RECONOCIMIENTO DE FORMAS

MEMORÍA DE LAS PRÁCTICAS

David Guillermo Morales Sáez

ÍNDICE

INTRODUCCIÓN

En la siguiente memoria se recogen las pruebas y análisis realizados durante la asignatura para cada una de las seis prácticas que componen el temario tratado.

La finalidad de este documento es la de obtener una visión global del trabajo realizado durante el curso en las sesiones de laboratorio de la asignatura, sin olvidar las bases de conocimiento adquiridas en las clases de teoría.

Finalmente se podrá encontrar a modo de Anexo los códigos en Matlab desarrollados para cada una de las prácticas.

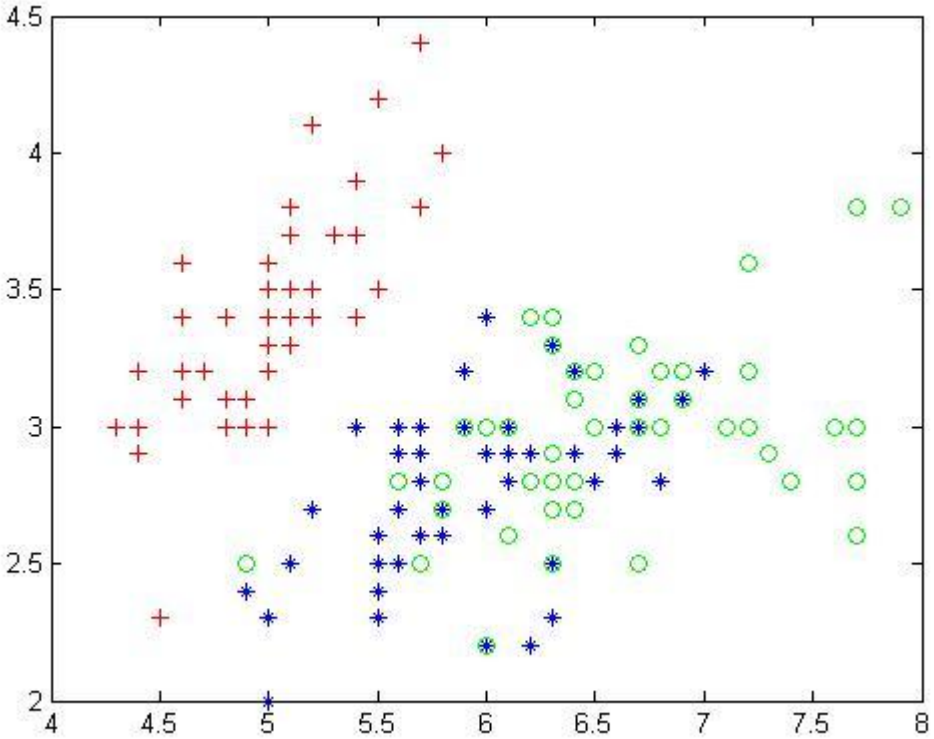
PRÁCTICA 1. Organización y Visualización de Conjuntos de Datos.

El primer paso para poder abordar nuestros objetivos en la asignatura consiste en la lectura de los datos a tratar.

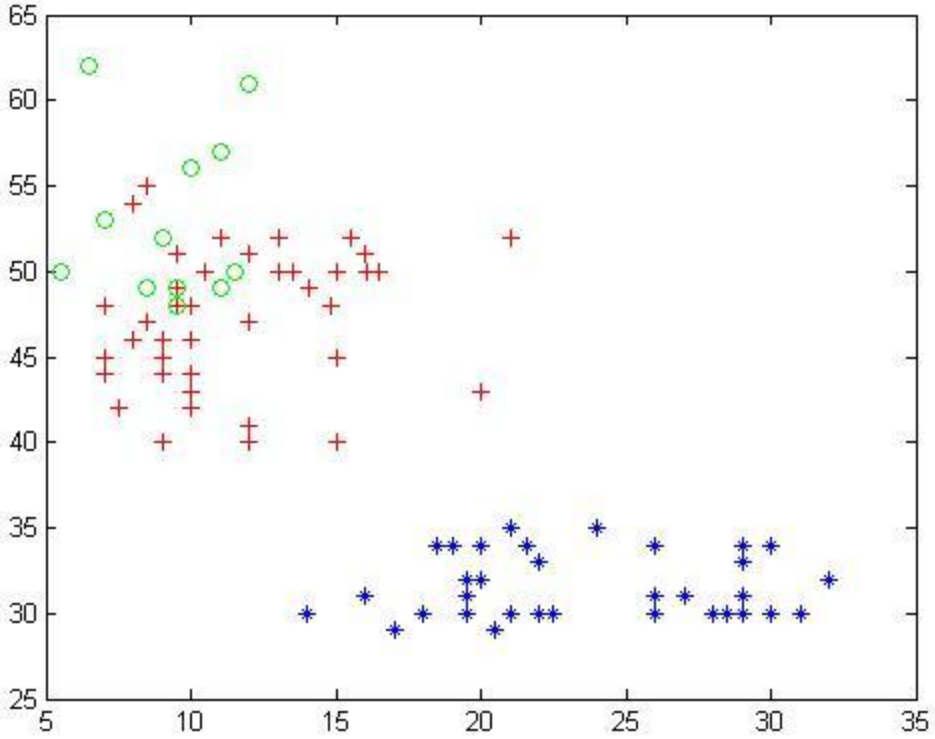
Es esencial organizar y controlar los datos para su correcto procesamiento a efectos de realizar con garantías el futuro análisis y las operaciones que necesitemos para clasificar los mismos.

Para ello construimos una función, *`Practica1`*, encargada de leer y cargar los datos del conjunto de datos que vamos a estudiar.

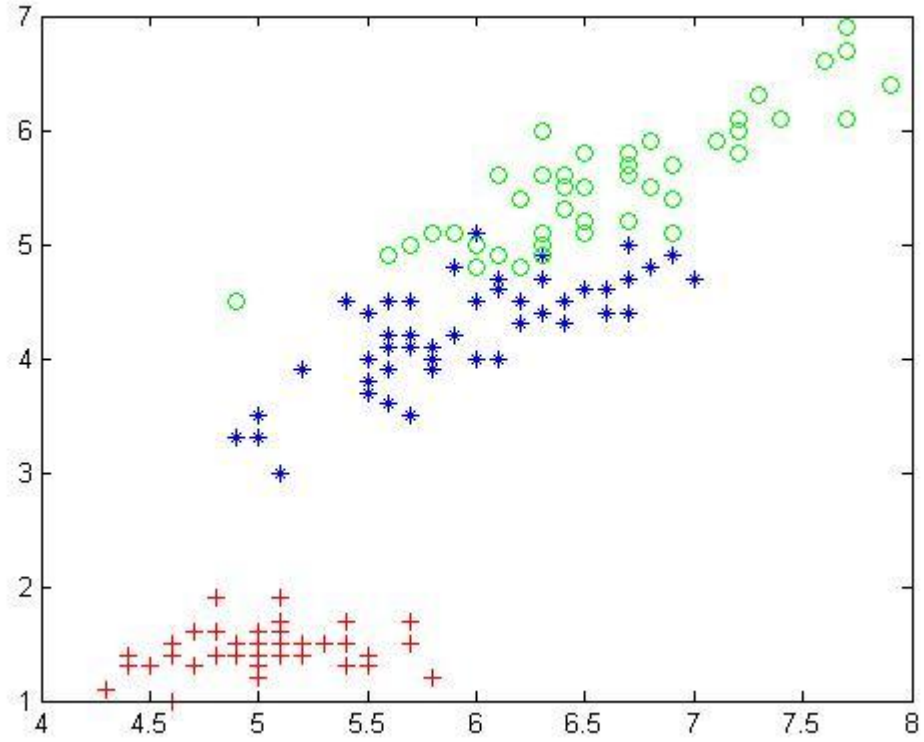
Posteriormente se creó una función, *`Displaydataset`*, para mostrar los datos mediante una representación gráfica en 2D sencilla. A continuación se amplió esta visualización para poder seleccionar las distintas clases de datos en 2D. Por último se desarrollo una función equivalente para visualizar los datos globales en 3D *`Displaydataset3`*. De forma adicional se modificaron los archivos anteriores para mostrar al mismo tiempo dos conjuntos de datos, mediante las funciones *`DisplayDatasetDoble`* y *`DisplayDataset3Doble`*.



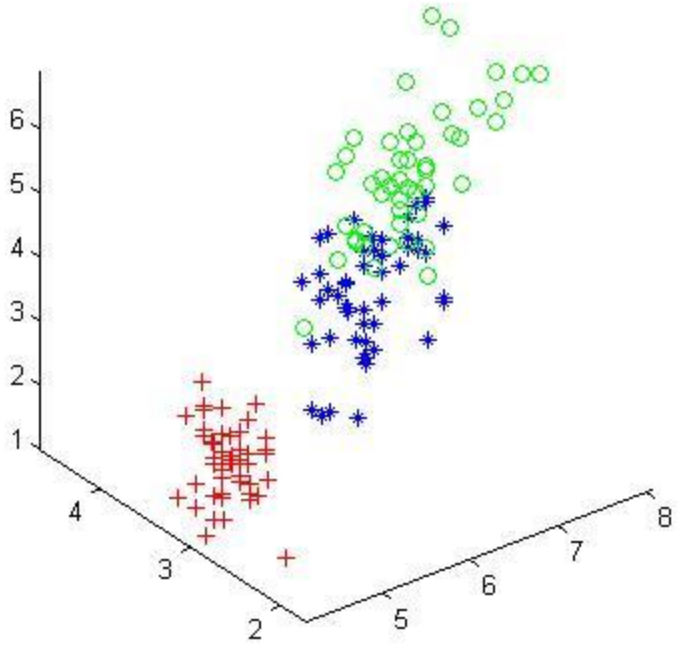
Resultado de DisplayDataSet para iris.dat



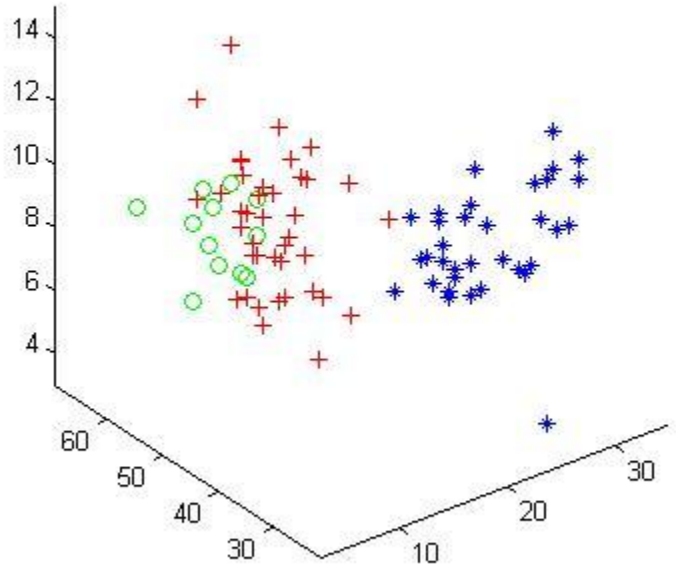
Displaydataset para fosil.dat



Displaydataset2 (1, 3) para iris.dat



Displaydataset3 para iris.dat



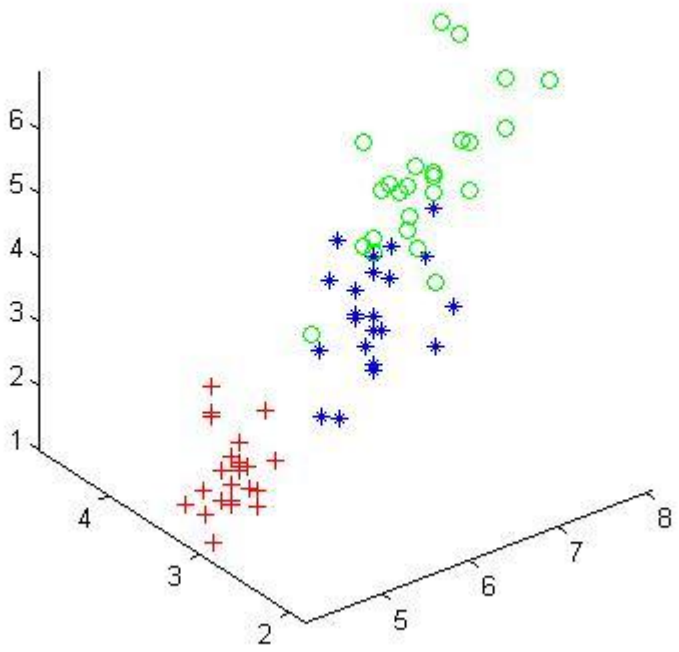
Displaydataset3 para fosil.dat

Tras probar diferentes conjuntos de datos resulta evidente la necesidad e importancia de la visualización de los mismos. El análisis visual, sobre todo en entorno 3D, nos ha mostrado grandes diferencias y una separabilidad real y efectiva que no parecía tan acusada en el análisis 2D y mucho menos en el análisis directo de los datos sin visualizar.

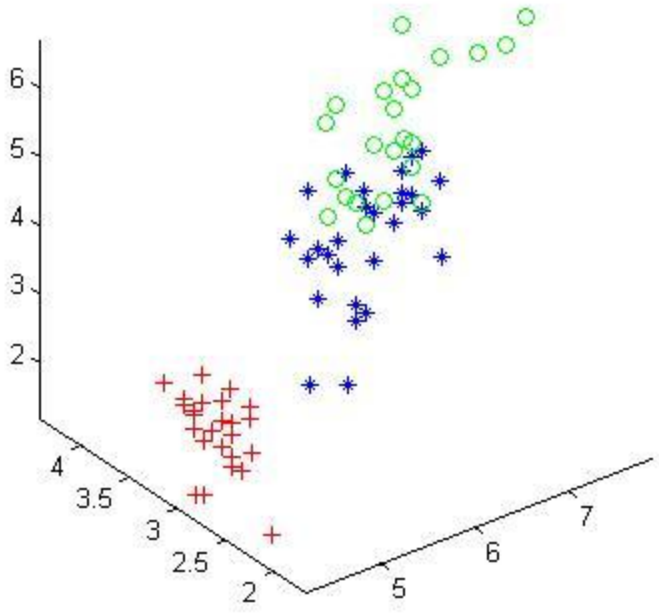
PRÁCTICA 2. Bootstrap, NN, KNN y Test de Conjuntos de Datos.

En esta segunda tarea procederemos al tratamiento de conjuntos de datos como fuentes para la creación y análisis posterior de nuevos conjuntos aleatorios. En este ámbito dispondremos de conjuntos de aprendizaje destinados al entrenamiento del clasificador y conjuntos de test para probar el clasificador entrenado. Podemos así determinar diferencias y obtener una tasa de error de la clasificación realizada.

El primer paso consiste en crear una función, `separador`, que genere el conjunto de entrenamiento en base a un conjunto inicial dado.



Dataset 1 generado por separador para iris.dat



Dataset 1 generado por separador para iris.dat

Tras ello procedemos a crear y probar dos clasificaciones. Según el vecino más próximo, *NN*, su ampliación a los k vecinos más próximos, *KNN*.

Seguidamente implementamos una función de test de clasificación, *Test_KNN* donde probaremos el entrenamiento realizado por *KNN*, para finalmente dar los resultados en *Resultado_KNN*, encargado de representar gráficamente los errores por k de KNN.

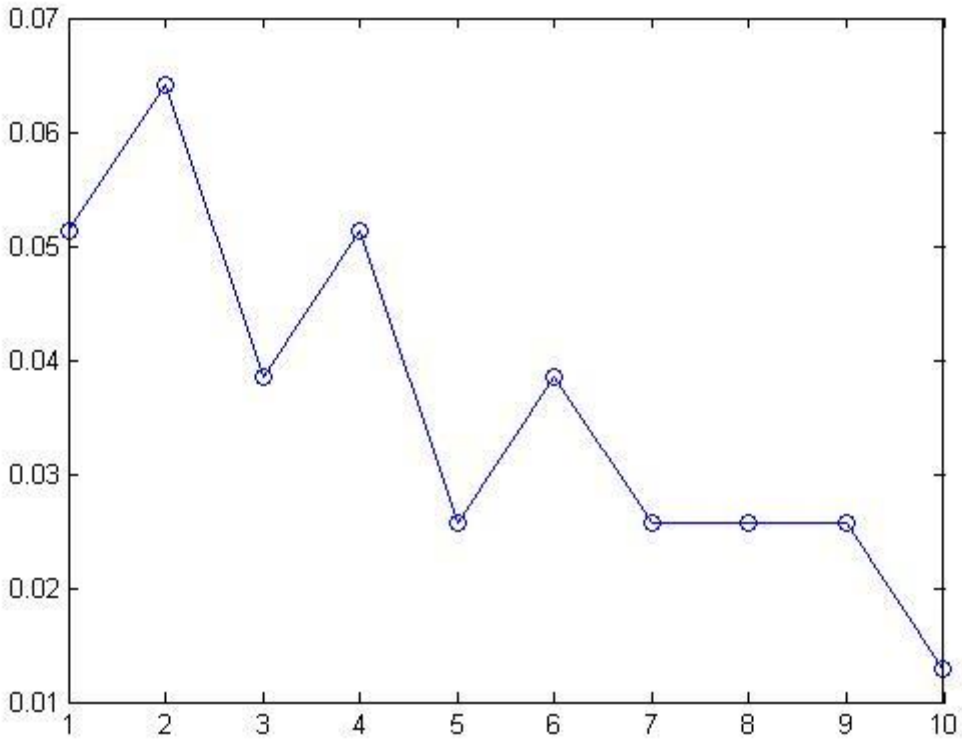


Grafico de error para iris.dat

Podemos apreciar que a mayor número de vecinos evaluados obtenemos una menor tasa de error, pero también una menor identidad.

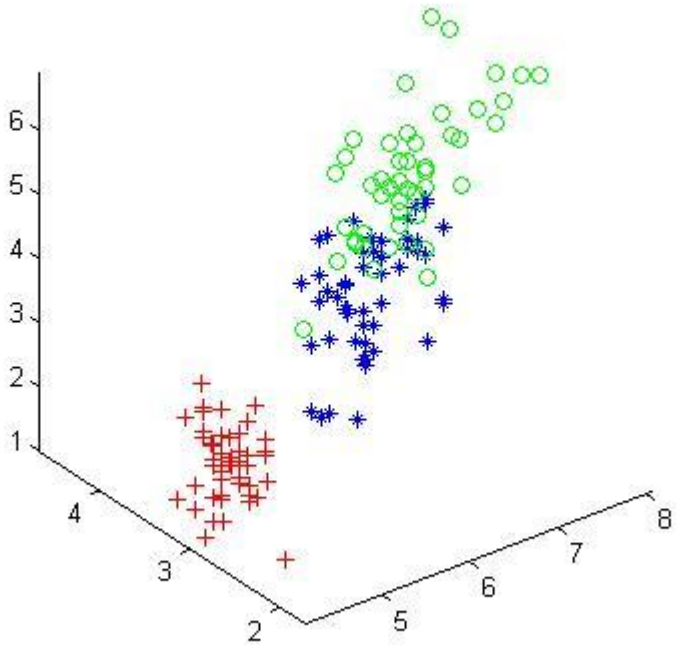
Las distintas clases mantienen un orden similar tras ser separadas. Si bien hay cierto desplazamiento en los grupos de valores, a efectos de la separabilidad esta se mantiene en los subconjuntos generados por `separador`.

PRÁCTICA 3. Condensación y Edición de Dataset

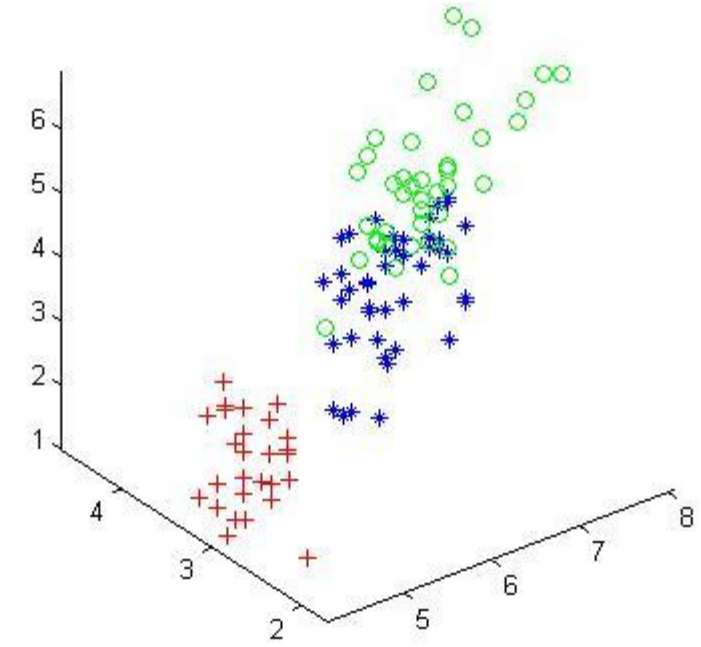
En esta parte nos centraremos en las decisiones al procesar un conjunto e datos referentes a la condensación y edición de un dataset.

Para la condensación abordaremos dos métodos, la Frontera de Decisión Consistente y el Conjunto Mínimo Consistente, implementadas en las funciones `CondensacionDBC` y `CondensacionMCS` respectivamente.

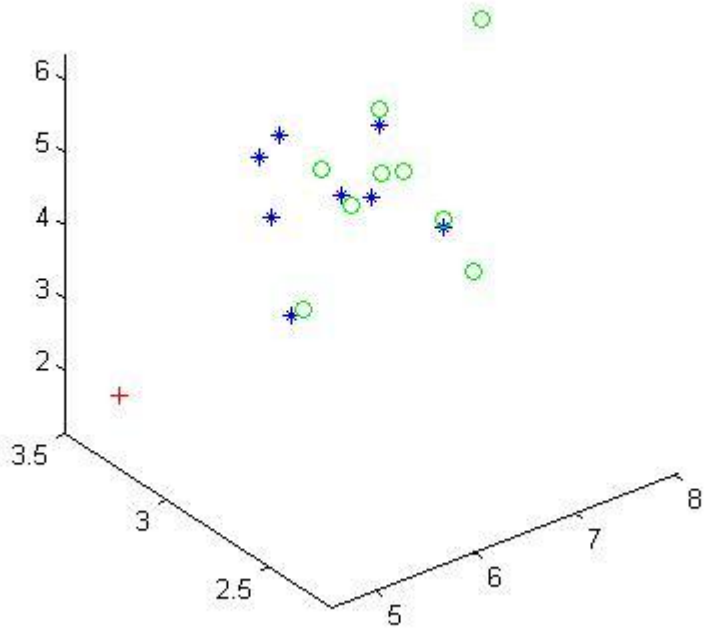
La primera mantiene las fronteras de decisión originales, mientras que la segunda desprecia las fronteras pero mantiene el resultado del clasificador NN. Por definición la segunda resulta mucho más agresiva que la primera, lo cual puede verse reflejado en los resultados obtenidos.



Conjunto iris.dat original.



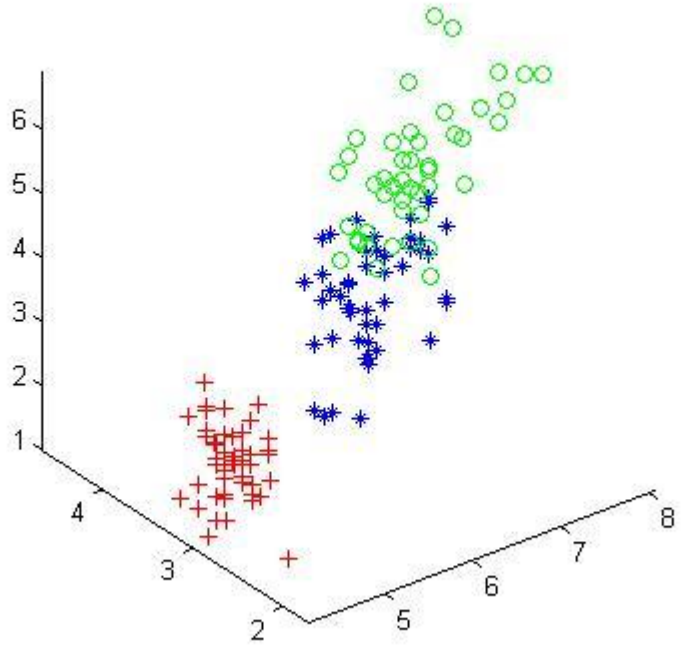
Condensación Frontera de Decisión Consistente aplicada a iris.dat



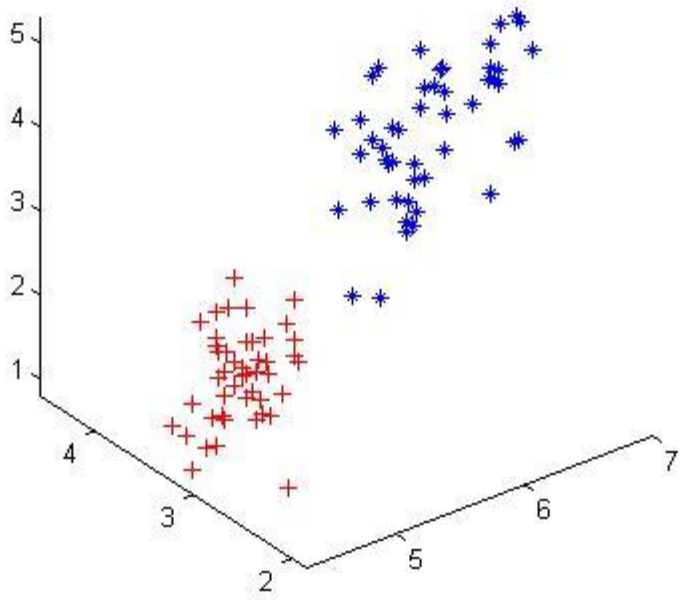
Condensación por Conjunto Mínimo Consistente aplicada a iris.dat

La siguiente cuestión a tratar será la Edición, desarrollada en la función *Edicion*. Podemos considerarla una variación de las anteriores pero centrada en kNN, de modo que funcione como un filtro donde el valor de k define el grado de suavización de las fronteras, lo que implica también cierta distorsión del problema.

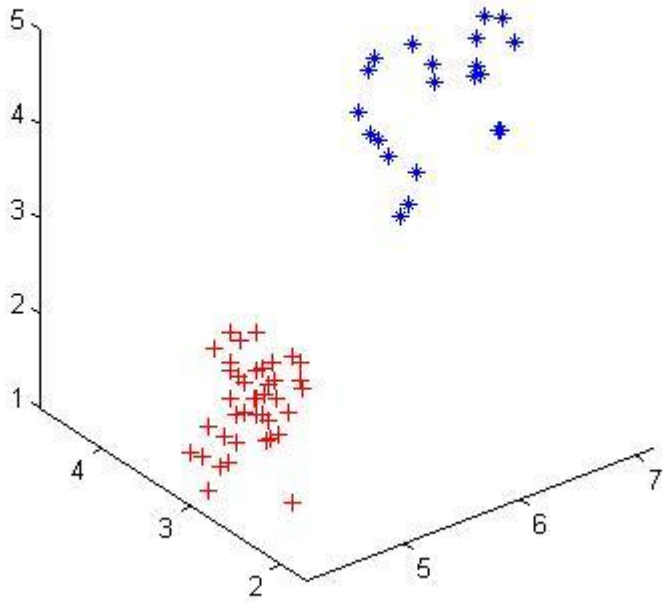
Esto se confirma en los resultados obtenidos, donde al incrementar k los cambios son más que evidentes, llegando en este caso particular a desaparecer en la práctica una de las clases en la representación.



Edición para iris.at con k=25



Edición para iris.at con k=100

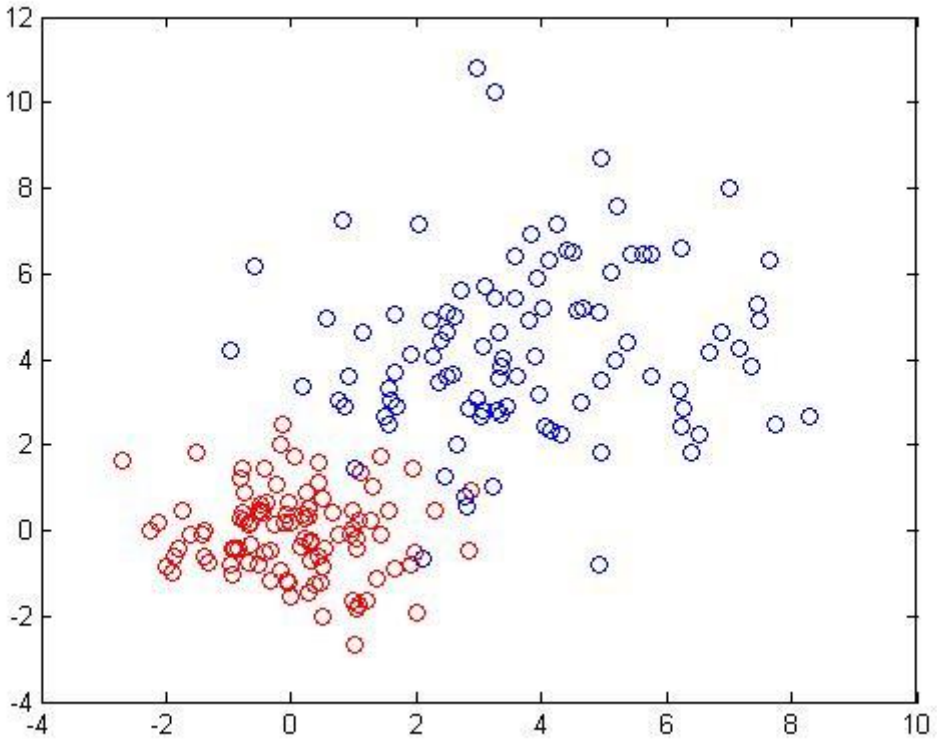


Edición para iris.at con k=150

PRÁCTICA 4. Clasificadores Lineales. Perceptron Simple.

El primer paso en el análisis a realizar consiste en linealizar el conjunto de datos sobre la clase a estudiar, considerando el resto como muestras negativas de la clase contraria.

El conjunto de estudio será Gauss2D por contar con una clase separable linealmente.



Gauss2D

Para poder realizar el análisis demandado se emplean dos funciones, PerceptronSimple1, para el `Perceptron` simple y `MuestraPerceptron` para el definido por muestras.

Destacar como resultados obtenidos para el conjunto de interés Gauss2D la existencia de convergencia confirmada y los tiempos de ejecución obtenidos:

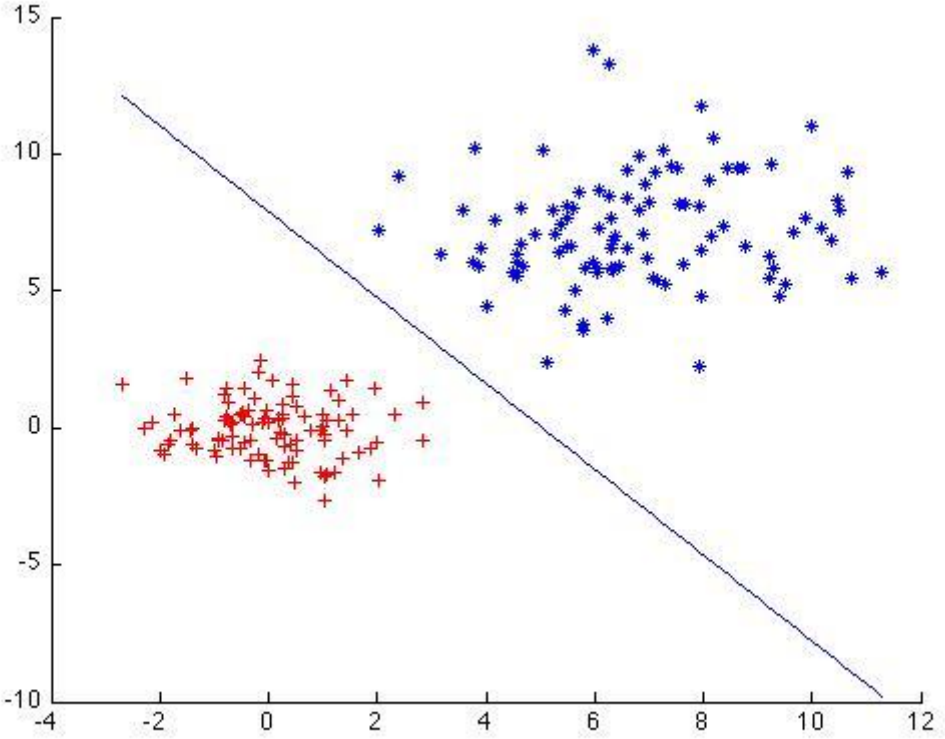
- Para el Perceptrón Simple: 0,0016.
- Para el Perceptrón por muestras: 0.0083.

El segundo caso tarda 5 veces más que el primero, como era de esperar por la gran cantidad de iteraciones en bucle que realiza el segundo y que ralentizan su funcionamiento.

PRÁCTICA 5. Clasificadores Lineales. Optimización de Funciones Objetivos

Tema 5, apartados 5.4

Si bien hemos podido clasificar y separar hasta ahora, el siguiente paso en esta tarea será la optimización y mejora de los resultados. Para ello aplicaremos la Máquina del Vector Soporte, implementada en `svm1`, ideada para obtener un clasificador con el margen más amplio posible entre las muestras próximas de las clases a tratar.



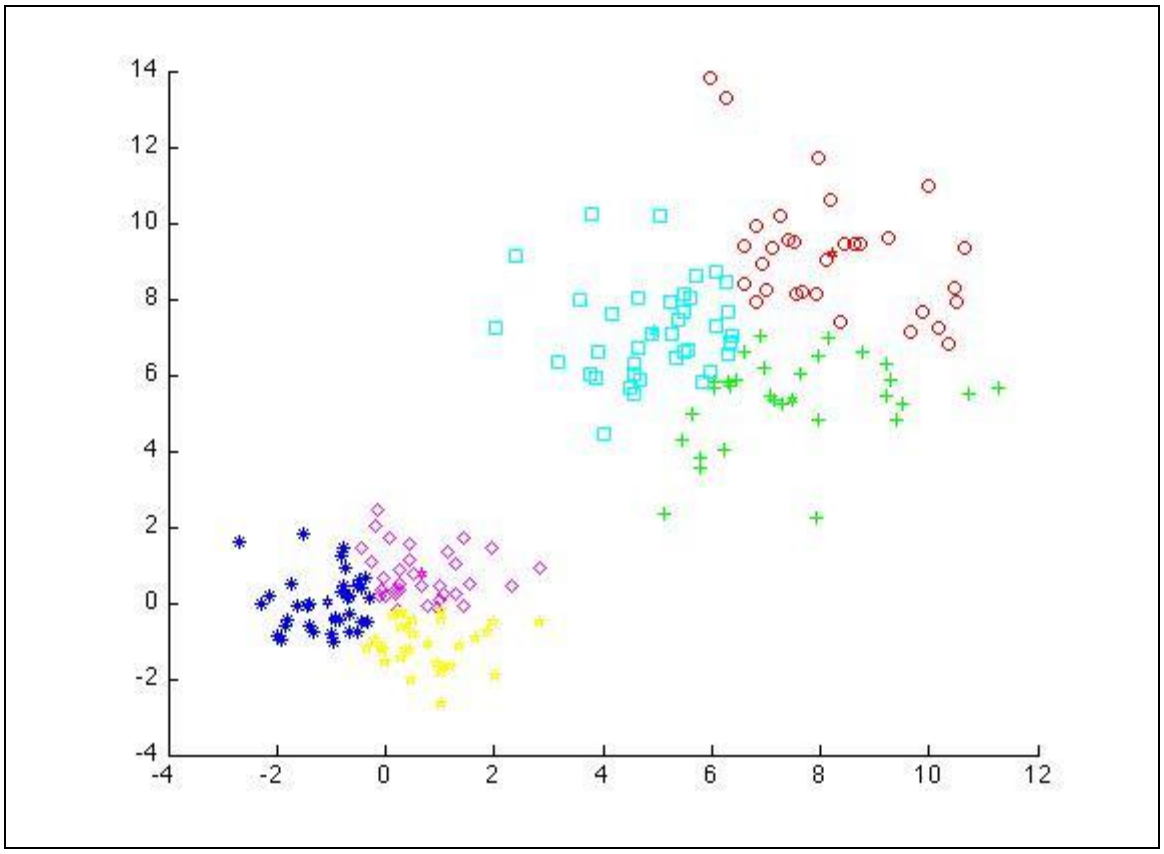
SMV para Gauss2D

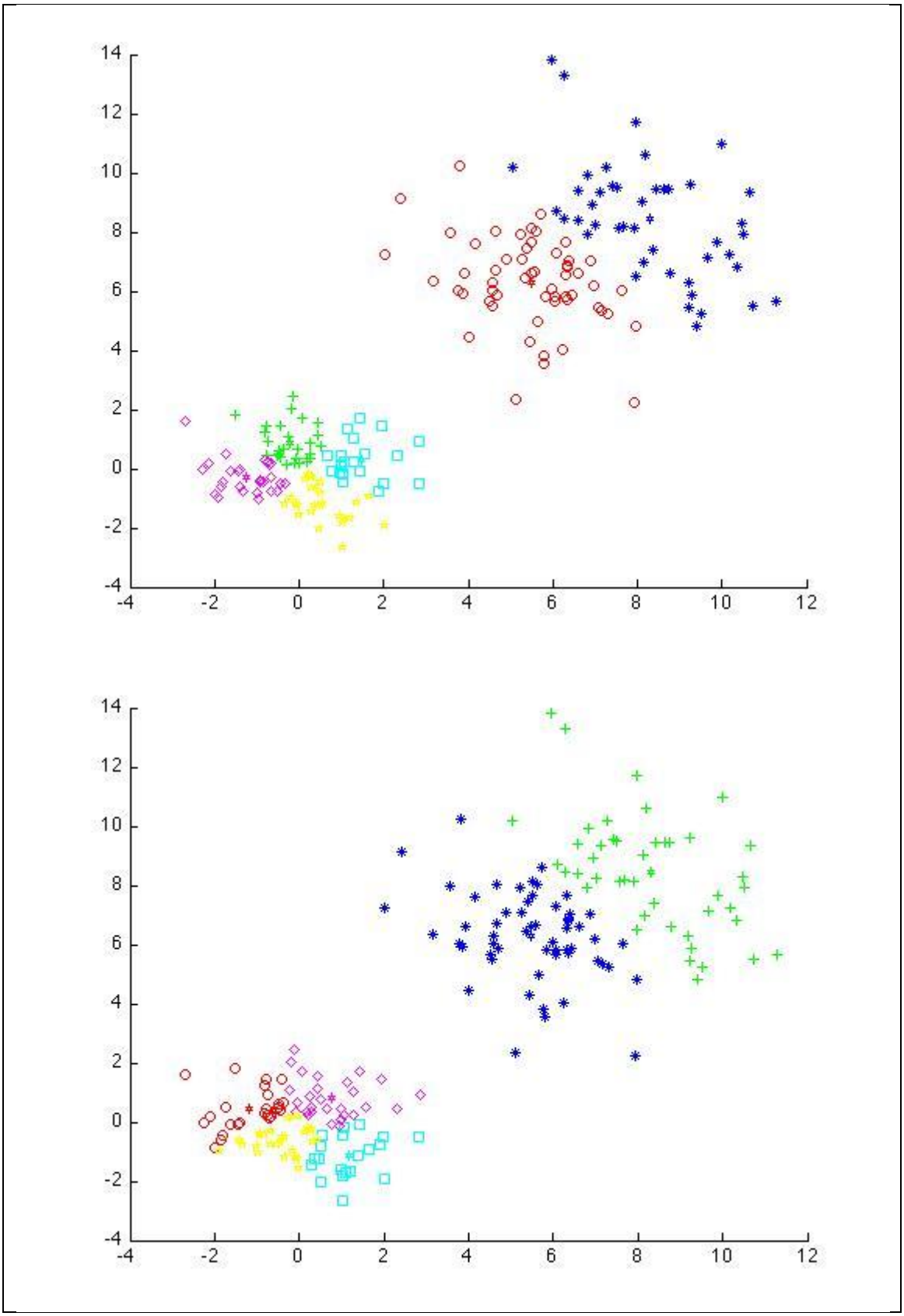
PRÁCTICA 6. Métodos de Agrupamientos

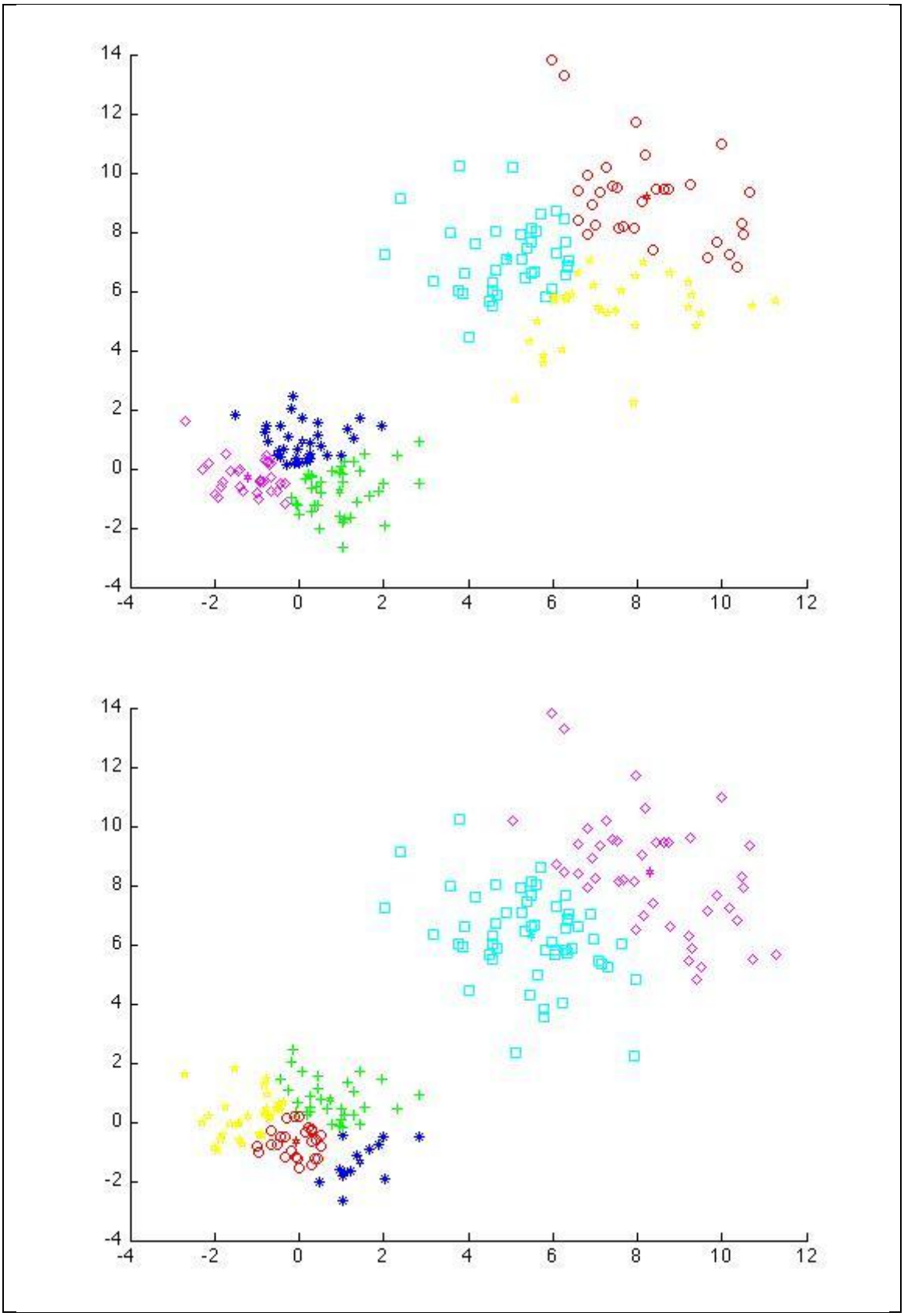
El último concepto que trabajaremos será los Métodos de Agrupamiento, concretamente los procedimientos iterativos K-medias y de Mínimo Error Cuadrático, desarrollados en las funciones *kmedia* y *minerror* respectivamente.

Destacar también las funciones de muestra de resultados que necesitaremos, `ImprimirCluster` e `ImprimirDiscriminante` encargadas de mostrar los Clúster y discriminantes resultantes.

Vemos algunos ejemplos de clusterización por el método K-medias en la siguiente tabla para el mismo conjunto de datos.

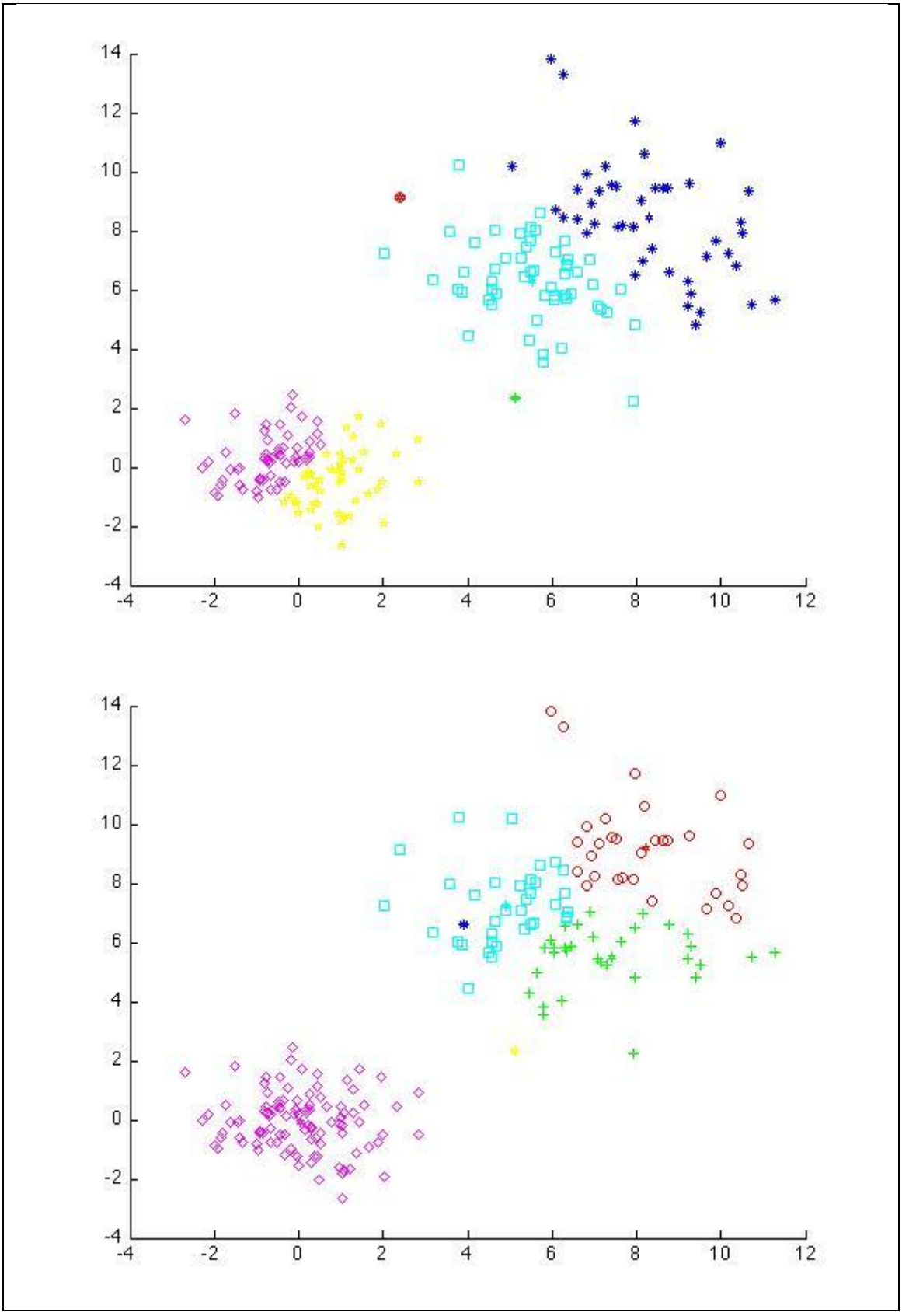


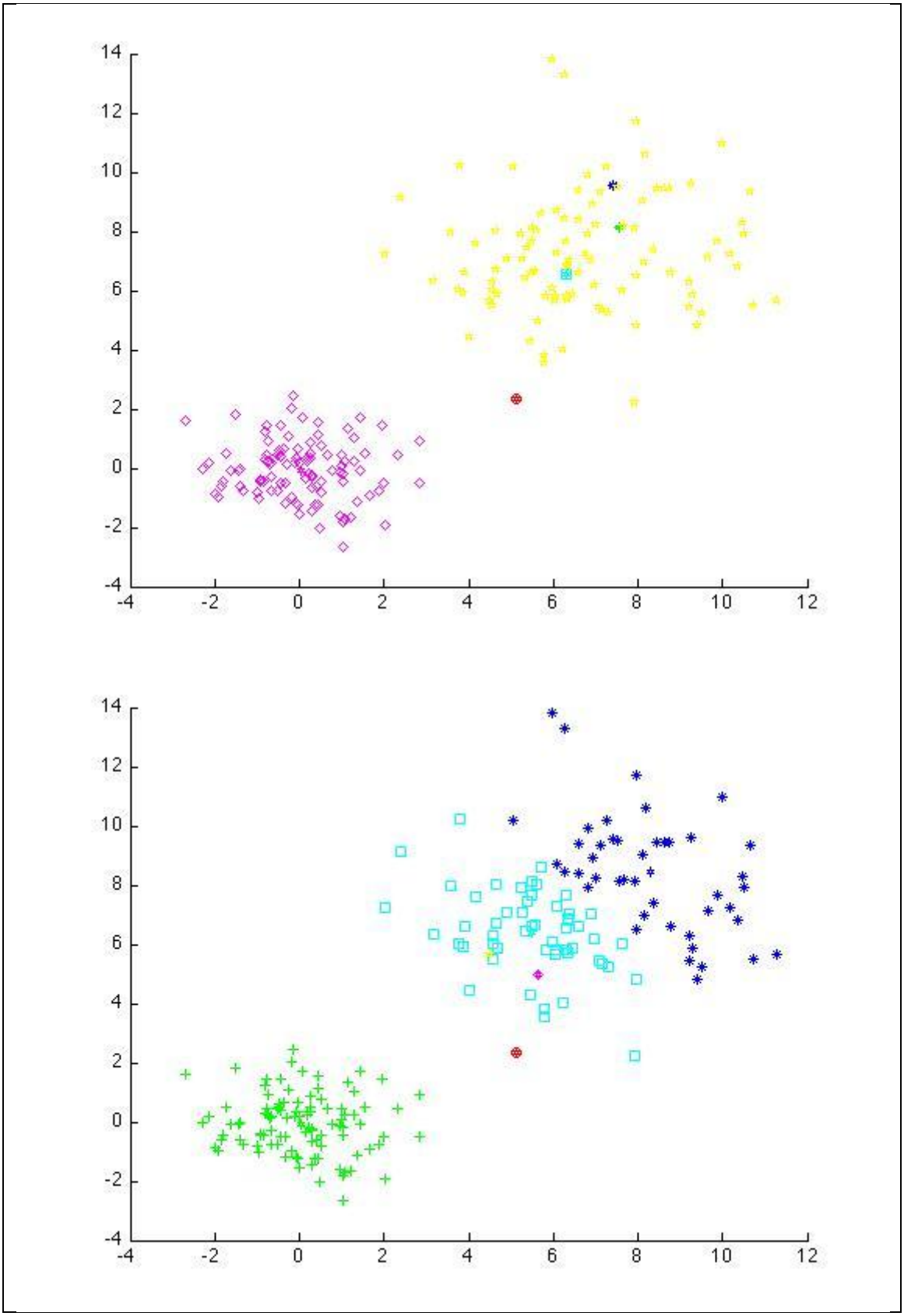

14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12
14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12

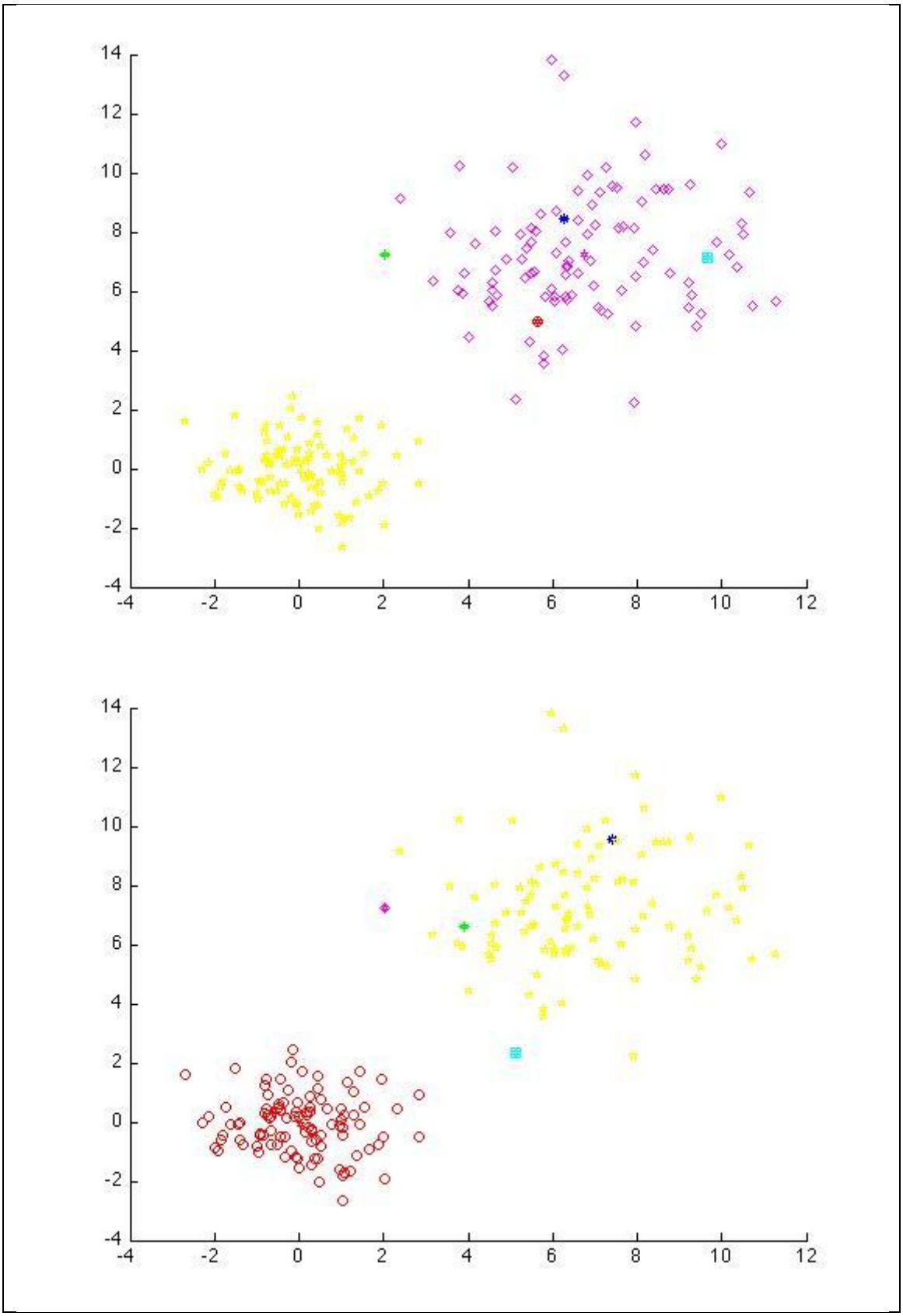


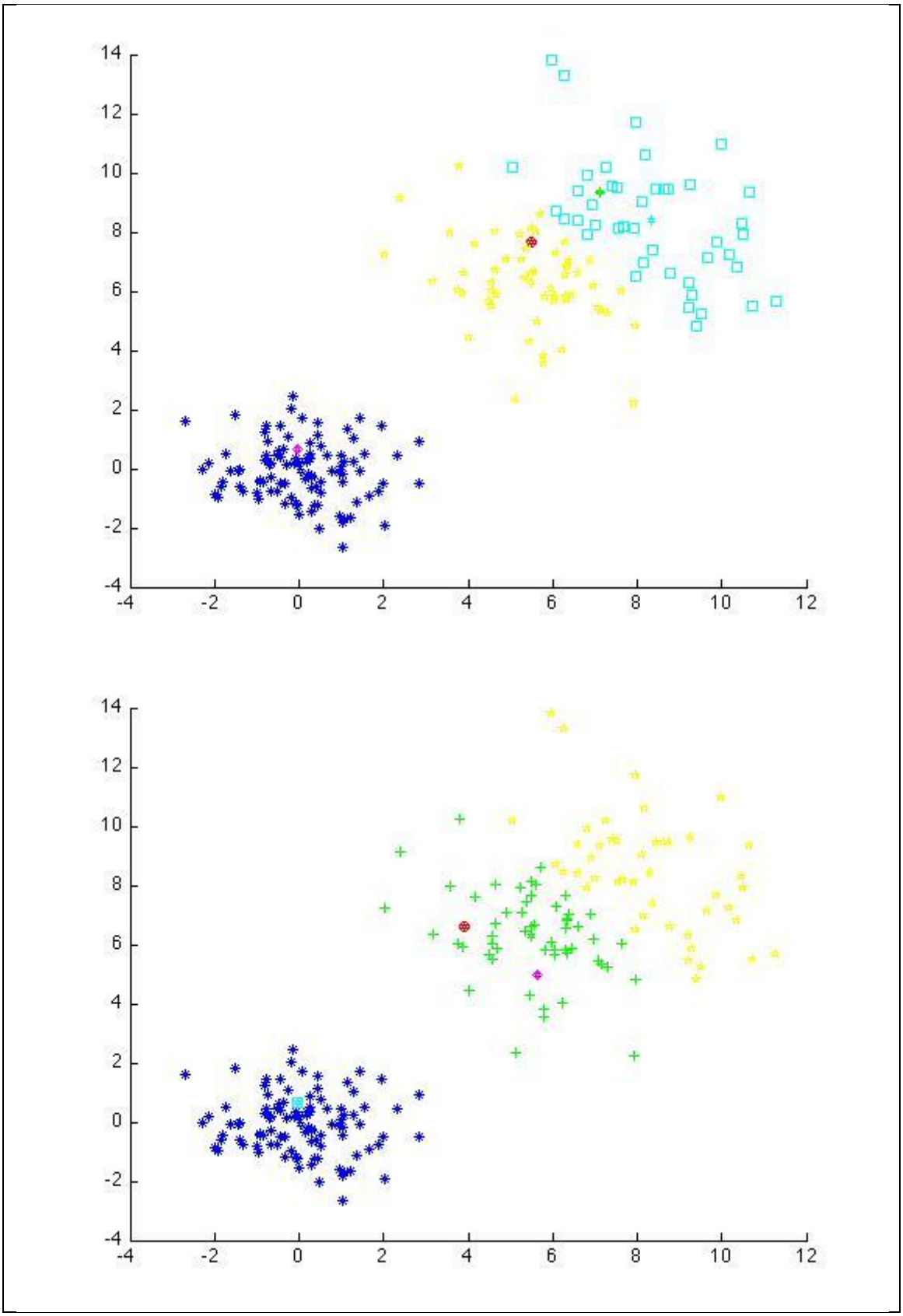
Clusterización por el método k-medias en Gauss2D

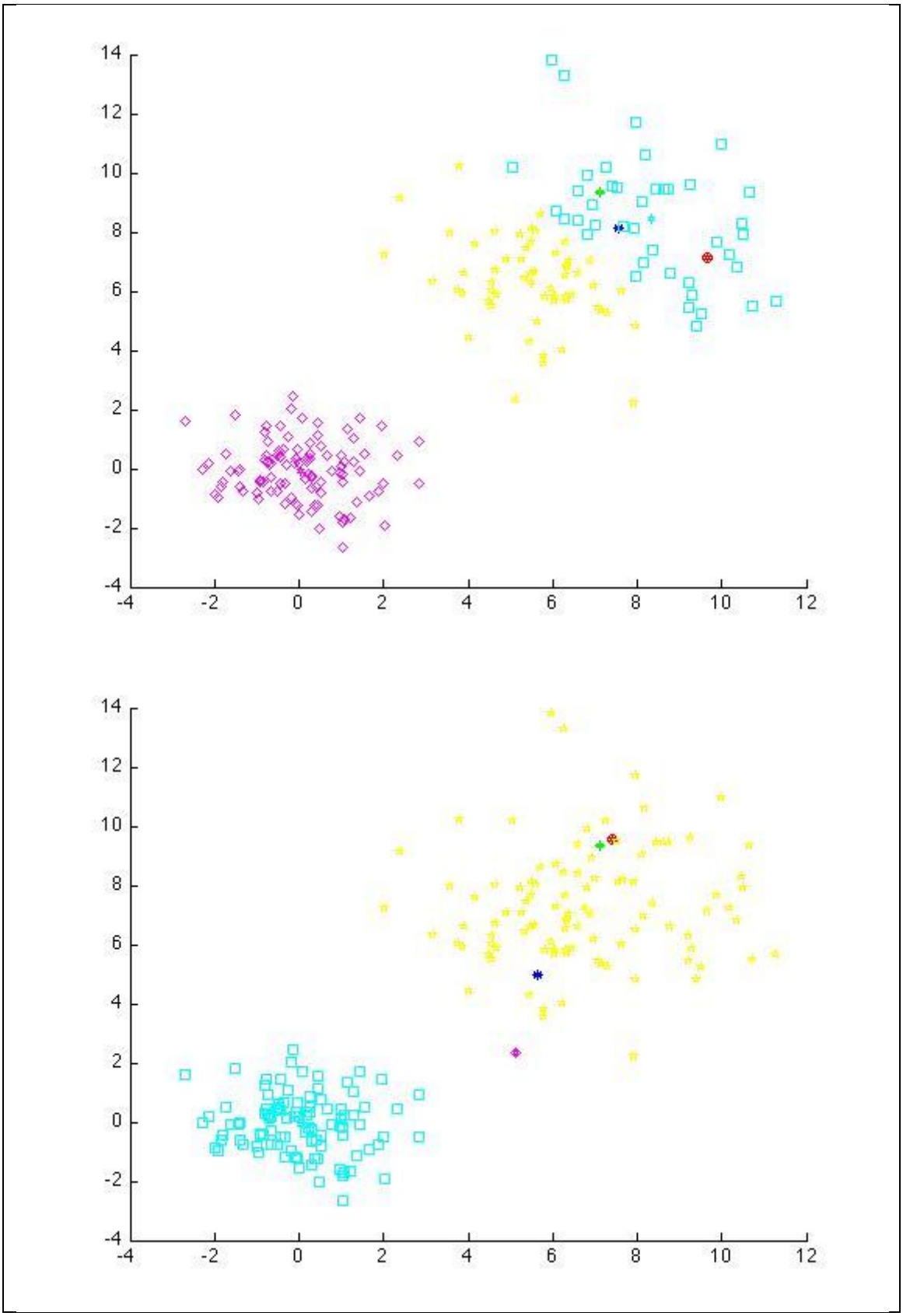
Finalmente presentamos diez casos de separación e identificación de clúster para la función *minerror* que nos permiten determinar la existencia de mínimos locales en el conjunto Gauss2D analizado.




14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12
14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12


14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12


14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12
14
12
10
8
6
4
2
0
-2
-4
-4
-2
0
2
4
6
8
10
12



Clúster y Mínimos locales en Gauss2D por Mínimo Error Cuadrático

ANEXO. Funciones desarrolladas.

PRÁCTICA 1. Organización y Visualización de Conjuntos de Datos.

Practica1 (nombre, tipo)

```
function [s] = Practica1 (nombre, tipo)
    fid = fopen(nombre);
    if(fid == -1)
        return;
    end
    nobj = fscanf(fid, '%u', 1);
    natrib = fscanf(fid, '%d', 1);
    nclases = 0;
    frec = [];
    if(tipo ~= 0)
        index = [];
        for j=1:nobj
            for k=1:natrib
                dato(j,k) = fscanf(fid, '%f ',1);
            end
            clase(j) = fscanf(fid, '%d\n',1);
            if(clase(j)>length(frec))
                while(clase(j)>length(frec))
                    frec(length(frec)+1) = 0;
                end
            end
            frec(clase(j)) = frec(clase(j))+1;
            if(frec(clase(j))==1)
                nclases = nclases+1;
            end
        end
    else
        for j=1:nobj
            for k=1:natrib-1
                dato(j,k) = fscanf(fid, '%f ',1);
            end
            dato(j,s) = fscanf(fid, '%d\n',1);
            clase(j) = 0;
            frec(clase(j)) = 0;
        end
    end
    s = struct('nobj', nobj, 'natrib', natrib, 'nclases', nclases, 'frec', frec, 'dato', dato, 'clase', clase);
end
```

Displaydataset(dataset)

```
function [] = Displaydataset (dataset)
    x = find(dataset.clase == 1);
    y = find(dataset.clase == 2);
    z = find(dataset.clase == 3);
    plot(dataset.dato(x,1), dataset.dato(x,2), 'r+', dataset.dato(y,1), dataset.dato(y,2), 'b*', dataset.dato(z,1), dataset.dato(z,2), 'go');
end
```

Displaydataset2(dataset,x,y)

```
function [] = Displaydataset2 (dataset, x, y)
  a = find(dataset.clase == 1);
  b = find(dataset.clase == 2);
  c = find(dataset.clase == 3);
  plot(dataset.dato(a,x), dataset.dato(a,y),'r+', dataset.dato(b,x), dataset.dato(b,y), 'b*', dataset.dato(c,x), dataset.dato(c,y), 'go');
end
```

Displaydataset3(dataset,x,y,z)

```
function [] = Displaydataset3 (dataset, x, y, z)

    a = find(dataset.clase == 1);

    b = find(dataset.clase == 2);

    c = find(dataset.clase == 3);

    plot3(dataset.dato(a,x), dataset.dato(a,y), dataset.dato(a,z), 'r+', dataset.dato(b,x), dataset.dato(b,y), dataset.dato(b,z), 'b*', dataset.dato(c,x), dataset.dato(c,y), dataset.dato(c,z), 'go');

    cameratoolbar

end
```

DisplaydatasetDoble(dataset,dataset2)

```
function [] = DisplaydatasetDoble (dataset,dataset2)

    hold on;

    x = find(dataset.clase == 1);

    y = find(dataset.clase == 2);

    z = find(dataset.clase == 3);

    plot(dataset.dato(x,1), dataset.dato(x,2), 'ro', dataset.dato(y,1), dataset.dato(y,2), 'bo', dataset.dato(z,1), dataset.dato(z,2), 'go');

    x = find(dataset2.clase == 1);

    y = find(dataset2.clase == 2);

    z = find(dataset2.clase == 3);

    plot(dataset2.dato(x,1), dataset2.dato(x,2), 'r*', dataset2.dato(y,1), dataset2.dato(y,2), 'b*', dataset2.dato(z,1), dataset2.dato(z,2), 'g*');

    hold off;

end
```

Displaydataset3Doble(dataset,dataset2)

```
function [] = DisplayDataset3Doble (dataset,dataset2)

    hold on;

    x=1;

    y=2;

    z=3;



    a = find(dataset.clase == 1);

    b = find(dataset.clase == 2);

    c = find(dataset.clase == 3);

    plot3(dataset.dato(a,x), dataset.dato(a,y), dataset.dato(a,z), 'ro', dataset.dato(b,x), dataset.dato(b,y), dataset.dato(b,z), 'bo', dataset.dato(c,x), dataset.dato(c,y), dataset.dato(c,z), 'go');

    a = find(dataset2.clase == 1);

    b = find(dataset2.clase == 2);

    c = find(dataset2.clase == 3);

    plot3(dataset2.dato(a,x), dataset2.dato(a,y), dataset2.dato(a,z), 'r*', dataset2.dato(b,x), dataset2.dato(b,y), dataset2.dato(b,z), 'b*', dataset2.dato(c,x), dataset2.dato(c,y), dataset2.dato(c,z), 'g*');

    hold off;

    cameratoolbar

end
```

separador(dataset,factor)

```
function [dataset1, dataset2] = separador(dataset,factor)

    rand('state', sum(100*clock));

    n = dataset.nobj*factor;

    d1nobj = 0;

    d2nobj = 0;

    d2natrib = dataset.natrib;

    d1natrib = dataset.natrib;

    d2clases = dataset.nclases;

    d1clases = dataset.nclases;

    d1dato = 0;

    d2dato = 0;

    d1clase = 0;

    d2clase = 0;

    d1frec = 0;

    d2frec = 0;

    for i=1:dataset.nclases

        f = round(dataset.frec(i)*factor)+d1nobj;

        datos = find(dataset.clase==i);

        for k=1:dataset.natrib

            dato(:, k) = dataset.dato(datos,k);

        end

        j = 0;

        while (j<length(find(dataset.clase == i))),
```

```
        if((rand()<factor)&&(d1nobj<f))
            for k=1:dataset.natrib
                d1dato(d1nobj+1, k) = dato(j+1, k);
            end
            d1nobj = d1nobj + 1;
            d1clase(d1nobj) = i;
        else
            for k=1:dataset.natrib
                d2dato(d2nobj+1, k) = dato(j+1, k);
            end
            d2nobj = d2nobj + 1;
            d2clase(d2nobj) = i;
        end
        j = j+1;
    end
end
for i=1:dataset.nclases
    d1frec(i) = length(find(d1clase==i));
    d2frec(i) = length(find(d2clase==i));
end

dataset1 = struct('nobj', d1nobj, 'natrib', d1natrib, 'nclases', d1clases, 'frec', d1frec, 'dato', d1dato, 'clase', d1clase);
dataset2 = struct('nobj', d2nobj, 'natrib', d2natrib, 'nclases', d2clases, 'frec', d2frec, 'dato', d2dato, 'clase', d2clase);
end
```

NN(dataset,vector)

```
function [clase, objeto] = NN(dataset, vector)

    x = repmat(vector, dataset.nobj, 1);

    x = dataset.dato - x;

    b = sqrt(dot(x(:,:)',x(:,:)')')

    n = find(b==min(b));

    sort(b)

    objeto = dataset.dato(n,:);

    clase = dataset.clase(n);

end
```

KNN(dataset,vector,k)

```
function [clase, prob] = KNN(dataset, vector, k)

    x = repmat(vector, dataset.nobj, 1);

    x = dataset.dato - x;

    b = sqrt(dot(x(:,:)',x(:,:)')');

    j = sort(b);

    j = j(1:k);

    for i=1:k

        w = find(j(i)==b(:));

        clases(i) = dataset.clase(w(1));

    end

    prob(1:dataset.nclases) = 0;

    for i=1:dataset.nclases
```

```
        prob(i) = length(find(clases(:) == i))/k;
    end
    clase = find(max(prob)==prob);
    clase = clase(1);
end
```

Test_KNN(datset1,dataset2,k)

```
function [error, prob] = Test_KNN(dataset1, dataset2, k)
   probs(1:dataset1.nclases, 1:dataset1.nclases) = 0;
   clases(1:dataset2.nobj) = 0;
   for i=1:dataset2.nobj
       [clases(i) probs(i,:)] = KNN(dataset1, dataset2.dato(i,:), k);
   end
   error = 0;
   prob(dataset1.nclases, dataset1.nclases) = 0;
   nclases(1:dataset2.nclases,1:dataset2.nclases) = 0;
   for i=1:dataset2.nobj
       if clases(i) ~= dataset2.clase(i);
           error = error + 1;
       end
       nclases(dataset2.clase(i),clases(i)) = nclases(dataset2.clase(i),clases(i)) + 1;
   end
   ntot(1:dataset2.nclases) = 0;
   for i=1:dataset2.nclases
       for(j=1:dataset2.nclases)
```

```
            ntot(i) = ntot(i)+ nclases(i,j);
        end
    end
    for i=1:dataset2.nclases
        prob(i,:) = nclases(i,:)/ntot(i);
    end
    error = error / dataset2.nobj;
end
```

Resultado_KNN(dataset,factor,kmax)

```
function [] = Resultado_KNN(dataset, factor, kmax)
    [dataset1 dataset2] = separador(dataset, factor);
    for i=1:kmax
        [error prob] = Test_KNN(dataset1, dataset2, i);
        total(i) = error;
    end
    plot(1:kmax, total, 'bo-');
end
```

PRÁCTICA 3. Condensación y Edición de Dataset

CondensacionDBC(dataset1)

```
function dataset2 = CondensacionDBC(dataset1)

    Conexiones = Conectividad(dataset1.dato);

    dato=[];

    clase=[];

    for i=1:size(Conexiones,1),

        insertar=false;

        for j=1:size(Conexiones,1),

            if (Conexiones(i,j) & j ~= i)

                if (dataset1.clase(i)~= dataset1.clase(j))

                    insertar=true;

                    break;

                end

            end

        end

        if (insertar)

            dato = [dato; dataset1.dato(i,:)];

            clase = [clase; dataset1.clase(i)];

        end

    end

    dataset2 = struct('nobj', size(dato,1), 'natrib', dataset1.natrib, 'nclases', dataset1.nclases, 'frec', dataset1.frec, 'dato', dato,
'clase', clase);

Displaydataset3 (dataset2, 1, 2, 3)
```

```
end



function C = Conectividad(X)

    T = delaunayn(X);

    C = eye(size(X,1));

    for i=1:size(T,1),

        A = nchoosek(T(i,:),2);

        for j=1:size(A,1),

            C(A(j,1),A(j,2)) = 1;

        C(A(j,2),A(j,1)) = 1;

        end

    end

end
```

CondensacionMCS(dataset1)

```
function dataset2 = CondensacionMCS(dataset1)

    % 1. Hacer S el dataset original y C vacio.

    C=[];

    S=[1:dataset1.nobj];



    % 2. Elegir un elemento arbitrario de S y pasarlo a C.

    Random = round(rand(1)*(size(S,2)-1))+1;

    C=struct('nobj', 1, 'natrib', dataset1.natrib, 'nclases', dataset1.nclases, 'frec', dataset1.frec, 'dato', dataset1.dato(Random,:),
'clase', dataset1.clase(Random));

    S = extraer(S,Random);



    %3. Clasificar elementos en S en base a C y transferir el primero erroneo desde S a C.

    Continuar = 1;

    while(Continuar)

        Continuar = 0;

        STemp = S;

        while (STemp)

            Random = round(rand(1)*(size(STemp,2)-1))+1;

            [clase, objeto] = NN(C, dataset1.dato(STemp(Random),:));

            if(clase ~= dataset1.clase(STemp(Random)))

                %4. Retornar a 3 hasta que no exista ninguno erroneo o S

                %este vacio.

                C.nobj= C.nobj + 1;

                C.dato= [C.dato; dataset1.dato(STemp(Random),:)];

                C.clase= [C.clase; dataset1.clase(STemp(Random))];
```

```
            IndiceAux =find(S==STemp(Random));

            S = extraer(S,IndiceAux);

            Continuar = 1;

            break;

        end

        STemp = extraer(STemp,Random);

    end

end

dataset2 = C;

Displaydataset3 (dataset2, 1, 2, 3)



end



function V = extraer(V0,Indice)

    if(Indice >1)

        V = [V0(1:(Indice-1)), V0((Indice+1):end)];

    else %% Caso que Indice sea 0

        V = V0(2:end);

    end

end
```

Edicion(dataset1,k)

```
function [s] = Edicion(dataset1,k)



   conjunto = [];

   clases = [];

   frec = [];

   for i=1:dataset1.nclases

       frec(i)=0;

   end

   tam=1;



  for i=1:dataset1.nobj

      [clase prob]=KNN(dataset1, dataset1.dato(i,:), k);

      if dataset1.clase(i)==clase

          for j=1:dataset1.natrib

              conjunto(tam, j) = dataset1.dato(i, j);

          end

          clases(tam)=clase;

          frec(clase)=frec(clase)+1;

          tam=tam+1;

      end

  end



s = struct('nobj', tam-1, 'natrib', dataset1.natrib, 'nclases', dataset1.nclases, 'frec', frec, 'dato', conjunto, 'clase', clases);

end
```

PRÁCTICA 4. Clasificadores Lineales. Perceptron Simple.

PerceptronSimple1(dataset, clase, ro, num)

```
function [a,test] = PerceptronSimple1(dataset, clase, ro, num)

    if nargin < 3

        ro = 0.001;

        num = 100;

    end

    [m,n] = size(dataset.dato);

    I = ones(m,1);

    Y = [I, dataset.dato];

    I(find(dataset.clase ~= clase)) = - 1;

    Psi = repmat(I,[1,n+1]).*Y;




    [m,n] = size(Psi);

    a = rand(n,1);

    test = 0;

    for i=1:num

        b = Psi*a;

        index = find(b<0);

        if isempty(index)

            test = 1;

            break;
```

```
        end;

        s = sum(Psi(index,:),1);

        a = a + ro*s';

    end

end
```

PerceptronMuestra(dataset, clase, ro, num)

```
function a = PerceptronMuestra(dataset, clase, ro, num)
    if nargin < 3
        ro = 0.001;
        num = 100;
    end
    [m,n] = size(dataset.dato);
    I = ones(m,1);
    Y = [I, dataset.dato];
    I(find(dataset.clase ~= clase)) = - 1;
    Psi = repmat(I,[1,n+1]).*Y;



    [m,n] = size(Psi);
    a = rand(n,1);
    while 1
        cambios = 0;
        for i=1:m
            Z = Psi(i,:);
            if Z*a < 0
                a = a + ro*Z';
                cambios = cambios + 1;
            end
        end
        if cambios == 0 break,end
    end
end
```

PRÁCTICA 5. Clasificadores Lineales. Optimización de Funciones Objetivos

svm1(dataset, clase)

```
function [a,lambda] = svm1(dataset,clase)

    [m,n] = size(dataset.dato);

    I = ones(m,1);

    Y = [I, dataset.dato];

    I(find(dataset.clase ~= clase)) = - 1;

    Psi = repmat(I,[1,n+1]).*Y

    [m,n] = size(Psi);

    Q = eye(n);

    Q(1,1)=0;

    [a,fval,exitflag,out,lam] = quadprog(Q,[],-Psi,-ones(m,1));

    lambda = lam.ineqlin;

end
```

PRÁCTICA 6. Métodos de Agrupamientos

kmedia(dataset,k)

```
function [prot,W]=kmedia(dataset,k)

    [m,n] = size(dataset.dato);

    % La Matriz W representa a cual de los K clusters pertenece cada uno de

    % los m elementos. Asi W(i,j)==1, significa que el elemento j pertenece

    % al cluster k . Cada elemento debe pertenecer a un y solo a un cluster
```

```
% y cada cluster tiene que tener algun elemento
% Se inicializara la matriz W, como una Matriz (Numero de clases X
% Numero de muestras), todo a 0


W = zeros(k,m);


% Se hara que cada elemento pertenezca a algun cluster
for i = 1 : m
    W(randi(k),i)= 1;
end
% Se comprobara que todos los cluster tienen almenos un elemento, y si
% alguno no tiene se moveran elementos aleatoriamente hasta que se
% cumpla
while 1
    for i = 1 : k
        while sum(W(:,i)) == 0
            aux = randi(m);
            W(find(W(:,aux)== 1),aux) = 0;
            W(i,aux) = 1;
        end
    end
    aux = 0;
    for i = 1 : k
        if (sum(W(:,i)) == 0)
        aux = aux +1;
```

```
        end
    end
    if aux == 0
        break;
    end

end
% Ahora W esta inicializada aleatoriamente
C = m;
while C ~= 0
    % Calcular el vector Z, que contendra los prototipos de cada cluster,
    % por asi decirlo, su centro
    Z = zeros(k,n);
    for i = 1:k
        % "Ni" sera el numero de elemento que pertenezen al cluster i
        Ni = 0;
        for j = 1:m
            if W(i,j) == 1
                Ni = Ni +1;
                for o = 1:n
                    Z(i,o) = Z(i,o) +dataset.dato(j,o);
                end
            end
        end
        if(Ni == 0)
```

```
        Ni = 1;
    end
    for o = 1:n
        Z(i,o) = Z(i,o)/Ni;
    end
end
C = 0;
% Ahora se colocaran cada una de las muestras en el cluster del
% prototipo mas cercano
for i = 1:m
    r = find(W(:,i)==1);
    rt = r;
    % Distancia de la muestra al prototipo de su cluster
    aux = 0;
    for o = 1:n
        aux = aux + (dataset.dato(i,o)-Z(r,o))^2;
    end
    %aux = sqrt(aux);
    for j = 1:k
        if j ~= r
            aux2 = 0;
            for o = 1:n
                aux2 = aux2 + (dataset.dato(i,o)-Z(j,o))^2;
            end
            %aux2 = sqrt(aux2);
```

```
                if aux > aux2
                    aux = aux2;
                    rt = j;
                end
            end
        end
        if r ~= rt
            % Hay que cambiar
            C = C +1;
            W(r,i) = 0;
            W(rt,i) = 1;
        end
    end
    prot = Z;
end
```

minerror(dataset,k)

```
function [prot,W,J] = minerror(dataset,k)
    [m,n] = size(dataset.dato);
    % La Matriz W representa a cual de los K clusters pertenece cada uno de
    % los m elementos. Asi W(i,j)==1, significa que el elemento j pertenece
    % al cluster k . Cada elemento debe pertenecer a un y solo a un cluster
    % y cada cluster tiene que tener algun elemento
    % Se inicializara la matriz W, como una Matriz (Numero de clases X
```

```
% Numero de muestras), todo a 0



% N sera un vector con el numero de elementos en cada cluster (para

% evitar contar varias veces

W = zeros(k,m);

N = zeros(k);



% Se hara que cada elemento pertenezca a algun cluster

for i = 1 : m

    aux = randi(k);

    W(aux,i)= 1;

    N(aux) = N(aux)+1;

end

% Se comprobara que todos los cluster tienen almenos un elemento, y si

% alguno no tiene se moveran elementos aleatoriamente hasta que se

% cumpla

while 1

    for i = 1 : k

        while sum(W(:,i)) == 0

            aux = randi(m);

            aux2 = find(W(:,aux)== 1);

            W(aux2,aux) = 0;

            W(i,aux) = 1;

            N(aux2)= N(aux2)-1;

            N(aux)= N(aux)+1;
```

```
        end
    end
    aux = 0;
    for i = 1 : k
        if (sum(W(:,i)) == 0)
        aux = aux +1;
        end
    end
    if aux == 0
        break;
    end

end
% Ahora W esta inicializada aleatoriamente
C = m;
while C ~= 0
    % Calcular el vector Z, que contendra los prototipos de cada cluster,
    % por asi decirlo, su centro
    Z = zeros(k,n);
    for i = 1:k
        % "Ni" sera el numero de elemento que pertenezen al cluster i
        Ni = 0;
        for j = 1:m
            if W(i,j) == 1
                Ni = Ni +1;
```

```
            for o = 1:n
                Z(i,o) = Z(i,o) +dataset.dato(j,o);
            end
        end
    end
    if Ni == 0
        Ni = 1;
    end
    for o = 1:n
        Z(i,o) = Z(i,o)/Ni;
    end
end

C = 0;

% Ahora se colocaran cada una de las muestras en el cluster del
% prototipo mas cercano
for i = 1:m
    r = find(W(:,i)==1);
    rt = r;
    % Distancia de la muestra al prototipo de su cluster
    % aux2 = (Ni/Ni-1)(||Zi -Xr||^2)
    aux = 0;
    for o = 1:n
        aux = aux + (dataset.dato(i,o)-Z(r,o))^2;
```

```
end

%aux = sqrt(aux);

aux = aux* N(r) / (N(r)-1);

for j = 1:k

    if j ~= r

        % aux2 = ||Zi -Xr||^2

        aux2 = 0;

        for o = 1:n

            aux2 = aux2 + (dataset.dato(i,o)-Z(j,o))^2;

        end

        %aux2 = sqrt(aux2);

        aux2 = aux2* N(j) / (N(j)-1);

        if aux > aux2

            aux = aux2;

            rt = j;

        end

    end

end

if r ~= rt

    % Hay que cambiar

    C = C +1;

    W(r,i) = 0;

    W(rt,i) = 1;

    N(r) = N(r)-1;

    N(rt) = N(rt)+1;
```

```
            end
        end
    end
    prot = Z;
    % Ahora se calculara la funcion objetivo para cumplir con los
    % requisitos de la practica. No se ha computado antes ya que habria
    % requerido recomputarlo con cada cambio.
    J = zeros(k,1);
    for i = 1:m
        aux = 0;
        aux2 = find(W(:,i)==1);
        for o = 1:n
            aux = aux + (dataset.dato(i,o)-Z(aux2,o))^2;
        end
        J(aux2)= J(aux2)+ aux;
    end
end
```

ImprimirCluster(dataset,W,prot)

```
function ImprimirCluster(dataset,W,prot)
    k = size(prot(:,1));
    [m,n]= size(dataset.dato);
    hold on;
    for i = 1:k
        X=[];
```

```
Y=[];
for j = 1:m
    if W(i,j) == 1
        X = [X dataset.dato(j,1)];
        Y = [Y dataset.dato(j,2)];
    end
end
switch int2str(i)
    case '1'
        plot(X, Y, 'b*');
        plot(prot(i,1),prot(i,2),'bh');
    case '2'
        plot(X, Y, 'g*');
        plot(prot(i,1),prot(i,2),'gh');
    case '3'
        plot(X, Y, 'r*');
        plot(prot(i,1),prot(i,2),'rh');
    case '4'
        plot(X, Y, 'c*');
        plot(prot(i,1),prot(i,2),'ch');
    case '5'
        plot(X, Y, 'm*');
        plot(prot(i,1),prot(i,2),'mh');
    case '6'
        plot(X, Y, 'y*');
```

```
            plot(prot(i,1),prot(i,2),'yh');

        end

    end

    hold off;

end
```

ImprimirDiscriminante(a, dataset, clase)

```
function ImprimirDiscriminante(a, dataset, clase)
    if nargin < 3
        clase = 1;
    end
    maxX = max(dataset.dato(:,1))
    minX = min(dataset.dato(:,1))
    X = minX:abs((minX-maxX)/100):maxX
    Y = (-a(1)-a(2)*X)/a(3)
    hold on;
    plot(X,Y);
    Displaydataset(dataset);
    hold off;
end
```