APÉNDICE

A. CÓDIGOS

A continuación se presentan los códigos de las funciones de las memorias asociativas geométricas. Estos códigos se presentan en sintaxis de MatLab, por ser más entendible matemáticamente y porque es más sencillo de implementar en la práctica.

Clasificación supervisada

La fase de aprendizaje de una memoria asociativa geométrica para el modo de clasificación consiste en construir una híper-esfera óptima que cubra a los patrones de una clase específica y los patrones de las demás clases estén fuera.

La función *separa* construye una esfera de separación donde los patrones dados por la variable *p* están dentro de la híper-esfera y los patrones dados por *q* están fuera de ella. La función se basa en los desarrollos mostrados en la sección **¡Error! No se encuentra el origen de la referencia.**.

```
function [c,r] = separa(p,q)
% [c,r]=separa(p,q)
% Dados los puntos interiores p y exteriores q de una híper-esfera,
% se construye la híper-esfera óptima de separación, en este caso el
% arreglo de puntos exteriores puede ser vacio, por lo que la
% superficie que se genera es una cubierta.
% Se usa la función del toolbox de optimización quadprog.
% por lo que se construyen las matrices h, a y los vectores f, b.
% Se regresan el centro y el radio de la híper-esfera generada
[n,d] = size(p); % n - núm. de patrones en p;
```

```
                % d– dimensión de  los patrones
m = size(q , 1); % m- núm. de patrones en q;
% Se construyen h y f.
h = [q ; p];
h(: , d + 1) = -ones(m + n , 1);
f = -h;
for i = 1 : m + n
    f(i,:) = sum(h(i , 1 : d) .* h(i , 1 : d)) * f(i , :);
end
f = sum(f)';
h = h' * h;
% Se construyen a y b.
a0 = []; % Puntos fuera de la esfera
b0 = []; % Condición para puntos fuera de la esfera
if m > 0
   b0 = 0;
   a0 = [q , -ones(m , 1)];
   for j = 1 : d
       b0 = b0 + q(: , j) .* q(: , j);
   end
   b0 = b0 / 2 – 1.0e-10; %Valor de e
end
a1 = -[p , -ones(n , 1)]; % Puntos dentro de la esfera
b1 = 0; % Condición para puntos dentro de la esfera
for j = 1 : d
    b1 = b1 + p(: , j) .* p(: , j);
end
b1 = b1 / (-2);
a = [a1 ; a0];
b = [b1 ; b0];
% Con los parametros definidos se invoca a quadprog.
s = quadprog(2 * h , f , a , b);
% Se extraen el centro y el radio del círculo optimo.
c = s(1 : d);
r = sqrt(sum(s(1 : d) .* s(1 : d)) – 2 * s(d + 1));
```

La función *EntrenaGam* realiza la fase de entrenamiento de las memorias asociativas geométricas, como se describe en la sección **¡Error! No se encuentra el origen de la referencia.**.

```
function S = EntrenaGAM(P)
% Función Memoria=EntrenaGAM(Set_Patrones)
%
%                | Patron-1, clase |
%                | Patron-2, clase |
% Set_Patrones = |   ...   ,  ...  |
%                | Patron-K, clase |
%
% Patron-i = [Rasgo1, Rasgo2, ... , RasgoN]
%
[M , N] = size(P); % N - Tamaño del patron, M - Numero de Patrones
C = max(P(: , N)); % Obtener el número de clases
N = N - 1;         % Se elimina el atributo Clase del patrón
S = [];
for i = 1 : C %Recorrer para encontrar los patrones de la misma clase
    Ti = [];  % Guarda los patrones que están dentro de la esfera
    To = [];  % Guarda los patrones que están fuera de la esfera
    k = 1;
    j = 1;
    for m = 1 n M            % Recorrer todos los patrones
        if P(m, N + 1) == i  % En caso de encontrar los de la clase i
            Ti(k , :) = P(m , 1 : N);   % Guardarlos en Ti
            k = k + 1;
        else                 % En caso de que no sean de la clase i
            To(j , :) = P(m , 1 : N);   % Guardarlos en To
            j = j + 1;
        end
    end
    [c , r] = separa(Ti, To);  %#ok<NOPRT> % Obtener el centro y el
                               %radio de la esfera de separación
    S(i , :) = [c', -r * r / 2 + c' * c / 2 , 1]; %#ok<AGROW>
                          % Almacena la esfera optima en el arreglo S
end
```

La función *Interior* calcula el producto interior entre dos objetos en álgebra geométrica conforme.

```
function d = Interior(P1 , P2)
% Función int=Interior(Pat1,Pat2)
%
% Aplica el producto interior entre dos conjuntos de vectores
% El resultado es un vector de tamaño max(size(P1,1), size(P2,1))
% Los vectores deben estar en el espacio conforme
%
m1 = size(P1 , 1);  % Número de vectores en P1
m2 = size(P2 , 1);  % Número de vectores en P2
d = zeros(m1 , m2); % Vector de salida

n = size(P1 , 2) - 2; % dimensión de los patrones
for k = 1 : m1
    for j = 1 : m2
        for i = 1 : n
            % Producto interior tradicional entre los vectores
            d(k , j) = d(k , j) + P1(k , i) * P2(j , i);
        end
        % Calcular los valores de las componentes einf y e0
        d(k , j) = d(k , j) - P1(k , n + 1) * P2( j ,n + 2)
                - P1(k , n + 2) * P2(j , n + 1);
    end
end
```

La función *Euclid2Conf* convierte el patrón euclidiano a su correspondiente notación conforme, ver ecuación .

```
function P = Euclid2Conf(p)
% Funcion P = Euclid2Conf(p)
% cambia p a su representación conforme P
%     | p1 |
%     | p2 |
% p = | .. |
%     | pm |
% Los puntos a cambiar a su representación conforme
```

```
% Donde
% p(i) = |r1,r2,...,rn|
%
%     | P1 |
%     | P2 |
% P = | .. |
%     | .. |
%     | Pm |
% Los puntos en representación conforme
% Donde
% P(i) = |r1,r2,...,rn, rn+1, rn+2|
%
[m , n] = size(p);
for i = 1 : m        % Recorre el arreglo de puntos p
    P(i , 1 : n) = p(i , 1 : n); % La parte euclidiana es la misma
    P(i , n + 1) = p(i , 1 : n) * p(i , 1 : n)'; % Para einf
    P(i , n + 2) = 1;        % Para e0
end
```

La función *ClasificaGam* es la encargada de decidir si un patrón dado pertenece a alguna de las clases definidas por las híper-esferas construidas en la fase de entrenamiento, regresa el índice al que pertenece un patrón dado.

```
function c = ClasificaGAM(M , p)
% Función clase=ReconoceGAM(MemoriaAG,PatronEntrada)
% Fase de clasificación de una MAG
%
%            |  S1  |
%            |  S2  |
% MemoriaAG= |  S3  |
%            | ...  |
%            |  Sk  |
%
% Donde: MemoriaAg(i) = [S1, S2, S3, ... Sn+1, Sn+2]
%    PatronEntrada = [P1, P2, P3, ... Pn]
%         n = La dimensión del espacio de trabajo
%
C = size(M , 1);   % C - Numero de clases  % N  - Tamaño del patrón
```

```
N = size(p , 2);
radios = zeros(N , 1); % Inicializa la matriz con radios 0
Mayor = -5000;
c = 0;
%Convierte Patron a su representación conforme
p = Euclid2Conf(p)
clases = [];
%Calcular los radios de las esferas
for i = 1 : C
    radios(i) = M(i, 1 + N) - M(I , 1 : N) * M(i, 1 : N)' / 2;
                      % El radio -r/2 = S4-(P*P)/2
end
% obtener los productos interiores entre las esferas y el patrón

for i = 1 : C
     d = Interior(M(i , :), p);  % Calcula el producto interior entre
                                 % la esfera y el patrón conforme
     if d < 0  % Verifica  que el patrón esté dentro de la esfera
         clases(i) = -Inf;
               % Si no está dentro de la esfera, no utilizarlo
     else
         clases(i) = d + radios(i);
               % #ok<AGROW>  Calcula la distancia hacia el centro de
               % la esfera en signo negativo
     end
     if clases(i) > Mayor
              % Obtiene el valor más cercano al centro de la esfera
            Mayor = clases(i);
                          c=i; % La clase a la que pertenece
     end
end
```


Restauración de patrones

La fase de aprendizaje de una memoria asociativa geométrica para el modo de restauración de patrones consiste en construir una híper-esfera óptima que cubra a

cada uno de los patrones, con la condición de que dos patrones no estén dentro de una misma híper-esfera.

La función *separaR* construye una esfera de separación donde el patrón *p* es el centro de de una híper-esfera óptima con la condición de que los patrones en la variable *q* están fuera de esa híper-esfera. La función se basa en los desarrollos mostrados en la sección **¡Error! No se encuentra el origen de la referencia.**.

```
function [c , r] = separaR(p , q)
% [c,r]=separacion1(p,q)
% Dados el punto interior p y los puntos exteriores q de una híper-
% esfera, se construye la híper-esfera óptima de separación, donde el
% centro de la esfera es, precisamente, el punto p.
% Se usa la función del toolbox de optimización
% quadprog(h,f,a,b,[],[]),
% por lo que se construyen los valores de h, a, f, b.
d = size(p , 2); % d dimensión de los patrones
m = size(q , 1); % m número de patrones en q
p0 = [p ; q];
% Se construyen h y f.
% h = m - Numero total de puntos
h = m + 1;
% f = sum (2(pj.pi) +(pj)^2)
f = 0;
for i = 1 : m + 1
    f = f + 2 * p0(i , :) * p(1 , :)' + p0(i , :) * p0(i , :)';
end
% Se construyen a y b.
% Construcción de a
a = ones(1 , m + 1);
a(1) = 2;           % Para el punto dentro de la esfera
for j = 2 : m + 1   % Para los puntos fuera de la esfera
    a(j) = -2;
end
% Construcción de b
b = ones(1 , m + 1);
b(1) = p(1 , :) * p(1 , :)';
% Para el punto dentro de la esfera
```

```
for j = 2 : m + 1     % Para los puntos fuera de la esfera
    b(j) = q(j-1 ,:) * q(j-1 , :)' - 2* q(j-1,:) * p(1,:)' - 1.0e-10;
end
% Con los parámetros definidos se invoca a quadprog.
s = quadprog(2 * h, f, a' , b);
% Se extraen el centro y el radio de la híper-esfera óptima.
% El resultado es S(n+1), de ahí se extrae el radio, el centro es p
c = p(1 , :);
r = sqrt(p(1 , :) * p(1 , :)' - 2 * s);
```

La función *EntrenaGamR* realiza la fase de entrenamiento de las memorias asociativas geométricas para restauración, como se describe en la sección **¡Error! No se encuentra el origen de la referencia.**.

```
function S = EntrenaGAMR(P)
% Función Memoria=EntrenaGAMR(Set_Patrones)
% Entrenamiento de una MAG para Restauración
%
%                | Patron-1 |
%                | Patron-2 |
% Set_Patrones = |   ...    |
%                |   ...    |
%                | Patron-K |
% Patron-i = [Rasgo1, Rasgo2, ... , RasgoN]
[M , N] = size(P); % N - Tamaño del patrón, M - Numero de Patrones
S = [];
R = [];
for i = 1 : M    % Recorrer todos los patrones a clasificar
    Ti = [];     % Guarda el patrón que está dentro de la esfera
    To = [];     % Guarda los patrones que están fuera de la esfera
    m = 1;
    for j = 1 : M      % Recorrer todos los demás patrones
        if i == j      % En caso de ser el patrón a clasificar
            Ti = P(j , 1 : N);   % Guarda el patrón en Ti
        else          % En caso de que no sean de la clase i
            To(m , : ) = P(j , 1 : N); %#ok<AGROW> % Guardarlos en To
            m = m + 1;
        end
```

```
    end
    [c , r] = separaR(Ti , To); %#ok<NOPRT> % Obtener el centro y el
                                % radio de la esfera de separación
    S4 = (c * c' – r * r) / 2;
    S(i , :) = [c, S4 , 1]; %#ok<AGROW>
end
```

La función *ReconoceGam* es la encargada de restaurar un patrón dado, regresando el centro de la híper-esfera que lo cubre. A diferencia de su contraparte de clasificación, esta función regresa el centro de la híper-esfera como el patrón restaurado.

```
function p = ReconoceGam(M , p)
% Función PatronSalida= ReconoceGam(MemoriaAG,Patronentrada)
% Fase de restauración de una MAG
%
%            |  S1  |
%            |  S2  |
% MemoriaAG= |  S3  |
%            | ...  |
%            |  Sk  |
% Donde: S(i) = [S1, S2, S3, ... Sn+1, Sn+2]
%    Patron = [P1, P2, P3, ... Pn]
%         n = La dimensión del espacio de trabajo
C = size(M , 1);   % C - Numero de clases  % N  - Tamaño del patrón
N = size(p , 2);
radios = zeros(N , 1); % Inicializa la matriz con radios 0
Mayor = -5000;
c = 0;
p = Euclid2Conf(p) %Convierte p a su representación conforme
clases = [];
%Calcular los radios de las esferas
for i = 1 : C
    radios(i) = M(i, 1 + N) - M(I , 1 : N) * M(i, 1 : N)' / 2;
                    % El radio -r/2 = S4-(P*P)/2
end
% obtener los productos interiores entre las esferas y el patrón
for i = 1 : C
     d = Interior(M(i , :), p);  % Calcula el producto interior entre
```

```
                    % la esfera y el patrón conforme
    if d < 0  % Verifica  que el patrón esté dentro de la esfera
        clases(i) = -Inf;
             % Si no está dentro de la esfera, no utilizarlo
    else
        clases(i) = d + radios(i);
             % #ok<AGROW>  Calcula la distancia hacia el centro de
             % la esfera en signo negativo
    end
    if clases(i) > Mayor
            % Obtiene el valor más cercano al centro de la esfera
          Mayor = clases(i);
                      c=i; % La esfera a la que pertenece
    end
end
p = M(c , 1 : N); % Se regresa el centro de la esfera
```

Clasificación no supervisada

El algoritmo de clasificación no supervisada o agrupamiento es una sola función, se basa en el algoritmo presentado en la sección **¡Error! No se encuentra el origen de la referencia.**.

La función *GAM_Cluster* utiliza una memoria asociativa geométrica para agrupar una serie de punto sin etiquetar modificando el algoritmo tradicional de *k*-medias.

```
function [S , j , C] = GAM_cluster(p , K)
% Funcion Clases=GAM_cluster(Puntos,K)
%
% Memorias Asociativas Geométricas con aprendizaje no supervisado,
% utiliza las características de las MAG para obtener la clase más
% cercana a la que pertenecen los puntos y usa un convex hull para
% agrupar los puntos en las agrupaciones correspondientes.
%           | P1 |
%           | P2 |
%  Puntos = | .  |
%           | .  |
```

```
%           | Pm |
% Conjunto de puntos a ser agrupados, donde
%  P(i) = |r1, r2, ..., rn|
% K -> El número de agrupaciones a generar
% Clases -> Los índices de las clases que corresponden a cada punto
%
[m , n] = size(p);
% Seleccionar el valor máximo para los vectores aleatorios
x = max(max(abs(p)));
% Generar k centroides aleatorios:
C = rand(K , n) * x * 2 - x;
% Formar la MAG con los centroides:
M = Euclid2Conf(C);
P = Euclid2Conf(p);
% Determinar la clase a la que pertenecen cada punto de acuerdo a la
% distancia a los centros
for j = 1 : 1000 %Número máximo de iteraciones
    clase = Interior(M , P);
    [temp, c] = max(clase);
    for i = 1 : m
            r(i , 1 : n) = p(i , 1 : n); %#ok<AGROW>
            r(i, n + 1) = c(i); %#ok<AGROW>
    end
    S = EntrenaGAM(r);
    if S == M     % Si hay convergencia
        break;    % Condición para terminar
    else
        M = S;    % Reemplazar los nuevos centroides
    end
end
```