ no confunde ni a más ni a menos elementos de los que confunde $\sigma(X)$. $\therefore$ $cE(\sigma(X)) = cE(X)$ y $ce_{\sigma(x)}(\alpha) = ce_x(\alpha)$ para cualquier $x \in \sigma(X)$. ∎

COROLARIO: $ce_{\sigma(X)}$ es una función inyectiva sobre $c\sigma(X)$.

DEMOSTRACION: Por lo descrito en el segundo caso de la demostración anterior y por la propia definición de $\sigma(X)$, queda demostrado. ∎

PROPOSICION 6.4: Si $X \subset \mathcal{R}$ es un testor entonces $\sigma(X)$ es también un testor.

DEMOSTRACION: Como ya se demostro anteriormente, $e_{\sigma(X)}(\alpha) = e_x(\alpha)$ Si X es testor entonces $e_x(\alpha) = 0$ entonces $e_{\sigma(X)}(\alpha) = 0$, que lo caracteriza también como testor. ∎

PROPOSICION 6.5 No existe ctestor típico alguno cT tal que $c\rho^n(\mathcal{R}) \succ cT \succ c\rho^{n+1}(\mathcal{R})$.

DEMOSTRACION: Como antes, supongamos lo contrario y lleguemos a una contradición. Denotemos $X = c\rho^n(\mathcal{R})$. Por la definición de ρ existen 3 situaciones:

a) c:e_X no es inyectiva y X es testor.;Entonces $c\rho(X) = c\rho^{n+1}(\mathcal{R}) = \sigma(X)$. Como $\sigma(X)$ lo que hizo fué quitar todas aquellas variables para las cuales ce_x no era inyectiva sobre X y suponemos que cT está entre X y $\sigma(X)$, entonces cT debe tener variables sobre las que $e_{c:T}$ no es inyectiva, lo que contradice que sea ctestor típico.

b) c:e_X es inyectivo y X es testor. Entonces

$$c\rho^n(X) = c\rho^{n+1}(\mathcal{R}) = \beta(X).$$

Pero entre X y $\beta(X)$ no existe ningún conjunto, ya que $\beta(X)$ es el conjunto inmediato de X por definición.

c) X no es e:testor. Entonces $c\rho(X) = c\rho^{n+1}(\mathcal{R}) = c\alpha(X)$. Pueden suceder 2 cosas:

1) $\alpha(X) = \emptyset$. esto nos lleva a deducir que X estaba conformado por los últimos elementos de $\mathcal{R}$, entonces $cT \subset X$, pero como X no es testor, un subconjunto suyo no puede ser testor, mucho menos típico.

2) $\alpha(X) \neq \emptyset$. En este caso los tres conjuntos coinciden en sus j-1 primeros elementos por la definición 3.12. Ahora el elemento α_{1_j}, que es el $\max\{\alpha \in X |\ r \in \mathcal{R}\backslash X$ con $\alpha < r\}$ pertenece a X y no

pertenece a $\alpha(X)$ pudiendo o no pertenecer a cT. Veamoslo por partes:

i) $\alpha_{i_j} \in cT$ entonces $X = \{\alpha_{i_1},\ldots,\alpha_{i_j}\} \cup \{\alpha_{i_{j+1}},\ldots,\alpha_n\}$

$$Y = \{\alpha_{i_1},\ldots,\alpha_{i_j}\} \cup \{t_{j+1},\ldots,t_m\}$$

pero por la definición de α_{i_j}, $\{\alpha_{i_{j+1}},\ldots,\alpha_n\}$ son los últimos elementos de $\mathcal{R}$ luego $\{t_{j+1},\ldots,t_m\} \subset \{\alpha_{i_{j+1}},\ldots,\alpha_n\}$ de donde se concluye que $cT \subset X$, lo que no le permite ser testor típico al mismo tiempo que subconjunto de un conjunto que no es testor.

ii) Si $\alpha_{i_j} \notin cT$ entonces $cT \subset \alpha(X)$ lo que implicaría que

$$\alpha(X) \succ cT,$$

lo que contradice la suposición inicial.

Ejemplo: Considereos la siguiente matriz de básica:

α_1	α_2	α_3	α_4	α_5	α_6
0	0	1	1	0	1
1	1	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1

En la siguiente hoja se presenta el recorrido en busca de 2-testores:

	RASGOS						ERROR e_x						e_x INY	TESTOR	LISTA DE TESTORES
	α_1	α_2	α_3	α_4	α_5	α_6	α_1	α_2	α_3	α_4	α_5	α_6			
	1	1	1	1	1	1	11	8	5	3	3	0	NO	SI	
σ	1	1	1	1	1	0									
	1	1	1	1	0	1	11	8	5	3	X	0	SI	SI	$\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_6\}$
β	1	1	1	1	0	0	11	8	5	3	X	X	--	NO	
α	1	1	1	0	1	1	11	8	5	X	5	2	--	NO	
	1	1	1	0	1	0									
α	1	1	1	0	0	1									
	1	1	1	0	0	0									
	1	1	0	1	1	1	11	8	X	6	5	2	--	NO	
	1	1	0	1	1	0									
	1	1	0	1	0	1									
	1	1	0	1	0	0									
α	1	1	0	0	1	1									
	1	1	0	0	1	0									
	1	1	0	0	0	1									
	1	1	0	0	0	0									
	1	0	1	1	1	1	11	X	8	6	5	3	--	NO	
	1	0	1	1	1	0									
	1	0	1	1	0	1									
	1	0	1	1	0	0									
	1	0	1	0	1	1									
	1	0	1	0	1	0									
	1	0	1	0	0	1									
α	1	0	1	0	0	0									
	1	0	0	1	1	1									
	1	0	0	1	1	0									
	1	0	0	1	0	1									
	1	0	0	1	0	0									
	1	0	0	0	1	1									
	1	0	0	0	1	0									
	1	0	0	0	0	1									

continua en la siguiente hoja...

	RASGOS	ERROR e_x	e_x INY	TESTOR	LISTA DE TESTORES
	x_1x_2 x_3x_4 x_5x_6	x_1 x_2 x_3 x_4 x_5 x_6			
	1 0 0 0 0 0				$\{x_2,x_3,x_4,x_5,x_6\}$
β	0 1 1 1 1 1	X 9 6 4 3 0	SI	SI	$\{x_1,x_2,x_3,x_4,x_6\}$
	0 1 1 1 1 0	X 9 6 4 3 X	--	NO	
α	0 1 1 1 0 1	X 9 6 4 X 1	--	NO	
α	0 1 1 1 0 0				
	0 1 1 0 1 1	X 9 6 X 5 2	--	NO	
	0 1 1 0 1 0				
	0 1 1 0 0 1				
α	0 1 1 0 0 0				
	0 1 0 1 1 1	X 9 X 7 6 3	--	NO	
	0 1 0 1 1 0				
	0 1 0 1 0 1				
	0 1 0 1 0 0				
α	0 1 0 0 1 1				
	0 1 0 0 1 0				
	0 1 0 0 0 1				
	0 1 0 0 0 0				
	0 0 1 1 1 1	X X 9 7 6 3	--	NO	
	0 0 1 1 1 0				
	0 0 1 1 0 1				
	0 0 1 1 0 0				
	0 0 1 0 1 1				
	0 0 1 0 1 0				
α	0 0 1 0 0 1				
	0 0 1 0 0 0				
	0 0 0 1 1 1				
	0 0 0 1 1 0				
	0 0 0 1 0 1				
	0 0 0 1 0 0				
	0 0 0 0 1 1				
	0 0 0 0 1 0				
	0 0 0 0 0 1				
	0 0 0 0 0 0	F I N .			

En este ejemplo en particular el desempeño de la función α fué muy valioso, ya que permitió dar saltos bastante grandes. El conjunto de candidatos a testor fué exactamente igual que el conjunto de testores típicos.

CONCLUSIONES.

En los procesos de Reconocimiento de patrones una de las etapas más importantes es la selección de variables.

Un buen método para seleccionar las variables de un problema, ya sea para fines descriptivos, de comprensión del espacio de representación, u otros fines, resulta ser el algoritmo REC.

Los resultados obtenidos en la presente tesis, se desarrollaron en dos sentidos: el primero, era obtener un algoritmo, mucho más flexible y real, para lo cual se extendieron las definiciones y conceptos, de manera que fuese posible trabajar:

- con clases disjuntas, es decir, con elementos multi-clasificados.
- sobre la matriz básica
- con epsilón testores.

La extensión del algoritmo para que pueda trabajar con clases disjuntas le permite aplicarse a una gama más amplia de problemas reales.

Hacer que el algoritmo trabaje sobre la matriz básica, permite un ahorro en memoria y tiempo, lo que resulta en una reducción de la complejidad en la búsqueda de los testores típicos hecha por REC.

La introducción del concepto de ε-testores, logra flexibilizar la función de semejanza entre los objetos, permitiendo una modelación más eficaz y adecuada para ciertos problemas, lo que se traduce en una mejor interpretación del mundo real.

En resúmen, las modificaciones llevadas a cabo permiten contar con un algoritmo para la localización de testores típicos más eficiente, flexible y por lo tanto útil, todo en el primer sentido.

En el segundo sentido del desarrollo de la tesis, se llevó a cabo toda una sustentación matemática como soporte a las inovaciones descritas en los parrafos precedentes. Esta parte del trabajo no fué fácil y representa una de las partes más valiosas del presente trabajo.

REC ha sido codificado con estas adiciones en el algortimo que se presenta en el APENDICE 1 (llamado RECPLUS), y que queda implementado de tal forma que permita seguir siendo ampliado en cuanto al tipo de variables a considerar, a la elección de criterios de comparación y cuenta con un descriptor de archivos para facilitar su interación con el usuario. Es considerado como un subsistema auxiliar de PROGNOSIS que es un sistema de clasifición realizado en el INSTITUTO DE CIBERNETICA, MATEMATICA Y FISICA de la Academia de Ciencias de Cuba, por el grupo de Reconocimiento de Patrones.

DESARROLLO COMPUTACIONAL DE RECPLUS

IMPLANTACIÓN DEL ALGORITMO REC.

El algoritmo RECPLUS es un programa que tiene por objetivo encontrar todos los testores o e-testores para un ε mayor o igual a cero a partir de una matriz de aprendizaje.

La matriz de aprendizaje puede tener rasgos booleanos, k-valentes y reales. Puede tener elementos multiclasificados.

El primer aspecto a considerar cuando se desarrolla un sistema es el lenguaje computacional que se empleará. Se debe considerar la portabilidad del lenguaje, las capacidades para escribir de manera modular, los factores humanos y técnicos, y finalmente la rapidez del lenguaje. Además de los criterios técnicos, se debe considerar el costo.

Se tomó la decisión de emplear el lenguaje C, ya que este lenguaje cumple con los requerimientos de portabilidad que se requieren, ya que en los futuros proyectos de expansión del sistema se podría trasladar a una estación de trabajo o simplemente a otro sistema operativo. El lenguaje C, permite además trabajar el código con modularidad, requisito indispensable cuando se pretende integrar el sistema a un sitema más grande, o bien, cuando se considera la posibilidad de poder compartir las funciones que serán aquí desarrolladas para poder servir como codigo reusable de otros proyectos.

Por otra parte se requería un lenguaje que permitierá el manejo de estructuras de datos como listas simplemente ligadas, circulares y doblemente ligadas, así como otras estructuras que necesitaban manejo de memoria de manera dinámica, ya que el algoritmo trabaja a partir de una Matriz de Aprendizaje, cuyo tamaño en ocasiones no conoce ni el propio usuario al momento de iniciar su interacción con el sistema. Se requeria además contar con estructuras que permitieran cambiar el tipo de los datos en el transcurso de la ejecución. Además, era requisito utilizar un lenguaje que permitiera el manejo de archivos de información de una manera sencilla y confiable. Finalmente se requeria un lenguaje que fuera ampliamente difundido en el sentido de que cualquier usuario pudiera trabajarlo y en un momento dado modificarlo, en cualquier computadora convencional. Todo lo anterior era cubierto de manera satisfactoria por el lenguaje C.

Para finalizar, estaba la consideración de que este sistema, será un subsistema de un sitema mucho más grande para la clasificación como es PROLONGAIS. Dicho sistema existe ya de manera comercial y fué desarrollado en lenguaje C, con una interficie de usuario en C++. Actualmente, se desarrolla en el Instituto de Cibernetica, Matemática y Física de la Academia de Ciencias de Cuba, una versión en ambiente de Ventanas.

Elegido el lenguaje, se procedio ha analizar las partes que requeria el lenguaje. Se necesitaba un modulo de lectura de los datos; un modulo que trasladara la Matriz de Aprendizaje a la Matriz de Diferencias y finalmente a la Matriz Básica; y un modulo que aplicará el algoritmo como tal.

El primer modulo debía permitir introducir los datos directamente o poder leerlos de un archivo de datos. La primera opción conllevaba entonces a diseñar una pantalla de captura de la Matriz de Aprendizaje, que por sus dimensiones generalmente provoca muchos errores de captura, por lo cuál, se debía elaborar un analizador lexicográfico, que fungiera en determinados momentos como "guardián" de la entrada de datos, es decir, que informará en la medida de los posible cuando los datos introducidos no fueran factibles en función de el tipo de dato especificado por el propio usuario. Además se requería que esta pantalla de captura fuera lo más flexible posible, en el sentido de permitir introducir los datos de la Matriz en cualquier orden, permitiendo además "navegar" a través de la tabla de datos y modificar su valor cuantas veces fuese necesario durante el proceso de captura. La segunda opción debía poder brindar ayuda al usuario en cuanto a los archivos existentes y las características y tipo de información que estos guardaban. Para llevar a cabo esta tarea fue que se implanto un descriptor de archivos, que además de cumplir con este cometido, se encargará de dar información al sistema en el transcurso de sus operaciones ahorrando cálculos reiterados o tardados.

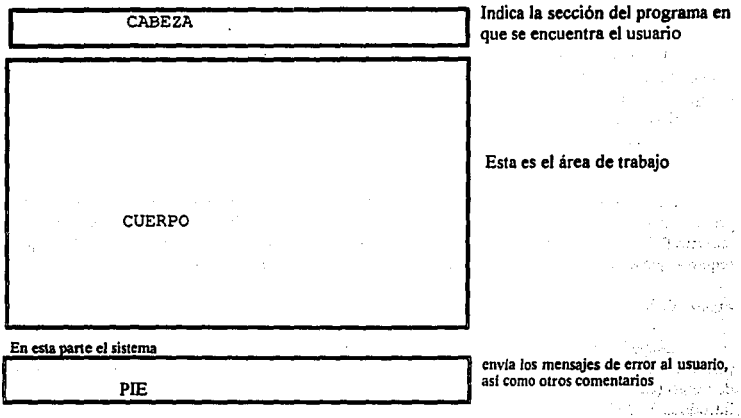
El segundo modulo, que debía trasladar la Matriz que se recibía como entrada en la Matriz de Diferencias y luego ésta, a la Matriz de Básica, debía además de solicitar los criterios de comparación de cada rasgo de la matriz, localizar los objetos multiclasificados y considerar la ausencia de información. Esta es la parte que más dinámicamente trabajaría la memoria y por lo tanto es aquí donde se debía tratar de hacer las mayores optimizaciones. Pensando en esto, se pensó en hacer procesos "virtuales", como el de no generar nunca como tal la Matriz de Diferencias, sino únicamente generar un renglón de dicha matriz cada vez y verificar si este era o no una fila básica, de modo tal que la memoria utilizada para la Matriz de Aprendizaje nunca excediera el tamaño de un renglón informacional, y al mismo tiempo ir generando la Matriz Básica que en la mayoría de las ocasiones es mucho mas pequeña que la Matriz de Diferencias, esto al final se traduciría en un ahorro en tiempo y espacio. La comparación entre los rasgos a partir de un criterio de comparación se encuentra dispuesto dentro de una estructura "case" lo que permitirá seguir agregando funciones de comparación sin necesidad de modificar ninguna otra parte del programa.

El tercer modulo, que aplicaría en si el algoritmo, debía ser completamente modular ya que las futuras extensiones del algoritmo, modificaran en esencia el desempeño de las funciones y los criterios que se siguen para aplicarlas.

Dicho todo lo anterior, pasemos ahora a ver el desarrollo de estos objetivos.

MANUAL DE USUARIO.

En todo el programa la pantalla estará dividida en tres cajas con las siguientes funciones:



LECTURA DE DATOS

Existen dos formas de leer la matriz de aprendizaje: por archivo o por teclado. Por archivo se pueden leer matrices que se han introducido con anterioridad, por teclado se despliega una pantalla de captura que permite introducir una Matriz de Aprendizaje ya que el programa cuenta con un editor para tal efecto.

LECTURA DE DATOS POR TECLADO

Para introducir una matriz por teclado, al iniciar el programa aparece, la pregunta expresa de como serán introducidos los datos:

>la introduccion de los rasgos será por:(t)teclado (a)archivo

el usuario, debe entonces teclear: t.

El programa entonces preguntará el nombre del primer rasgo, mediante la siguiente pregunta:

>Dame el nombre del rasgo 1.

El nombre de cada rasgo puede estar constituido por letras, números o símbolos y se da por concluido con un enter.
La siguiente pregunta del programa es acerca de el tipo del rasgo que puede ser:

(b) booleano
(r) real
(k) k-valente

Para informar al sistema acerca de nuestra elección se debe teclear la letra b, r ó k. El primer tipo admitirá solo valores de 0 ó 1. El segundo tipo admite cualquier valor real, de máximo 9 dígitos y un punto en cualquier sitio siempre y cuando no sea el primero . El tercer tipo. requiere que el usuario introduzca el valor de k mediante la siguiente pregunta:

>valor de k:

Después de introducir la letra elegida seguida por un enter el sistema volverá a solicitar un nuevo nombre de variable o rasgo y el proceso se repetirá hasta que en lugar de recibir un nombre de rasgo (cadena de caracteres) se reciba únicamente un enter. Esto informará al sistema que la introducción de la lista de rasgos ha concluido

La siguiente pregunta que hará el sistema es el número de objetos que constituyen la matriz de aprendizaje, sin importar a que clase pertenecen., Aquí se espera siempre un valor entero positivo.

Entonces aparecerá la pantalla de captura de la matriz de aprendizaje(fig 1), que tiene como títulos los nombres de los rasgos y en la columna a la izquierda la lista de objetos. La primera columna dentro de la tabla debe contener la clase a la que pertenece el objeto. Es importante que todos los objetos que pertenecen a la misma clase sean introducidos uno seguido de otro, en bloques.

Dentro de la pantalla de captura el usuario puede moverse a cualquier casilla mediante las fechas del teclado. Puede corregir cualquier dato borrando antes la información anterior con la tecla BackSpace y para indicar que a concluido la introducción de los datos debe presionar la tecla Esc. Es importante que la tecla ESC no sea presionada sino hasta estar completamente seguros de que la información introducida es la correcta. Cuando se carece de la información de un rasgo para un objeto se puede introducir un asterisco "*" o bien se puede dejar en blanco el espacio correspondiente al cruce de la columna respectiva al rasgo y el renglón del objeto. En general, en los sucesivo denotaremos por casilla al espacio que se encuentra en el cruce de cada columna (una columna por rasgo) y cada renglón (un renglón por Objeto). Enseguida de la fig 1, se muestra un lista de las teclas que se pueden emplear y la función que tienen asociada.

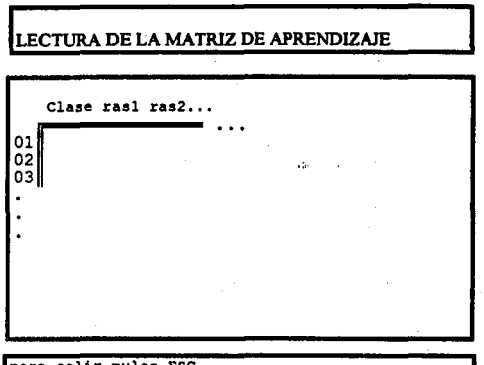


FIG.1 pantalla de captura de la Matriz de Aprendizaje

Tecla	función
→	Traslada a la casilla inmediata de la izquierda
←	Traslada a la casilla inmediata de la derecha
↑	Traslada a la casilla inmediata superior
↓	Traslada a la casilla inmediata inferior
BackSpace	Borra el contenido de la casilla en la que esta posicionado el cursor.
Esc	Indica que se ha concluido la entrada de la Matriz de Aprendizaje.
0..9	Para introducir los valores para los números reales (Se admite el punto).
A..Z	pueden ser usados exclusivamente en la columna correspondiente a la clase.
*	Indica que se carece de la información para esa casillas.

En caso de que el usuario emplee una tecla distinta a las anteriores el programa mandara un mensaje de error en la parte inferior.

Al terminar de introducir la matriz de Aprendizaje el sistema desplegará en pantalla la matriz con la que va a trabajar.

En la parte inferior de la siguiente pantalla aparecerá la pregunta:

>¿Se graba en archivo esta matriz?(s/n):

contestar que no, hace que el programa pase a la sección de criterios de comparación. Contestar que si, creará un archivo con el nombre que en ese momento es solicitado en el usuario:

>Introduce el nombre del nuevo archivo:

Este nombre debe comenzar con un carácter, tener una longitud de 8 ó menos caracteres y por uniformidad debe tener extensión .DAT, esto es recomendable pero no obligatorio.

El programa introducirá este nombre y sus caracteristicas asociadas en el descriptor de archivos. Luego pasará a la lectura de criterios de comparación.

LECTURA DE DATOS POR ARCHIVO

Para leer una matriz por archivo el usuario debe introducir en la primera pregunta:

```
>la introduccion de los rasgos será por: (a)archivo (t)teclado
```

la letra: a.

Entonces el programa preguntara por el nombre del archivo que se desea leer:

```
>Nombre del Archivo:
```

Los nombres de archivos no contienen mas de 8 caracteres, comienzan con una letra y para uniformidad deben tener extensión .DAT . Es recomendable que sean tecleados con mayúsculas.

Con esta información el programa abre el descriptor de archivos y busca el nombre que le fue proporcionado. Si lo encuentra despliega en la pantalla la matriz leída. En caso contrario, informa al usuario que el archivo no fue encontrado y pregunta si éste desea ver los archivos existentes en el descriptor de archivos mediante la siguiente pregunta:

```
>¿Deseas ver los archivos disponibles?
```

En caso afirmativo se empezará a desplegar en pantalla el descriptor. Los archivos que contiene el descripto comienzan con la descripción de el mismo. Para cada uno de los archivos restantes, se despliega:

- el nombre del archivo
- el numero de objetos que contiene
- el numero de rasgos

y seguido de cada entre:

- cada uno de los nombres de sus rasgos con su respectivo tipo.

La siguiente pregunta que hace el sistema es:

```
>¿Quieres introducir nuevamente el nombre del archivo?
```

Si la repuesta es **s** (si) el programa volverá a solicitar un nombre de archivo. En caso de que la respuesta sea **n** (no), el programa terminará su ejecución.

Cuando el nombre del archivo existe en el descripto, el programa abre el archivo, lee la matriz de aprendizaje y la despliega en la pantalla.

ELABORACIÓN DE LA MATRIZ BÁSICA

Después que se ha llevado a cabo la lectura de la Matriz de Aprendizaje, el sistema solicitará al usuario que introduzca los criterios de comparación de cada uno de los rasgos para construir la matriz Básica. Existen básicamente 2 tipos de criterios:

1. De igualdad: Considerará los rasgos semejantes cuando sean estrictamente iguales.
2. Con un Épsilon: Considerará iguales aquellos valores de rasgos que difieran a lo más en Épsilon.

En los rasgos que sean boleados, solo tiene sentido utilizar el primer criterio. Los rasgos de tipo k-valente y real, tienen la opción de elegir cualquiera de los dos criterios anteriores. Esta información será capturada por el programa mediante la pantalla mostrada en la fig. 2. y se espera que el usuario introduzca únicamente 1 ó 2, en función del criterio que desea (1 para el criterio de igualdad estricta y 2 para el criterio de igualdad en términos de una distancia menor a un Épsilon).

Esta es la única información que el sistema necesita del usuario para llevar a cabo el proceso de construcción de la Matriz de Diferencias y la Matriz de Diferencias y de la Matriz Básica. (Aunque a decir verdad, la Matriz de Diferencias nunca es construida como tal.) . En esta parte, los cálculos y operaciones son transparentes para el usuario.

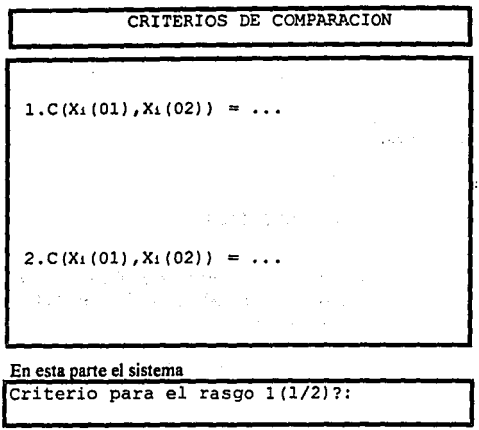


FIG. 2: Pantalla de captura de los criterios de comparación.

Cuando el programa termina de comparar rasgo a rasgo todos los objetos muestra en pantalla la matriz básica obtenida. Nunca se construye como tal la matriz de Diferencias pero se considera como un paso implícito. Esto ahorra tiempo y memoria.

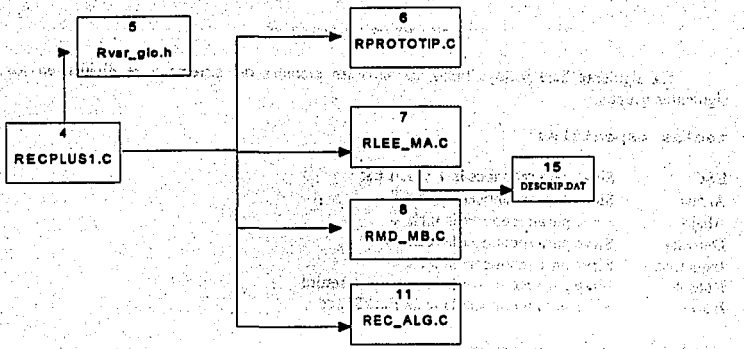
La siguiente pantalla que se despliega es la Matriz Básica formada. Sobre esta matriz, el sistema aplica el algoritmo RECPLUS para localizar los ε-testares, y para ello solicita al usuario el valor de Épsilon. Cuando el usuario desea conocer sólo los testores típicos clásicos, debe introducir 0 como valor de Épsilon.

La ultima pantalla del sistema muestra la lista de candidatos a testores típicos encontrados primero y luego la lista de testores típicos que resultaron serlo.

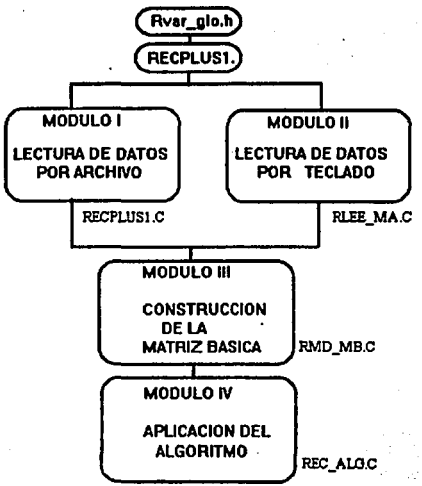
Compare estas dos listas resultantes con la lista de 2^n conjuntos posibles.

MANUAL TÉCNICO:

RECPLUS, esta conformado por 7 archivos básicamente, cuya relación jerárquica se puede apreciar en el siguiente diagrama:



La función de cada uno de los archivos se puede explicitar de la siguiente manera:



VARIABLES GLOBALES

La siguiente lista incluye todas las variables globales del sistema y se dividen en los siguientes grupos:

teclas especiales:

ESC	Sirve para reconoce la TECLA Esc
Arriba	Sirve para reconocer la tecla ↑
Abajo	Sirve para reconocer la tecla ↓
Derecha	Sirve para reconocer la tecla →
Izquierda	Sirve para reconocer la tecla ←
Return	Sirve para reconocer la tecla entre ó return
Borra	Sirve para reconocer la tecla BackSpace

para el manejo de pantalla

MaxX	Máximo numero de columnas de la pantalla
MaxY	Máximo número de renglones de la pantalla
MinX	Primera columna en la pantalla de captura
MinY	Primer renglón en la pantalla de captura
Tot_ren	Total de renglones utilizables en la pantalla de captura
Tot_col	Total de columnas de 5 de ancho utilizables en la pantalla de captura.
blanco	renglón con 75 espacios en blanco
renglón	guarda la anterior posición del renglón del cursor en la pantalla de captura.
columna	guarda la anterior posición de la columna del cursor en la pantalla de captura.

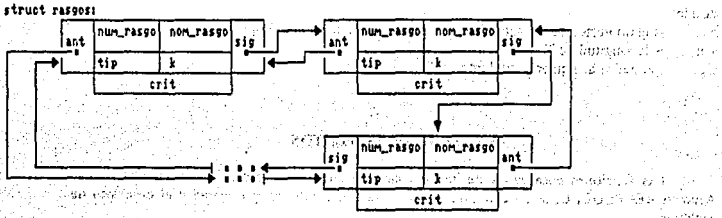
para el manejo de archivos

nom_arch	Identifica el nombre del archivo que contiene o contendrá la Matriz de Aprendizaje.
nom_des	Contiene el nombre (y en ocasiones la dirección) del descripto de archivos.
token	sirve para leer un token de un archivo
token_num	indica el tipo de token que acaba de ser leído.
entrada	apuntador al descripto de archivos.

otros

tot_obj	total de objetos de la Matriz de Aprendizaje
rasgonum	total de rasgos de la Matriz de Aprendizaje
longi	longitud del conjunto de rasgos analizado en turno.

la lista de rasgos se maneja con asignación dinámica de memoria mediante una lista doblemente ligada con la siguiente estructura:

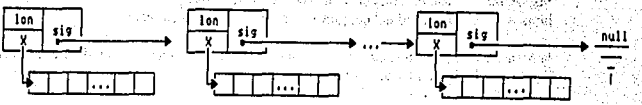


donde:

num_rasgo:	Es el número del rasgo. Guarda números enteros sucesivos de 1 a N,
nom_rasgo:	Es el nombre del rasgo. Cadena de 8 caracteres
tipo:	es el tipo del rasgo.(b=booleano, k=k-valente ó r=real)
k:	la valencia del rasgo. (Si el rasgo no es k-valente se almacena 1).
crit:	guarda el tipo de criterio de comparación. Si el valor es 0, es criterio de comparación es de igualdad. Si el valor es distinto de cero, el criterio de comparación considerará iguales a cualquier par de valores de los rasgos cuya diferencia sea menor que el numero almacenado en este espacio.
ant:	apuntador al rasgo siguiente.
sig:	apuntador al rasgo anterior.

la lista de testores se maneja con una lista simplemente ligada que tiene dentro de su estructura un vector que guarda los datos de los subíndices de los elementos que forman el conjunto X.

struct lista_test:



donde:

X: es el un vector con el subíndice de los elementos del conjunto que intervienen.
lon: es la longitud de X
sig: apuntador al siguiente conjunto

LECTURA DE LOS DATOS

Las funciones encargadas de llevar a cabo la lectura por archivo se encuentran en el Archivo RECPLUS1.C, junto con ellas se localizan las funciones que manejan el descripto de archivos.

Las funciones encargadas de llevar a cabo la lectura por teclado se encuentran en el archivo LEE_MA.C, en esta parte, se localizan entonces el analizador lexicográfico en OBTEN_TOKEN y LEE_MA_TECLA y las funciones para desplegar la pantalla de captura y grabar en los casos necesarios la nueva información en el descripto de archivos.

ELABORACIÓN DE LA MATRIZ BÁSICA

La lectura de los criterios de comparación (LEE_CRITERIO), y la elaboración de la Matriz Básica (CREA_MD) se encuentran junto con sus funciones auxiliares en el archivo RMD_MB.C

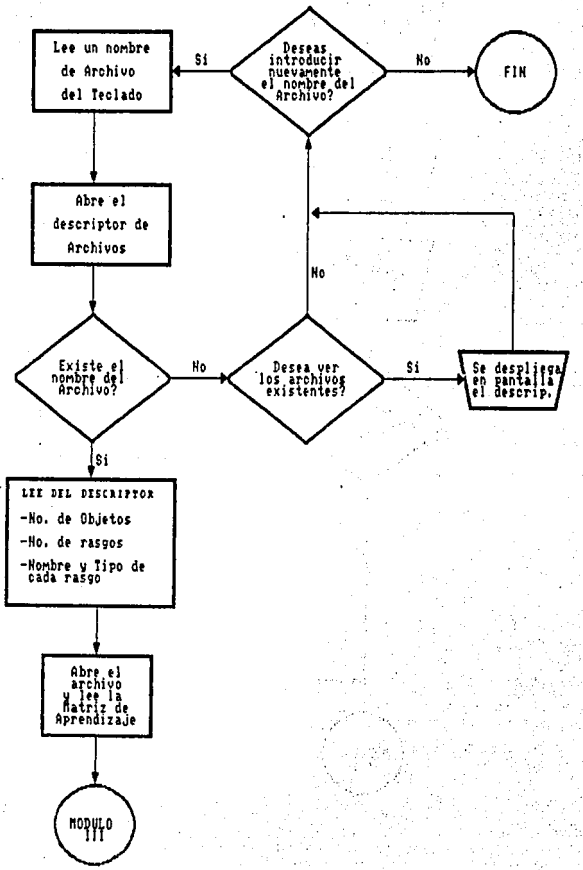
IMPLANTACIÓN DEL ALGORITMO

La implantación del Algoritmo se encuentra en el Archivo REC_ALG.C y se conforma por la función principal ALGORITMO que es la encargada de invocar como funciones auxiliares a SIGMA, BETA y ALFA que son respectivamente las implantaciones de las funciones sigma, beta y alfa.

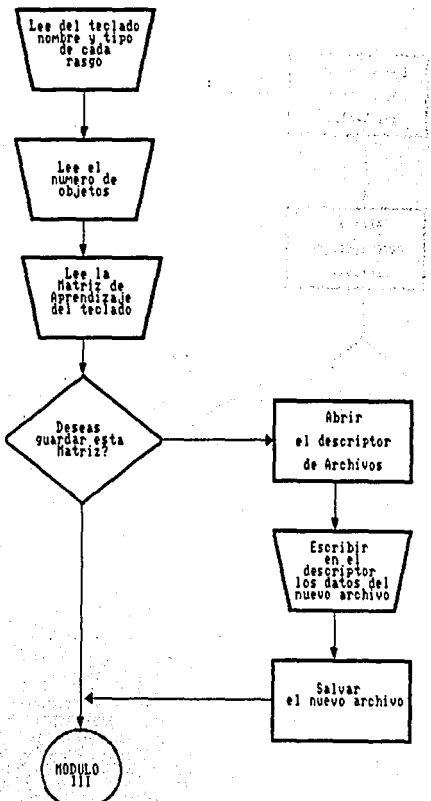
La descripción detallada de los parámetros de entrada y las funciones que realizan todas y cada una de las funciones del sistema se encuentra en el archivo RPROTOTI.C que es el archivo que contiene todos los prototipos de las funciones empleadas.

En las hojas siguientes se muestran los diagramas de flujo correspondientes a los módulos.

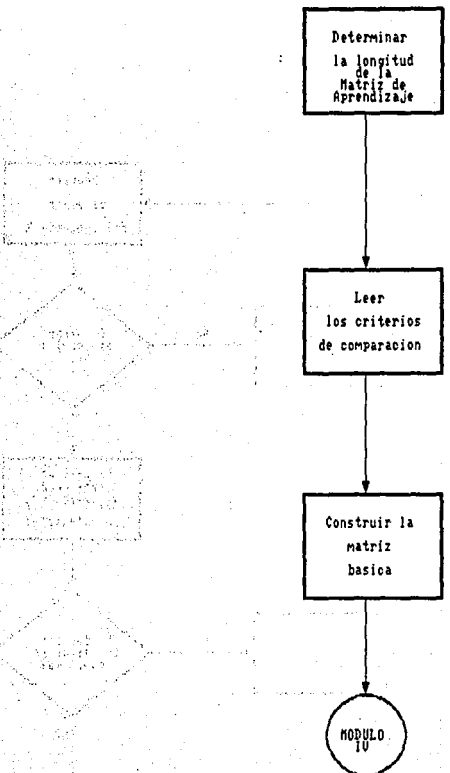
MODULO I. Lectura por Archivo



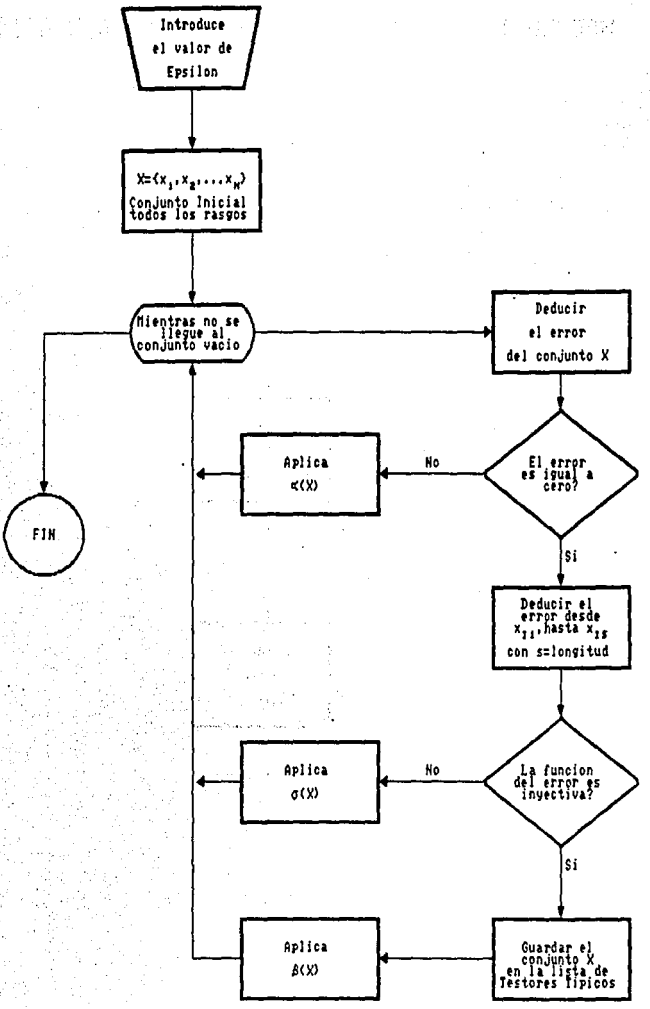
MODULO II. Lectura de Datos por teclado



MODULO III. CONSTRUCCION DE LA MATRIZ BASICA



MODULO IV. APLICACION DEL ALGORITMO.



ANEXO 1
LISTADOS

```
/*************************************************
* Nombre del archivo: RVAR_GLO.H                 *
* DESCRIPCION: define todas las variables globales *
*************************************************/

#define ESC         0X1b
#define Arriba       72
#define Abajo        80
#define Derecha      77
#define Izquierda    75
#define Return       13
#define MaxX         80
#define MaxY         20
#define MinX          6
#define MinY          7
#define Tot_ren      13
#define Tot_col      14
#define Ancho_col     6
#define Entero       sizeof(int)
#define blanco       ' '
#define borra         8

struct rasgos{                  //estos nodos rasgos sirven para la lista ligada
  int num_rasgo;                //que maneja los nombres y tipos de los rasgos
  char nom_rasgo[25];
  char tip;
  char k;
  struct rasgos *ant;
  struct rasgos *sig;
};

struct lista_test{              //Esta lista sirve para manejar la lista de tetores típicos que va localizando el algoritmo
  int *X;
  int lon;
  struct lista_test *sig;
};

typedef char CADENA[10];
typedef struct lista_test NODO_TEST;
typedef struct rasgos NODO_RASGO;

NODO_RASGO *pri_nom_rasg;       //apuntador al primer nodo de la lista de rasgos
CADENA *MA;                      //apuntador a la tabla de la matriz de Aprendizaje
CADENA *MB;                     //apuntador a la tabla de la matriz de Aprendizaje
char nom_arch[25];              //nombre del archivo que continene la MA
char nom_des[30];               //nombre del archivo de descripciones
char token[25];                 //se guarda un token
char buffer[10];
int token_num, indice;          //auxiliares de la funcion obten_token;
int tot_obj;                    //total de objetos
int rasgonum;                   //total de rasgos

FILE *entrada;                  //apuntador al archivo de descripciones primero
                                //y al archivo de la MA despues
int renglon=0, columna=0;       //controlan el renglon y columna en la pantalla
                                //de captura
int longi;
```

```
/*************************************************
* Nombre del archivo : RPROTOTI.C                *
* Descripción: Describe cada una de las funciones y define *
*              los prototipos de cada una de ellas.  *
*************************************************/


/*****************************
 * FUNCION: ERROR
* ENTRADA: cadena, un apuntador a una cadena de caracteres
* DESCRIPCION: Esta funcion es llamada cuando se localiza un error que
*         no permite que continue el programa, se imprime el mensaje recibido
*         en cadena y se corta la ejecucion del programa
*****************************/
void ERROR(char *cadena);

/*****************************
* FUNCION: OBTEN_TOKEN
* ENTRADA: No hay
* DESCRIPCIÓN: Obtiene un token del archivo, utilizando los blancos y teclas
*         especiales como separador
* SALIDA: aux, que es un numero entero que indica el tipo de token que se leyo
*         en el archivo:
*         1 = valor entero              2 = cadena de caracteres
*         3 = un solo punto  4 = * que denota ausencia de informacion
*         5 = return                    0 = ninguno de los anteriores
*****************************/
int OBTEN_TOKEN(void);

/*****************************
* FUNCIÓN: OBTEN_TOKEN_TECLA
* ENTRADA: No hay
* DESCRIPCIÓN: Obtiene un token del teclado, utilizando los blancos y teclas
*         especiales como separador
* SALIDA: aux, que es un numero entero que indica el tipo de token que se leyó
*         en el archivo:
*         1 = valor entero              2 = cadena de caracteres
*         3 = un solo punto  4 = * que denota ausencia de información
*          5 = return                   0 = ninguno de los anteriores
*****************************/
int OBTEN_TOKEN_TECLA(void);

/*****************************
* FUNCIÓN: FOOT
* ENTRADA: caden y caden1, apuntadores a dos cadenas de caracteres
* DESCRIPCIÓN: Imprime las dos cadenas que recibe como entrada en el recuadro
*         inferior de la pantalla, con un beep.
* SALIDA: no hay salida
*****************************/
void FOOT(char *caden, char *caden1);

/*****************************
* FUNCIÓN: CABEZA
* ENTRADA: caden, apuntador a una cadena de caracteres.
* DESCRIPCION: Imprime la cadena que recibe como entrada en el recuadro
*         superior de la pantalla.
* SALIDA: no hay.
*****************************/
void CABEZA(char *caden);

/*****************************
* FUNCIÓN: VENTANA
```

```
* ENTRADA: X1,Y1: Son las coordenas del ángulo superior izquierdo del recuadro
*          X2,Y2: Son las coordenas del ángulo inferior derecho del recuadro
* DESCRIPCION: Imprime un recuadro con las coordenadas que recibió de entrada
*          en la pantalla.
* SALIDA: ninguna
*****************************/
void VENTANA(short X1, short Y1, short X2, short Y2);

/*****************************
* FUNCIÓN: LIMPIA_PANTALLA
* ENTRADA: ppio: numero del primer objeto a imprimir
*          ppior: numero del primer rasgo a imprimir
*          rasgo: apuntador a la lista de rasgos
* DESCRIPCIÓN: limpia la pantalla, imprime tres recuadro (la cabecera, el cuerpo
*          y para los pies de nota), imprime además, si ppio es mayor que cero, un
*          esquema para una matriz de ppio objetos y ppior rasgos.
* SALIDA: ninguna.
*****************************/
void LIMPIA_PANTALLA(int ppio, int ppior, NODO_RASGO *rasgo);

/*****************************
* FUNCIÓN: LECTURA
* ENTRADA: No hay pero debe estar ya definido nom_arch(el nombre del archivo)
* DESCRIPCIÓN: Verifica que se pueda abrir el archivo definido en nom_arch y
*          que no este vacío
* SALIDA: 1  Si pudo abrir el archivo y este no esta vacío
*         0  si no se logro lo anterior.
*****************************/
int LECTURA(void);

/*****************************
* FUNCIÓN: REAL
* ENTRADA: caden apuntador a una cadena de caracteres
* DESCRIPCIÓN: recorre la cadena recibida para verificar que solo tenga
*          valores numérico y a lo mas un punto, es decir, verifica si la
*          cadena de caracteres puede corresponder a un numero real
* SALIDA: 1 si la cadena puede ser considerada como un real
*         0 en caso contrario
*****************************/
int REAL(char caden[]);

/*****************************
* FUNCIÓN: PRUEBA
* ENTRADA: tipo: un carácter que contiene el tipo del rasgo en turno
*          variable: apuntador a la cadena de caracteres que contiene el
*                    valor del rasgo en turno
*          k: un carácter que contiene el valor de k en el caso de rasgos
*                    k-valente, en otro caso contiene 1.
* DESCRIPCIÓN: verifica que el valor introducido para la variable sea valido
*          en dependencia del tipo de la variable. Los tres tipos de rasgos son:
*                    booleano: admite solo 0 o 1
*                    k-valente: admite un entero en el intervalo [0,k]
*                    real: cualquier numero real
* SALIDA: 1 si el valor corresponde con el tipo del rasgo
*         0 en otro caso, pero en general si el valor no corresponde con el
*           tipo del rasgo será solicitado nuevamente al usuario, hasta que
*           el valor sea correcto.
*****************************/
int PRUEBA(char tipo, char *variable, char k);

/*****************************
* FUNCIÓN: PON_EN_MATRIZ
```

```
* ENTRADA: matriz: apuntador a una matriz de caracteres
*          ren y col: ambos enteros que definen la posición en la matriz.
*          valor: vector de caracteres que tiene la información a guardar.
*       REN: es un entero que define el tamaño del renglón de la matriz.
* DESCRIPCIÓN: Coloca en la posición [ren,col] la información de valor, en la
*       matriz cuyo apuntador define "matriz".
* SALIDA: ninguna
*****************************/
void PON_EN_MATRIZ(CADENA *matriz, short ren, short col, char valor[], int REN);

/*****************************
* FUNCIÓN: VALOR_MATRIZ
* ENTRADA: matriz: apuntador a una matriz de caracteres
*          ren y col: ambos enteros que definen la posición en la matriz.
*       REN: es un entero que define el tamaño del renglón de la matriz.
* DESCRIPCIÓN: Accesa la información de la posición [ren,col] de la
*       matriz cuyo apuntador define "matriz".
* SALIDA: un apuntador a la cadena de caracteres que se localizo en la posición
*       [ren,col] de la matriz
*****************************/
CADENA *VALOR_MATRIZ(CADENA *matriz, int ren, int col, int REN);

/*****************************
* FUNCION: INICIA_MA
* ENTRADA: MA: apuntador a una matriz de caracteres
* DESCRIPCIÓN: Inicializa toda la matriz MA con "*" que significa ausencia de
*       información.
* SALIDA: ninguna
*****************************/
void INICIA_MA(CADENA *MA);

/*****************************
* FUNCIÓN: LEE_RASGO_ARCH
* ENTRADA: auxrasgo, que es un apuntador al nodo del rasgo que se esta leyendo
*          en este momento
* DESCRIPCIÓN: es una función auxiliar de LEE_ARCH_DES y sirve para leer del
*       archivo de descripciones el nombre y el tipo del rasgo en turno
* SALIDA:  1 si se logro la lectura sin ningún problema
*          0 si no se logro la lectura de manera correcta
*****************************/
int LEE_RASGO_ARCH(NODO_RASGO *auxrasgo, int l);

/*****************************
* FUNCIÓN: LEE_RASGOS_TECLA
* ENTRADA: rasgo, que es un apuntador al primer nodo de la lista doblemente
*       ligada de rasgos, esta lista llega vacia, solo con un nodo creado
* DESCRIPCIÓN: Recibe los nombres y tipos de cada uno de los rasgos del
*       teclado y con esta información va formando la lista doblemente ligada
*       de rasgos.
*****************************/
void LEE_RASGOS_TECLA(NODO_RASGO *rasgo);

/*****************************
* FUNCIÓN: LEE_ARCH_DESC
* ENTRADA: No hay
* DESCRIPCIÓN: Abre el archivo de descriptos, busca el nombre del archivo
*       introducido por el usuario, y al encontrarlo lee en ese renglón:
*       el numero de objetos para inicializar tot_obj
*       el numero de rasgos para inicializar rasgonum
*       el nombre y tipo de cada uno de los rasgos, mediante la función
*       LEE_RASGO_ARCH
* SALIDA: -NULL si no se pudo hacer la lectura de manera correcta
```

```
*          -un apuntador a la lista de rasgos
*****************************/
NODO_RASGO* LEE_ARCH_DESC(void);

/*****************************
* FUNCIÓN: LEE_MA_ARCH
* ENTRADA: MA: es un apuntador a la tabla de caracteres que contendrá la info
*                de la matriz de aprendizaje
*       list_rasg: es un apuntador al primer nodo de la lista de rasgos
* DESCRIPCIÓN: Lee del archivo dado en nom_arch, la información de la sig.
*          manera: el primer valor de cada renglón lo toma como la clase del
*          objeto descrito en ese renglón, los "rasgonum" token siguientes los
*          toma como los valores de cada uno de los rasgos del objeto. Al mismo
*          tiempo se manda a pantalla estos valores.
*****************************/
void LEE_MA_ARCH(CADENA *MA, NODO_RASGO *list_rasg);

/*****************************
* FUNCION: LEE_MA_TECLA
* ENTRADA: MA: apuntador a una matriz de caracteres
*          rasgo: apuntador a la lista de rasgos.
* DESCRIPCION: Lee los valores de la Matriz de Aprendizaje desde el teclado
* SALIDA: ninguna
*****************************/
void LEE_MA_TECLA(CADENA *MA, NODO_RASGO *rasgo);

/*****************************
* FUNCIÓN: GRABA_DESCRIP
* ENTRADA: l_rasgos: apuntador a la lista de rasgos
* DESCRIPCIÓN: Graba en el descriptor de archivos los atributos del nuevo
*          archivo que contiene la matriz de aprendizaje que se acaba de leer
*          del teclado. Estos atributos son:
*          -nombre del archivo con extensión .DAT
*          -numero de objetos
*          -número de rasgos
*          -la lista de los nombres de rasgos y sus tipos.
* SALIDA: ninguna
*****************************/
void GRABA_DESCRIP(NODO_RASGO *l_rasgos);

/*****************************
* FUNCIÓN: GRABA_DESCRIP
* ENTRADA:MA: apuntador a la matriz de aprendizaje
*          l_rasgos: apuntador a la lista de rasgos
* DESCRIPCIÓN: Graba el nuevo archivo que contiene la matriz de aprendizaje
*          que se acaba de leer
*          del teclado.
* SALIDA: ninguna
*****************************/
void GRABA_ARCHIVO(CADENA *MA, NODO_RASGO *rasgos);

/*****************************
* FUNCIÓN: IMPRIME_DESC
* ENTRADA: No hay
* DESCRIPCIÓN: Se encarga de escribir en pantalla la información que
*          contiene el descriptor de archivos
*****************************/
void IMPRIME_DESC(void);

/*****************************
* FUNCIÓN: LEE_CRITERIO_COMP
* ENTRADA: crit: apuntador a una cadena de enteros, que entra vacía
```

```
*            liminf: apuntador a una cadena de enteros, que entra vacía
* DESCRIPCIÓN: aquí se despliegara una ventana con los criterios de comparación
*        se le pedir  al usuario que introduzca los criterios de comparación
*        para cada rasgo, esta información la ir  colocando en la lista crit
*        y en la lista liminf.
**********************************/
void LEE_CRITERIO_COMP(int *crit, int *liminf);

/**********************************
* FUNCIÓN: SUPERFILA
* ENTRADA: MD: apuntador al renglón de la Matriz de Diferencias.
*       MB: Apuntador a la matriz Básica
*           dimen2: entero que indica el numero de renglones de MB.
* DESCRIPCIÓN: Esta función se encarga de averiguar si el renglón MD es
*          subfila o superfila de alguno de los renglones de MB.
*          En el primer caso introduce el renglón de MD en MB y borra las
*          superfilas de MB.
*          En el segundo caso no la introduce en MB.
* SALIDA: dimen2, ya que si se introdujo la nueva fila y se borraron otras
*          el numero de renglones de MB varia.
**********************************/
int SUPERFILA(CADENA *MD, CADENA *MB, int dimen2);


/**********************************
* FUNCIÓN: CREA_MD
* ENTRADA: MA: Apuntador a la matriz de Aprendizaje
*          MB: Apuntador a la matriz Básica (vacía)
* DESCRIPCIÓN: Llama a la subrutina para leer los criterios de comparación
*          en base a ellos, crea una linea de la Matriz de diferencias cada vez
*          y va creando la Matriz Básica, con las filas básicas.
* SALIDA: dimen2: la dimensión de la Matriz Básica.
**********************************************/
int CREA_MD(CADENA *MA, CADENA *MB);

/**********************************
* FUNCIÓN: PON_RENGLON
* ENTRADA: r1: renglón de MD (siempre es cero) a colocar en MB
*           rMB: renglón de MB donde ser  colocado el nuevo renglón
*       MD: apuntador al renglón de MD.
*           MB: apuntador a la MB.
* DESCRIPCIÓN: coloca el renglón que recibe al final de MB.
**********************************/
void PON_RENGLON(int r1, int rMB, CADENA *MD, CADENA *MB);

/**********************************
* FUNCIÓN: DIMENSION_MB
* ENTRADA: MA : Apuntador a la matriz de aprendizaje
* DESCRIPCIÓN : Cuenta los elementos de cada clase de MA, para deducir el numero
*          de renlones que tendrá  MD.
* SALIDA: contador: indica el numero de renglones de MD.
**********************************/
int DIMENSION_MD(CADENA *MA);

/**********************************
* FUNCIÓN: FILA_CEROS
* ENTRADA: MD: Apuntador a un renglón de la matriz de diferencias
* DESCRIPCIÓN: Verifica que el renglón no este constituido exclusivamente por
*          ceros.
* SALIDA: 0 si el renglón contiene solo ceros
*          1 si por lo menos existe un 1.
**********************************/
```

```
int FILA_CEROS(CADENA *MD);

/**********************************
* FUNCIÓN: PINTA_MATRIZ
* ENTRADA: matriz :apuntador a la matriz en turno
*          ren: numero de renglones
*       col: numero de columnas
*       tipo: 0 : Sin etiquetas en la parte superior
*                1 : con etiquetas en la parte superior
* DESCRIPCIÓN: Despliega en la pantalla la matriz enviada.
* SALIDA: No hay.
**********************************/
void PINTA_MATRIZ(CADENA *matriz, int ren, int col, int tipo);

/**********************************
* FUNCIÓN: ELEMENTO
* ENTRADA: ele : numero del elemento
*          conj2: apuntador a el conjunto
*          lon2: longitud del conjunto (numero de elementos)
* DESCRIPCIÓN: verifica si ele pertenece o no a conj2.
* SALIDA: 0 si ele no pertenece a conj2
*          1 si ele pertenece a conj2
**********************************/
int ELEMENTO(int ele, int *conj2, int lon2);

/**********************************
* FUNCIÓN: SUBCONJUNTO
* ENTRADA: conj1: apuntador al conjunto 1
*       lon1: longitud del conjunto 1
*          conj2: apuntador al conjunto 2
*       lon2: longitud del conjunto 2
* DESCRIPCIÓN: verifica si el conjunto 1 es subconjunto del conjunto 2
* SALIDA: 0 si el conjunto 1 no es subconjunto del conjunto 2
*       1 si el conjunto 1 es subconjunto del conjunto 2
**********************************/
int SUBCONJUNTO(int *conj1, int lon1, int *conj2, int lon2);

/**********************************
* FUNCIÓN: BORRA
* ENTRADA: list_test: apuntador a la lista de testores
* DESCRIPCIÓN: libera la memoria de un nodo de la lista de testores
* SALIDA: Null: el nodo libre
**********************************/
NODO_TEST *BORRA(NODO_TEST *list_test);

/**********************************
* FUNCIÓN: GUARDA
* ENTRADA: X: apuntador a una lista de enteros que simula el conjunto X
*          lista: apuntador a la lista de testores
* DESCRIPCIÓN: guarda el conjunto X en la lista de testores lista.
* SALIDA: lista: el apuntador a la lista de testores.
**********************************/
NODO_TEST* GUARDA(int *X, NODO_TEST *lista);

/**********************************
* FUNCIÓN: ERROR_X
* ENTRADA: MB: Apuntador a MB.
*          dimen: numero de renglones de MB
*       X: apuntador a la lista de enteros que simulan al conjunto X
*          longitud: longitud del conjunto X
*          Epsilon: el numero de rasgos distintos que se permite para considerar
*                a dos objetos iguales.
```

```
* DESCRIPCIÓN: Verifica sobre MB el numero de renglones que no tienen por lo menos
*          Epsilon+1 unos, en la submatriz definida por X.
* SALIDA: El numero de renglones que no son distinguibles bajo el conjunto X.
**********************************/
int ERROR_X(CADENA *MB, int dimen, int *X, int longitud, int Epsilon);

/**********************************
* FUNCIÓN: INYECTIVA
* ENTRADA: error: lista de errores del conjunto X
*          X: apuntador al conjunto X
* DESCRIPCIÓN: Verifica en cuales variables de X el conjunto no es inyectivo
*          y marca con -1 esos valores.
* SALIDA: 0 si X tiene un error no inyectivo
*          1 si X tiene un error inyectivo.
**********************************/
int INYECTIVA(int *error, int *X);

/**********************************
* FUNCIÓN: BETA
* ENTRADA: X: apuntador al conjunto X.
* DESCRIPCIÓN: aplica sobre X la función Beta.
* SALIDA: longi: la nueva longitud del conjunto.
**********************************/
int BETA(int *X);

/**********************************
* FUNCIÓN: SIGMA
* ENTRADA: X: Apuntador al conjunto X
*          error: Lista de errores que comete X
* DESCRIPCIÓN: aplica sobre X la función sigma, eliminando los rasgos que
*          no son inyectivos.
* SALIDA: longi: la nueva longitud del conjunto
**********************************/
int SIGMA(int *X, int *error);

/**********************************
* FUNCIÓN: ALFA
* ENTRADA: X: apuntador al conjunto X.
* DESCRIPCIÓN: aplica al conjunto X la función Alfa.
* SALIDA: longi: la nueva longitud del conjunto.
**********************************/
int ALFA(int *X);

/**********************************
* FUNCIÓN: ALGORITMO
* ENTRADA: MB: Apuntador a la matriz Básica
*          dimen: numero de renglones de la matriz Básica
* DESCRIPCIÓN: Aplica sobre MB el algoritmo RECPlus.
* SALIDA: Como tal no existe salida, pero en el transcurso del desarrollo de
*          esta función se irán imprimiendo el conjunto de testores típicos de
*          la matriz MB.
**********************************/
void ALGORITMO(CADENA *MB, int dimen);
```

```
/*****************************************************
* nombre del archivo:RECPLUS1.C                      *
* Descripción: Contine el programa principal y lleva a cabo *
*        la lectura por archivo de los datos         *
*****************************************************/

#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>
#include <alloc.h>
#include <dos.h>
#include "rvar_glo.h"
#include "rprototi.c"
#include "RLEE_MA.C"
#include "RMD_MB.C"
#include "rec_alg.c"

void ERROR(char *cadena){
 gotoxy(5,22);printf("%s \n", cadena);
 exit(1);
}

int OBTEN_TOKEN(){
char carac;
int aux, indice =1;

 if(token_num){
   strcpy(token, buffer);
   buffer[0] = '\0';
   aux = token_num;
   token_num = 0;
   return(aux);
 }
 else{
   while( (carac = getc(entrada) ) == ' ' || carac == '\t')
                ;

        if(carac >= '0' && carac <= '9'){
          aux = 1;
          token[0] = carac;
          while( (carac = getc(entrada)) >= '0' && carac <= '9' || carac == '.')
            token[indice++] = carac;
          token[indice] = '\0';
          ungetc(carac, entrada);
        }
        else
         if( (carac >= 'A' && carac <= 'Z') || (carac >= 'a' && carac <= 'z') ){
           aux = 2;
           token[0] = carac;
           while( (carac = getc(entrada) ) != ' ' && carac != '\t' && carac != '\n'&& !feof(entrada))
                 token[indice++] = carac;
           token[indice] = '\0';
           ungetc(carac, entrada);
         }
         else if( carac == '.') aux = 3;
              else
                   if(carac == '*'){
                         aux = 4;
```

```
                        token[0] = carac;
                        token[1] = '\0';
                    }
                else
                    if(carac == '\n') aux = 5;
                    else aux = 0;
    return(aux);
  }
}


int LECTURA(void){
  char micar;

  if( (entrada = fopen(nom_arch, "r") ) == 0){
    printf("No se puede abrir  el archivo: %s", nom_arch);
    return(0);
  }
  else{
    micar = getc(entrada);
    if(!feof(entrada) ){
          ungetc(micar, entrada);
          return(1);
    }
    else{
          printf("\n El archivo: %s esta vac;o", nom_arch);
          return(0);
    }
  }
}

void LEE_RASGOS_TECLA(NODO_RASGO *rasgo){
NODO_RASGO *otro, *auxrasgo;
char tip, k, mira;
int indice;

  FOOT("Para terminar oprima enter","");
  gotoxy(MinX,MinY+2);
  printf("Dame el nombre de rasgo %d ", ++rasgonum);
  indice = 0;
  while((mira = getche()) != '\r') token[indice++] = mira;
  token[indice] = '\0';
  if(token[0] != '\0'){
    strcpy(rasgo->nom_rasgo, token);
    do{
     gotoxy(MinX, MinY+3);
     printf(" el tipo del rasgo %d [ ( b ) booleano  ( r ) real  ( k ) kvalente ]: ",rasgonum);
     flushall();
     scanf("%c", &tip);
     tip = tolower(tip);
     if(tip == 'k'){
           gotoxy(MinX,MinY+5);
           printf("valor de  k: ");
           flushall();
           scanf("%c",&k);
           rasgo->k = k;
           gotoxy(MinX,MinY+5); printf("%s",blanco);
     } else
            rasgo->k = '1';
    }while( tip != 'b' && tip != 'r' && tip != 'k');
    rasgo->num_rasgo = rasgonum;
    rasgo->tip = tip;
```

```
    rasgo->ant= NULL;
    token[0] = '\0';
    auxrasgo = rasgo;
   }
   else{
    free(rasgo);
    ERROR("No hay informacion sobre los rasgos");
   }
   do{
    gotoxy(MinX, MinY+2); printf("%s", blanco);
    gotoxy(MinX, MinY+3); printf("%s",blanco);
    gotoxy(MinX, MinY+2); printf("Dame el nombre de rasgo %d ", ++rasgonum);
    indice = 0;
    while((mira = getche()) != '\r') token[indice++] = mira;
    token[indice] = '\0';
    if(token[0] != '\0'){
     otro = (NODO_RASGO *) malloc (sizeof(NODO_RASGO));
     strcpy(otro->nom_rasgo, token);
     do{
           gotoxy(MinX, MinY+3);
           printf(" el tipo del rasgo %d [ ( b ) booleano  ( r ) real  ( k ) kvalente ]: ",rasgonum);
           flushall();
           scanf("%c", &tip);
           tip = tolower(tip);
           if(tip == 'k'){
            gotoxy(MinX,MinY+5);
            printf("valor de  k: ");
            flushall();
            scanf("%c",&k);
            otro->k = k;
            gotoxy(MinX,MinY+5); printf("%s",blanco);
           } else
           otro->k = '1';
     }while( tip != 'b' && tip != 'r' && tip != 'k');
     otro->num_rasgo = rasgonum;
     strcpy(otro->nom_rasgo, token);
     otro->tip = tip;
     token[0] = '\0';
     auxrasgo->sig = otro;
     otro->ant = auxrasgo;
     auxrasgo = otro;
    }
    else{
     auxrasgo->sig = rasgo;
     rasgo->ant = auxrasgo;
    }
   }while(!rasgo->ant);
   --rasgonum;
}

int LEE_RASGO_ARCH(NODO_RASGO *auxrasgo, int i){

   strcpy(auxrasgo->nom_rasgo, token);
   auxrasgo->num_rasgo = i+1;
   if(OBTEN_TOKEN() == 2){
     auxrasgo->tip = token[0];
     if(token[0] == 'k')
           if(OBTEN_TOKEN() == 1)
             auxrasgo->k = token[0];
           else{
             FOOT(blanco, blanco);
             gotoxy(3,21);
```

```
            printf("El valor de k no es un valor nÚmerico en el rasgo %d",i);
            return(0);
          }
    else
          auxrasgo->k = 'l';
    return(1);
  }
  else{
    printf("\nHay un error en el tipo del rasgo %d en el descriptor",i);
    return(0);
  }
}

NODO_RASGO* LEE_ARCH_DESC(){
int i;
char mira;
NODO_RASGO *rasgo, *auxrasgo, *otro;

  gotoxy(6,10);
  printf("Nombre del archivo:");
  scanf("%s",nom_arch);
  strcpy(nom_des, "descrip.dat");
  if( (entrada = fopen(nom_des, "r+") ) == NULL){
    FOOT("No se puede abrir el descriptor de archivos",blanco);
    exit(1);
  }
  while(!feof(entrada) && OBTEN_TOKEN() == 2){
    if(strcmp(token, nom_arch) == 0){
          if(OBTEN_TOKEN() == 1)
            tot_obj = atoi(token);
          else{
            FOOT("Hay un error en el numero de objetos del archivo en el descriptor",blanco);
            return(NULL);
          }
          if(OBTEN_TOKEN() == 1)
            rasgonum = atoi(token);
          else{
            FOOT("Hay un error en el num de rasgos del archivo  en el descriptor", blanco);
            return(NULL);
          }
          rasgo = (NODO_RASGO *) malloc (sizeof(NODO_RASGO));
          auxrasgo = rasgo;
          for(i = 0; i< (rasgonum-1); i++){
            if(OBTEN_TOKEN() == 2){
              if(!LEE_RASGO_ARCH(auxrasgo, i)) return(NULL);
              otro = (NODO_RASGO *) malloc (sizeof(NODO_RASGO));
              auxrasgo->sig = otro;
              otro->ant = auxrasgo;
              auxrasgo = otro;
            }
            else{
              FOOT("\n los nombres de los rasgos deben comenzar con una letra"," ");
              printf("el nombre del rasgo %d no cumple esta restricción",i);
              return(NULL);
            }
          }
          if(OBTEN_TOKEN() == 2){
            if(!LEE_RASGO_ARCH(auxrasgo, i)) return(NULL);
            auxrasgo->sig = rasgo;
            rasgo->ant = auxrasgo;
            return(rasgo);
          }
```

```
        else{
          FOOT("El nombre del último rasgo no es valido, dado que no comienza", "    con una letra");
          return(NULL);
        }
   } /*del if del strcmp */
   else do{
          mira =getc(entrada);
          if(feof(entrada)) break;
          ungetc(mira, entrada);
        }while(OBTEN_TOKEN() != 5);
} /*del primer while */
return(NULL); /* no se encontro el nombre del archivo en el descriptor */
}

int REAL(char caden[]){
char *c;
int punto;

 c = caden;
 punto = 0;
 while(*c){
   if(*c >= '0' && *c <= '9')
        c++;
   else
        if(*c == '.')
          if(!punto){
            punto = 1;
            c++;
          }
          else{
           FOOT("Hay mas de un punto en el número real! ","introduce nuevamente el valor ");
           return(0);
          }
        else{
          FOOT("El número tiene caracteres no numericos", "introduce nuevamente el valor ");
          return(0);
        }
 }
 return(1);
}

int PRUEBA(char tipo, char *variable, char k){
int lo,c;

  switch(tipo){
  case 'b':while(1){
             if(!strcmp(variable,"1") || !strcmp(variable,"0") || !strcmp(variable,"")){
                  return(1);
             }
             else{
                  FOOT("El rasgo es de tipo booleano asi que sólo puede tomar valor 0 ó 1 "," Dame el nuevo valor del rasgo: ");
                  lo = strlen(token);
                  gotoxy(columna,renglon);
                  for(c = lo; c >= 0; c--)
                    printf(" ");
                  gotoxy(columna,renglon);
                  scanf("%s",variable);
                  strcpy(token, variable);
                  gotoxy(2,22);printf("%s",blanco);
                  gotoxy(2,23);printf("%s",blanco);
             }
           }
```

```
  case 'r': while(1){
                    if(REAL(variable) || *variable == '0')
                      return(1);
                    else{
                      gotoxy(columna, renglon);
                      lo = strlen(token);
                      for(c = lo; c >= 0; c--)
                       printf(" ");
                      gotoxy(columna,renglon);
                      scanf("%s",variable);
                      gotoxy(2,22);printf("%s",blanco);
                      gotoxy(2,23);printf("%s",blanco);
                    }
                }
  case 'k':while(1){
              if((!(strcmp(variable,"0")<0) && *variable <= k) || *variable == '0')
                  return(1);
              else{
                  gotoxy(2,22);printf(" El valor de la variable debe estar en el rango [0,%c]",k);
                  FOOT(" "," Dame el nuevo valor del rasgo ");
                  lo = strlen(token);
                  gotoxy(columna,renglon);
                  for(c = lo; c >= 0; c--)
                    printf(" ");
                  gotoxy(columna,renglon);
                  scanf("%s",variable);
                  gotoxy(2,22);printf("%s",blanco);
                  gotoxy(2,23);printf("%s",blanco);
              }
            }
  }
  printf("\n algo raro paso");
  return(0);
}

void PON_EN_MATRIZ(CADENA *matriz, short ren, short col, char valor[], int REN){
  strcpy(*(matriz + (ren * (REN+1)) + col), valor);
}

CADENA *VALOR_MATRIZ(CADENA *matriz, int ren, int col, int REN){
  return(*(matriz + (ren * (REN+1)) + col));
  }

void LEE_MA_ARCH(CADENA *MA, NODO_RASGO *list_rasg){
int ren, col;
char mira;

  LIMPIA_PANTALLA(-1,1, NULL);
  CABEZA(" MATRIZ DE APRENDIZAJE");
  for(ren = 0; ren < tot_obj; ren++){
    if( (mira = getc(entrada)) != '\n') ungetc(mira,entrada);
    if(OBTEN_TOKEN() == 1){
          PON_EN_MATRIZ(MA, ren, 0, token, rasgonum);
          for(col = 1; col <= rasgonum; col++){
            OBTEN_TOKEN();
            while(list_rasg->num_rasgo != col)
              list_rasg = list_rasg->sig;
            if(PRUEBA(list_rasg->tip, token,list_rasg->k))
              PON_EN_MATRIZ(MA, ren, col, token, rasgonum);
            else break;
          }
    }
```

```
    else
      FOOT("\n Hay un error en el nÚmero de clase del objeto siguiente","");
  }
  PINTA_MATRIZ(MA, tot_obj, rasgonum, 1);
}

void IMPRIME_DESC(){
int i, num;
char mira;

  LIMPIA_PANTALLA(-1,1, NULL);
  CABEZA( "      DESCRIPTOR DE ARCHIVOS:");
  rewind(entrada);
  while(!feof(entrada)){
   if( (mira = getc(entrada)) != '\n') ungetc(mira, entrada);
   OBTEN_TOKEN();
   gotoxy(MinX,MinY-2); printf("Nombre del archivo: %12.12s",token);
   OBTEN_TOKEN(),
   gotoxy(MinX,MinY - 1);printf(" numero de objetos:%12.12s",token);
   OBTEN_TOKEN();
   gotoxy(MinX,MinY);printf(" numero de rasgos: %12.12s",token);
   gotoxy(MinX,MinY + 1);printf("=========================================");
   gotoxy(MinX,MinY + 2);printf("NOMBRE DEL RASGO   TIPO     K= ");
   gotoxy(MinX,MinY + 3);printf("=========================================");
   num = atoi(token);
   flushall();
   for(i = 0; i < num; i++){
         OBTEN_TOKEN();
         gotoxy(MinX-2,MinY + 5+i);printf("%d.-:   %s \t ",i+1,token);
         OBTEN_TOKEN();
         printf(" %s \t",token);
        if(tolower(token[0]) == 'k'){
          OBTEN_TOKEN();
          printf("\t con k = %s", token);
        }
//    FOOT("PRESIONE ENTER PARA CONTINUAR CON OTRO RASGO", blanco);
//    getch();
//    gotoxy(3,22); printf("%s",blanco);
   }
   FOOT("PRESIONE ENTER PARA CONTINUAR CON OTRO ARCHIVO", blanco);
   getch();
   gotoxy(3,22); printf("%s",blanco);
   for(i = 0; i < num; i++){
         gotoxy(MinX-2,MinY+5+i);printf("%s",blanco);
   }
  }
  LIMPIA_PANTALLA(-1,1,NULL);
}

/*****************************************
*              MAIN                      *
*****************************************/
void main(){
 char resp, res,lec;
 int dimen, dimen2;

 fclose(entrada);
 resp = 's';
 clrscr();
 LIMPIA_PANTALLA(-1,-1,0);
 CABEZA("        INTRODUCCION DE LOS RASGOS          ");
 gotoxy(6,6);
```

```
printf("la introduccion de los rasgos ser  por: (t)teclado (a)archivo \t");
scanf("%c", &lec);
if(tolower(lec) == 't'){
  pri_nom_rasg = (NODO_RASGO *) malloc (sizeof(NODO_RASGO));
  LEE_RASGOS_TECLA(pri_nom_rasg);
  gotoxy(6,12);
  printf(" Numero de objetos : ");
  scanf("%d",&tot_obj);
  MA = (CADENA *) malloc ( tot_obj * (rasgonum + 1) * (sizeof(CADENA)));
  LEE_MA_TECLA(MA, pri_nom_rasg);
}
else
  if(tolower(lec) == 'a'){
    do{
          if((pri_nom_rasg = LEE_ARCH_DESC()) != NULL){
           fclose(entrada);
           if(LECTURA()){
                MA = (CADENA *) malloc (tot_obj * (rasgonum + 1) * (sizeof (CADENA)));
                LEE_MA_ARCH(MA, pri_nom_rasg);
                resp = 'n';
           }
           else exit(1);
          }
          else{
           FOOT("El nombre del archivo no existe en el descriptor","Deseas ver los archivos disponibles?");
           flushall();
           scanf("%c", &res);
           if(tolower(res) == 's')
                IMPRIME_DESC();
           FOOT("Quieres introducir nuevamente el nombre del archivo?",blanco);
           flushall();
           scanf("%c",&resp);
           if(tolower(resp) == 'n')
                exit(1);
          }
    }while(tolower(resp) == 's');
  }
dimen = DIMENSION_MD(MA);
MB = (CADENA *) malloc (dimen * rasgonum * (sizeof(CADENA)));
dimen = CREA_MD(MA, MB);
ALGORITMO(MB, dimen);
free(MA);
free(MB);
}
```

```
/*********************************************
*Nombre del archivo: RMD_MB.C                *
*Descripción: lee los criterios de comparación y *
*            crea la Matriz Básica           *
*********************************************/


void LEE_CRITERIO_COMP(int *crit, int *liminf){
 int ras;

 CABEZA(" CRITERIOS DE COMPARACION ");
                        gotoxy(28,5); printf("È");
                        gotoxy(28,6); printf("° 1 si Xi(O1) = Xi(O2)");
 gotoxy(MinX-1,MinY); printf("1.- C(Xi(O1),Xi(O2)) = ");
                        gotoxy(28,8); printf("° 0 en otro caso");
                        gotoxy(28,9); printf("È ");
                        gotoxy(28,15);printf("È");
                        gotoxy(28,16);printf("° 1 si (Xi(O1) - Xi(O2)) < %c ",225);
 gotoxy(MinX-1,17);printf("2.- C(Xi(O1),Xi(O2)) = ");
                        gotoxy(28,18);printf("° 0 en otro caso");
                        gotoxy(28,19);printf("È ");
 ras = 0;
 while(ras < rasgonum){
   gotoxy(2,22); printf(" criterio para el rasgo %d (1/2):", ras+1);
   flushall();
   scanf("%d",&crit[ras]);
    if(crit[ras] == 2){
          gotoxy(2,23);printf("valor de %c",238);
          scanf("%d",&liminf[ras]);
    }
   ras++;
   gotoxy(2,22);printf("%s",blanco);
   gotoxy(2,23);printf("%s",blanco);
   }
   LIMPIA_PANTALLA(-1,0, NULL);
}



void BORRA_RENGLON(CADENA *MB, int i, int dimen2){
int ren, ras;

  for( ren = i; ren < dimen2-1; ren++)
    for(ras = 0; ras < rasgonum; ras++)
      PON_EN_MATRIZ(MB,ren, ras, *VALOR_MATRIZ(MB, ren+1, ras, rasgonum-1), rasgonum-1);
}



void PON_RENGLON(int r1, int rMB, CADENA *MD, CADENA *MB){
 int ras;

   for(ras = 0; ras < rasgonum; ras++)
     PON_EN_MATRIZ(MB, rMB, ras, *VALOR_MATRIZ(MD, r1, ras,rasgonum-1),rasgonum-1);
}



int SUPERFILA(CADENA *MD, CADENA *MB, int dimen2){
int i,ren, bandera;
```

```
char val1[2], val2[2];

 for(i = 0; i < dimen2; i++){
   ren = 0;
   bandera = 1;
   do{
         strcpy(val1, *VALOR_MATRIZ(MB, i, ren, rasgonum-1));
         strcpy(val2, *VALOR_MATRIZ(MD, 0, ren, rasgonum-1));
         if(!strcmp(val1, "0") && !strcmp(val2, "1")){
           bandera = 0;
           break;
         }
         ren++;
   }while(ren < rasgonum);
   if(bandera){
    BORRA_RENGLON(MB, i, dimen2);
    dimen2--;
    i--;
   }
 }
 for(i = 0; i < dimen2; i++){
   ren = 0;
   bandera = 1;
   do{
         strcpy(val1, *VALOR_MATRIZ(MB, i, ren, rasgonum-1));
         strcpy(val2, *VALOR_MATRIZ(MD, 0, ren, rasgonum-1));
         if(!strcmp(val1, "1") && !strcmp(val2, "0")){
           bandera = 0;
           break;
         }
         ren++;
   }while(ren < rasgonum);
   if(bandera) return(dimen2);
 }
 PON_RENGLON(0, dimen2, MD, MB);
 return(++dimen2);
}


int DIMENSION_MD(CADENA *MA){
int Obj1, Obj2, contador, subcontador;
CADENA num1,num2;

   Obj1 = 0;
   contador = 0;
   do{
     Obj2 = Obj1 + 1;
     subcontador = 1;
     strcpy(num1,*VALOR_MATRIZ(MA, Obj1, 0, rasgonum));
     strcpy(num2,*VALOR_MATRIZ(MA, Obj2, 0, rasgonum));
     while(Obj2 < (tot_obj) && !strcmp(num1,num2)){
           subcontador++;
           Obj2++;
           strcpy(num2,*VALOR_MATRIZ(MA, Obj2, 0, rasgonum));
     }
     Obj1 = Obj2;
     contador += subcontador * ((tot_obj - 1) - (Obj2-1));
   }while(Obj1 < (tot_obj - 1));
   return(contador);
}
```

```
int CREA_MD(CADENA *MA, CADENA *MB){
int *liminf, *limsup;          //apuntadores a los vectores de los limites de los criterios
int *criterio;                 //apuntador al vector de criterios de comparacion
int Obj1;                      //primer objeto;
int Obj2;                      //segundo objeto;
int Obj_aux;                        //auxiliar para encontrar el segundo objeto
int r;                         //contador para los rasgos
int dimen2;                    //dimension de MB
CADENA val1,val2;              //auxiliares del valor en la matriz
CADENA *MD;
int ri, entaux;

 liminf = (int *) malloc (rasgonum * (sizeof(int)));
 criterio = (int *) malloc (rasgonum * (sizeof(int)));
 LEE_CRITERIO_COMP(criterio, liminf);
 MD = (CADENA *) malloc (2 * rasgonum * (sizeof(CADENA)));
 dimen2 = 0;
 for( Obj1 = 0; Obj1 < (tot_obj - 1); Obj1++){
   Obj_aux = Obj1 + 1;
   r = 0;
   strcpy(val1,*VALOR_MATRIZ(MA, Obj1, r, rasgonum));
   strcpy(val2,*VALOR_MATRIZ(MA, Obj_aux, r, rasgonum));
   while(Obj_aux < tot_obj && !strcmp(val1,val2)){
         Obj_aux++;
         strcpy(val2,*VALOR_MATRIZ(MA, Obj_aux, r, rasgonum));
   }
   for(Obj2 = Obj_aux; Obj2 < tot_obj; Obj2++){
         for( r = 1; r <= rasgonum; r++){
           strcpy(val1,*VALOR_MATRIZ(MA, Obj1, r, rasgonum));
           strcpy(val2,*VALOR_MATRIZ(MA, Obj2, r, rasgonum));
           if( val1 == "*" || val2 == "*")
             PON_EN_MATRIZ(MD, 0, r-1, "0", rasgonum-1);
           else
             switch(criterio[r-1]){
                   case 1: if(!strcmp(val1,val2))
                               PON_EN_MATRIZ(MD, 0, r-1, "0",rasgonum-1);
                           else
                               PON_EN_MATRIZ(MD, 0, r-1, "1",rasgonum-1);
                           break;
                   case 2: if(atol(val1)>=liminf[r-1] && atol(val2)<=limsup[r-1])
                               PON_EN_MATRIZ(MD, 0, r-1, "0",rasgonum-1);
                           else
                               PON_EN_MATRIZ(MD, 0, r-1, "1",rasgonum-1);
                           break;
                   case 3: if(atoi(val1)-atoi(val2) <= liminf[r-1])
                               PON_EN_MATRIZ(MD, 0, r-1, "0",rasgonum-1);
                           else
                               PON_EN_MATRIZ(MD, 0, r-1, "1",rasgonum-1);
                           break;
             }
         }
   if(FILA_CEROS(MD))
     dimen2 = SUPERFILA(MD, MB, dimen2);
   }
 }
 CABEZA(" MATRIZ BASICA");
 PINTA_MATRIZ(MB, dimen2, rasgonum-1,0);
 free(criterio);
 return(dimen2);
}
```

```
int FILA_CEROS(CADENA *MD){
int rasgo, ren;

ren = 0;
rasgo = 0;
do{
  if(!strcmp(*VALOR_MATRIZ(MD, ren, rasgo, rasgonum-1),"1"))
    return(1);
  rasgo++;
}while(rasgo < rasgonum);
return(0);
}
```

```
/*****************************************************
* Nombre del archivo: REC_ALG.C                      *
* Descripción: Lleva a cabo la aplicación del Algoritmo *
*****************************************************/

int ERROR_X(CADENA *MB, int dimen, int *X,int longitud, int Epsilon){
int i,j, contador, cuenta_errores=0;

  for(i = 0; i < dimen; i++){
    contador = 0;
    for(j = 0; j <= longitud; j++)
          if(!strcmp(*VALOR_MATRIZ(MB, i, X[j], rasgonum-1),"1"))
              ++contador;
    if(contador <= Epsilon)
          cuenta_errores= cuenta_errores + (Epsilon +1) - contador;
  }
  return(cuenta_errores);
}


int INYECTIVA(int *error, int *X){
int i, valor, res = 1;

  valor = error[X[0]];
  for(i = 1; i <= longi; i++){
    if( !(valor > error[X[i]]) ){
          error[X[i]] = -1;
          res = 0;
    }
    else
          valor = error[X[i]];
  }
  return(res);
}


int BETA(int *X){
 int i;

 i = X[longi] + 1;
 if(X[longi] < rasgonum-1){
   X[longi] = i;
   while( (rasgonum-1) > i++)
     X[++longi] = i;
 }
 else
   if(X[longi] == rasgonum-1)
     longi--;
   else
     ERROR(" El numero de rasgos del conjunto X, es demasiado grande");
 return(longi);
}


int SIGMA(int *X, int *error){
int i, j;

  for(i = 0; i <= longi; i++)
```

```
    if(error[ X[i] ] == -1){
          for( j = i; j < longi; j++)
            X[j] = X[j + 1];
          --longi;
          --i;
    }
  return(longi);
}


int ALFA(int *X){
 int i, j, num;

  if(longi>0){
   if(X[longi] == rasgonum-1){
          i = longi - 1;
          while(X[i] == (X[i + 1] - 1) && i > -1)
            i--;
          if(i < 0) return(-1);
    }
    else
     i = longi;
  }
  else i = 0;
  num = X[i] + 1;
  for(j = i; num < rasgonum; num++, j++)
    X[j] = num;
  j--;
  return(j);
 }


int ELEMENTO(int ele, int *conj2, int lon2){
 int c2;
   for(c2 = 0; c2 < lon2; c2++)
     if(ele == conj2[c2])
           return(1);
   return(0);
}


int SUBCONJUNTO(int *conj1, int lon1, int *conj2, int lon2){
 int c;

   for(c = 0; c < lon1; c++)
     if(!ELEMENTO(conj1[c], conj2, lon2))
           return(0);
   return(1);
}


NODO_TEST *BORRA(NODO_TEST *list_test){
 NODO_TEST *auxlis_sig;

  free(list_test->X);
  free(list_test);
  return(NULL);
}
```

```
NODO_TEST* GUARDA(int *X, NODO_TEST *lista){
NODO_TEST *apunta, *aux, *otro, *auxant=NULL;
int i;

  gotoxy(MinX, renglon++);
  printf(" TESTOR TIPICO %d : { ", renglon-7);
  for(i = 0; i < longi; i++)
    printf("X%d , ", X[i]+1);
  printf(" X%d }",X[i] +1);
  auxant = NULL;
  apunta = lista;
  if(apunta){
   while(apunta){
          aux = apunta->sig;
          if(SUBCONJUNTO(apunta->X, apunta->lon, X, longi+1))
            return(lista);
          else
           if(SUBCONJUNTO(X,longi+1, apunta->X, apunta->lon)){
            if(apunta == lista){
              lista = apunta->sig;
              free(apunta->X);
              free(apunta);
              apunta = NULL;
              aux = lista;
            }
            else
              if(!apunta->sig){
                    auxant->sig = NULL;
                    free(apunta->X);
                    free(apunta);
                    apunta = auxant;
                    aux = NULL;
              }
              else{
                    aux = auxant->sig = apunta->sig;
                    free(apunta->X);
                    free(apunta);
                    apunta = auxant;
              }
          }
          auxant = apunta;
          apunta = aux;
   }
   otro = (NODO_TEST *) malloc (sizeof(NODO_TEST));
   otro->X = (int *) malloc ((longi+1) * sizeof(int));
   if(auxant && !apunta)
          apunta = auxant;
   if(apunta)
          apunta->sig = otro;
   otro->sig = NULL;
   apunta = otro;
   if(!lista)
     lista = otro;
  }
  else{
    lista = (NODO_TEST *) malloc (sizeof(NODO_TEST));
    lista->X = (int *) malloc ((longi+1) * sizeof(NODO_TEST));
    apunta = lista;
    apunta->sig = NULL;
  }
  for(i = 0; i <= longi; i++)
```

```
    apunta->X[i] = X[i];
  apunta->lon = longi+1;
  return(lista);
}


void IMPRIME(NODO_TEST *lista){
NODO_TEST *aux_lista;
int j, lngi;

 gotoxy(MinX, renglon++);
 printf("conjunto final :");
 while(lista){
   aux_lista = lista->sig;
   lngi = lista->lon;
   gotoxy(MinX, renglon++);
   printf(" testor tipico : {");
   for(j = 0; j < lngi-1; j++)
     printf(" X%d ,",lista->X[j]+1);
   printf(" X%d }",lista->X[j]+1);
   lista = BORRA(lista);
   lista = aux_lista;
 }
}



void ALGORITMO(CADENA *MB, int dimen){
 int *Conjunto_X, *error, *otro;
 int i, Epsilon;
 NODO_TEST *lista= NULL;

 CABEZA(" ALGORITMO REC ");
 gotoxy(5,5); printf(" Valor de Epsilon :");
 scanf("%d", &Epsilon);
 renglon = MinY;
 error = (int *)  malloc (rasgonum * sizeof(int));
 longi = rasgonum-1;
 Conjunto_X = (int *)  malloc (rasgonum * sizeof(int));
 for(i = 0; i <= longi; i++)
   Conjunto_X[i] = i;
 while(longi>=0){
   error[Conjunto_X[longi]] = ERROR_X(MB, dimen, Conjunto_X, longi, Epsilon);
   if( error[Conjunto_X[longi]] == 0){
     for(i = 0; i < longi; i++)
           error[Conjunto_X[i]] = ERROR_X(MB, dimen, Conjunto_X, i, Epsilon);
     if(INYECTIVA(error,Conjunto_X)){
           lista = GUARDA(Conjunto_X, lista);
           longi = BETA(Conjunto_X);
     }
     else longi = SIGMA(Conjunto_X, error);
   }
   else
     longi = ALFA(Conjunto_X);
 }
IMPRIME(lista);
free(Conjunto_X);
free(error);
}
```

BIBLIOGRAFIA

1.- Tou, Julius; Gonzalez Rafael.
"Pattern Recognition Principles"
Addison Wesley, Mass. 1974.

2.- Rosenblatt F.
"The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain".
Psychological Rev, 65 No. 6 pp. 386-408. 1958

3.- Ruiz Shulcloper, J; Lazo Cortés, M.
"Modelos Matemáticos para el Reconocimiento de Patrones".
Aportaciones Matemáticas. Serie Textos IMATE-UNAM(en prensa)

4.- Ruiz Shulcloper, José.
"Notas de clase para curso de posgrado"
Facultad de Ciencias UNAM, 1988.

5.- Lazo Cortés, Manuel.
"Modelos basados en la Teoría de Testores para la selección de rasgos y la clasificación supervisada con descripciones no clasicas de los objetos".
Tesis Doctoral de la Universidad de las Villas, Santa Clara, Cuba. 1994

6.- Alba Cabrera, Eduardo; López Reyes, Nancy; Ruiz Shulcloper, José.
"Extensión del Concepto de Testor Típico a partir de la función de Analogía".
Serie Amarilla CINVESTAV-IPN. (en prensa).

7.- Morales Gamboa, Rafael.
"Un sistema de clasificación y Reconocimiento de Patrones".
Tesis de Licenciatura en Matemáticas de la Facultad de

Ciencias de la Universidad Nacional Autónoma de México. 1988.

8.- Duda, R.; Hard, P.E.
"Pattern Classification and Scene Anlisys".
Jhon Wiley & Sons, N.Y. 1973.

9.- Schalkoff, Robert.
"Pattern Recognition statical, structural and neural approaches".
Jhon Wiley & Sons, N.Y. pp.1-30. 1992.

10.-Chapa Vergara, Sergio V.
"Herramientas para consulta y captura basadas en el descriptor de archivos"
Serie Amarilla de Investigación. CINVESTAV. No. 15A.

11.-Birkhoff, Garrett.
"Modern Applied Algebra"
Ed. Jhon Wiley & Sons, N.Y.

12.-Ross, A. Kenneth; Wright, R.B. Charles.
"Matemáticas Discretas"
Ed. Prentice-Hall Hispanoamericana, S.A. 1991.
pp. 224-347.