Universidad Autónoma de Yucatán

Facultad de Matemáticas

Licenciatura en Ingeniería en Computación

TEMAS SELECTOS DE APRENDIZAJE Y VISIÓN

Proyecto

Algoritmos Clasificadores

Integrantes:

- Gonzalez Gonzalez Juan Pablo
- Estrada Diaz Jesús Antonio.
- Peniche Pacheco Cristhian Kevin

Fecha: 26/ Enero/ 2018

Self Organising Maps

Los SOM o Self Organising Feature Maps nos proveen de una manera de representar información multidimensional en espacios de menores dimensiones. Es una técnica de compresión de información llamada “cuantización de vectores”.

El SOM es considerado un algoritmo de aprendizaje no supervisado, es decir, aprende a clasificar la información que se le alimenta sin necesidad de un input por parte del analista.

Existen varios tipos de SOM en distintas dimensiones, pero todos trabajan de la misma manera la cual es crear una red de nodos, cada uno de los cuales está conectado a la capa de input.

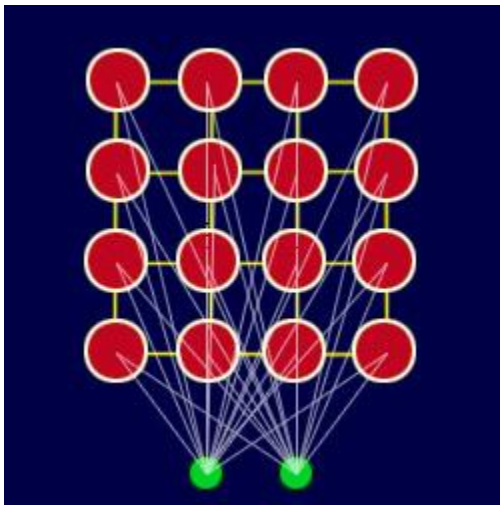

Figura 1. Mapa de nodos de 4x4

La figura 1 muestra un arreglo de nodos de 4x4 conectados a la capa de input.

Cada nodo tiene una posición topológica y contiene un vector de pesos de la misma dimensión que los vectores de input. De aquí el aprendizaje para la clasificación sigue los siguientes pasos

1. Los pesos de los nodos son inicializados
2. Un vector es elegido al azar de entre todos los vectores de input y es presentado a los nodos

3. Cada nodo entonces es examinado para determinar cual de todos tiene sus pesos de manera más similar a los valores del vector de entrada. El nodo ganador se conoce como el Best Matching Unit (BMU)
4. Se calcula un radio alrededor del BMU, este valor del radio comienza grande, pero va disminuyendo con cada iteración del algoritmo
5. Todos los nodos vecinos del BMU (nodos dentro del radio calculado) tendrán sus pesos afectados para asemejarse más al del vector de entrada, mientras más cercanos se encuentren estos al BMU más serán afectados sus pesos
6. Se repiten los pasos desde el dos hasta una condición de paro

Para la implementación del algoritmo de SOM en este ejercicio se usaron como datos de entrada características de 3 tipos de granos.

```
 8  nodos = randi(10,X,Y,Z);
 9  DataSeed;
10  radioIni=3;
11  consTiem=3;
12  LearningRI=1;
13  epocaFinal=100;
14
15  %Revolvemos los datos
16  for j=length(datos): -1: 1
17      idx = randi(j);
18      temp = datos(idx,:);
19      datos(idx,:) = datos(j,:);
20      datos(j,:) = temp(:);
21  end
```

Figura 2. Código para seleccionar un vector de entrada al azar

En la figura 2, en la línea 8 se inicializan un vector de nodos con valores aleatorios de la misma dimensión que los vectores de entrada. En el for de la línea 16 se revuelve el orden de los vectores de entrada de tal forma que uno será elegido al azar para comenzar el entrenamiento no supervisado

```
32 for epoca=1:1:epocaFinal
33   radio=radioIni*exp(-(epoca-1)/consTiem);
34   radio=radioIni;
35   for h=1: length(datos)
36     BMU_dist=9999999;
37
38     for i=1: 1: X
39       for j=1: 1: Y
40         distTemp=0;
41         for k=1: 1: Z
42           distTemp=distTemp+(datos(h,k)-nodos(i,j,k))^2;
43         end
44         distTemp=sqrt(distTemp);
45         if distTemp<=BMU_dist
46           BMU_dist=distTemp;
47           BMU_X=i;
48           BMU_Y=j;
49         endif
50         end
51     end
```

Figura 3. Código para calcular el BMU

En el for de la línea 32 se ejecuta el algoritmo de SOM hasta que se alcanza una condición de paro, en este caso dado por las épocas. El for de la línea 35 recorre busca el BMU de entre todos los nodos por cada vector de input

```
73     LearningR=LearningRI*exp(-(epoca-1)/consTiem);
74     for i=radioW: 1: radioE
75       for j=radioN: 1: radioS
76         for p=1:Z
77           nodos(i,j,p)=nodos(i,j,p)+(LearningR*(datos(h,p)-nodos(i,j,p)));
78         endfor
79       end
80     end
```

Figura 4. Código para cambiar los pesos del vecindario del BMU

Una vez encontrado el BMU procederemos a cambiar todos los pesos de todos los nodos que se encuentran en un radio previamente calculado alrededor del BMU con base a un *Learning Rate* y disminuiremos dicho radio.

El resultado será un mapa de nodos el cual poseerá características similares en sus pesos a los de una clase de semilla en una región del mapa.

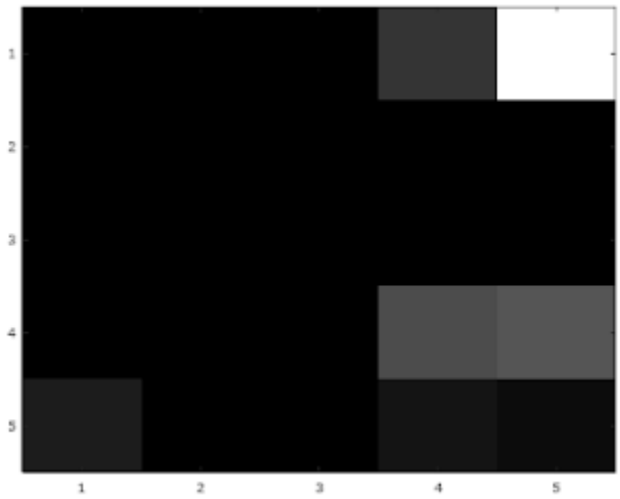

Figura 5. Clasificación resultante para la clase de semillas 1

La figura 5 representa un mapa de 5x5 de nodos cuyos valores en escala de grises representan la cantidad de veces que fueron seleccionados como nodos óptimos para la clase 1 de semillas.

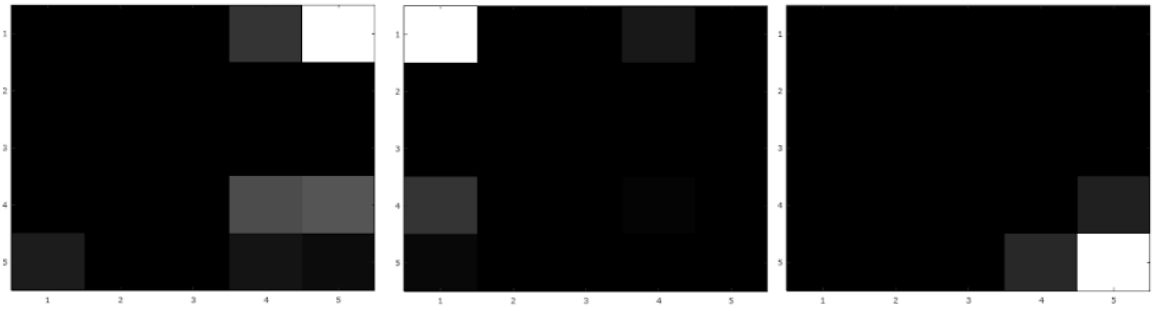

Figura 6. Clasificación resultante para las clases de semillas 1 (izq), 2 (centro), 3 (der).

Esta prueba se hizo tomando en cuenta 70 tipos de series de datos por cada clase de semilla distinta, 100 epocas, un *learning rate* de 1, un radio inicial de 3 y una constante de tiempo de decaimiento de 3.

Como resultados numéricos se obtuvieron las siguientes matrices que representan cada uno de los mapas

$$Matriz\ de\ ocurrencia_1 \begin{pmatrix} 0 & 0 & 0 & 7 & 33 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 11 \\ 4 & 0 & 0 & 3 & 2 \end{pmatrix}$$

$$Matriz\ de\ ocurrencia_2 \begin{pmatrix} 51 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Matriz\ de\ ocurrencia_3 \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 9 & 54 \end{pmatrix}$$

Con estas matrices de ocurrencia podemos calcular un índice de eficiencia con base a los datos que clasificó correctamente en contraste con los datos que no. Para este índice se toma en cuenta un radio de un espacio como el área en la que de haber sido clasificado ahí el vector será tomado como una clasificación aceptable centrado en el nodo con más ocurrencias.

$$Ef_1 = 57.14\%$$

$$Ef_2 = 72.85\%$$

$$Ef_3 = 100\%$$

K-medias

El algoritmo K-medias es muy sencillo y resuelve el problema clásico de clasificación en donde dado un conjunto de N datos con m características (*dimensiones*) crea k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano.

Consiste básicamente en dos pasos, dado un conjunto de k centroides, se alterna entre:

- Asignación: Asigna cada observación al grupo con la media (centroide) más cercana.
- Actualización: Calcular los nuevos centroides como el centroide de las observaciones en el grupo.

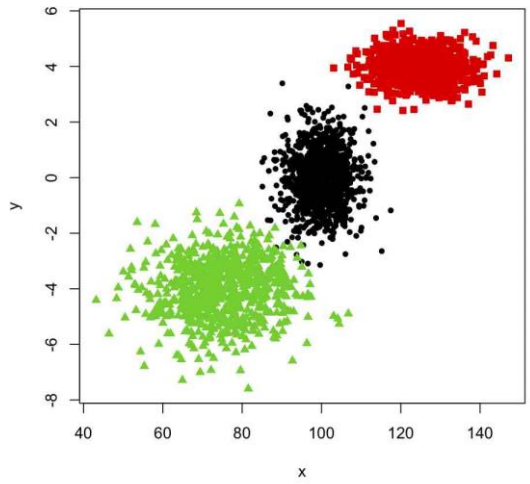

Figura 7. Representación de clasificación por el algoritmo k-medias

La implementación fue sencilla y siguiendo los pasos mencionados anteriormente. Primero fue importante seleccionar los k (en nuestro caso 3) centroides iniciales. Esto se realiza tomando aleatoriamente 3 muestras de 7 dimensiones de nuestro conjunto de datos, como se muestra en la figura 8.

```
16 cent = vector(randi(L, K, 1),:);
```

Figura 8. Selección aleatoria de centroides.

Posteriormente se realiza la parte de asignación, donde calculamos la distancia de cada dato a cada centroide y se guarda un vector de índices que especifica el centroide con menor distancia a cada dato.

```
for i = 1 : L
    for j = 1 : K
        dista(i,j) = sum((vector(i,:) - cent(j,:)) .^ 2);%guardamos distancias de centros a puntos
    end
    [minimo, ind] = min(dista(i,:));%Tomamos clase con menor distancia
    indices(i) = ind;
end
```

Figura 9. Parte de asignación del algoritmo.

Ahora, sabiendo qué centroide se le asigna a cada dato, podemos recalcular los tres centroides.

```
    for i = 1 : K
        clasei = vector(indices == i,:);%Separamos clases
        tamclase = size(clasei, 1);
        if tamclase == 1
            cent(i,:) = clasei;
        else
            cent(i,:) = sum(clasei)/tamclase;%Media de cada clase
        end
    end
end
```

Figura 10. Parte de actualización del algoritmo.

Esto se realiza tomando todos los vectores de cada clase y calculando su media. Se valida si la clase tiene solamente un dato y se repiten estos dos pasos hasta cumplir el número de iteraciones.

Podemos observar el desempeño del algoritmo simplemente comparando las etiquetas que tenemos de la base de datos con las etiquetas de salida del algoritmo.

```
Columns 1 through 38:

  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   3   1   1   3   1   1   1   3   1   1   3   3   1   1   1   1   1   1

Columns 39 through 76:

  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2   2   2
  1   3   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   3   3   3   3   3   1   1   1   1   1   3   2   2

Columns 77 through 114:

  2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
  2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   1   2   2   2   2   2   2   2   2   2   2

Columns 115 through 152:

  2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   3   3   3   3   3   3   3   3   3   3
  2   2   2   2   2   2   2   2   1   2   1   2   2   2   2   2   2   2   1   1   1   1   2   1   1   1   3   3   3   3   3   3   3   3

Columns 153 through 190:

  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3

Columns 191 through 210:

  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
  3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3   3
```

Figura 11. Vector de etiquetas originales (primera fila) y etiquetas generadas (segunda fila).

Así como los tres vectores centroide.

```
centroides =

   14.81910   14.53716    0.88052    5.59101    3.29936    2.70659    5.21754
   18.72180   16.29738    0.88509    6.20893    3.72267    3.60359    6.06610
   11.98866   13.28439    0.85274    5.22743    2.88009    4.58393    5.07424
```

Figura 12. Centroides generados por k-medias.

Aunque pueden ser difíciles de comparar debido al número de dimensiones del conjunto de datos. Por lo que puede ser útil encontrar la efectividad del algoritmo por cada clase simplemente dividiendo el número de ejemplos clasificados correctamente en cada clase por el total de ejemplos de dicha clase.

Para esto se generó un script llamado “kmedias_prueba” que calcule la efectividad de cada clase, teniendo los siguientes resultados.

```
>> kmedias_prueba
Total ejemplos
A =  210
Efectividad clase 1:
F1 =  82.857
Efectividad clase 2:
F2 =  85.714
Efectividad clase 3:
F3 =  100
```

Figura 13. Efectividad de clasificación por k-medias.

Donde claramente se observa que la clase 3 se encuentra muy bien separada de las demás, ya que todos sus ejemplos fueron asignados correctamente a la misma clase, a diferencia de la clase 1 y 2 que tienden a filtrar algunos de sus ejemplos entre ellas. Es importante mencionar que el algoritmo puede resultar con mejor efectividad en diferentes ejecuciones ya que los primeros centroides se seleccionan aleatoriamente.

Si queremos tener una representación un poco más visual del resultado podemos graficar cómo se agrupan o separan las clases con respecto a sus centroides, pero debido a que estos datos tienen 7 dimensiones tendríamos que tomar solamente dos o tres de ellas para graficar.

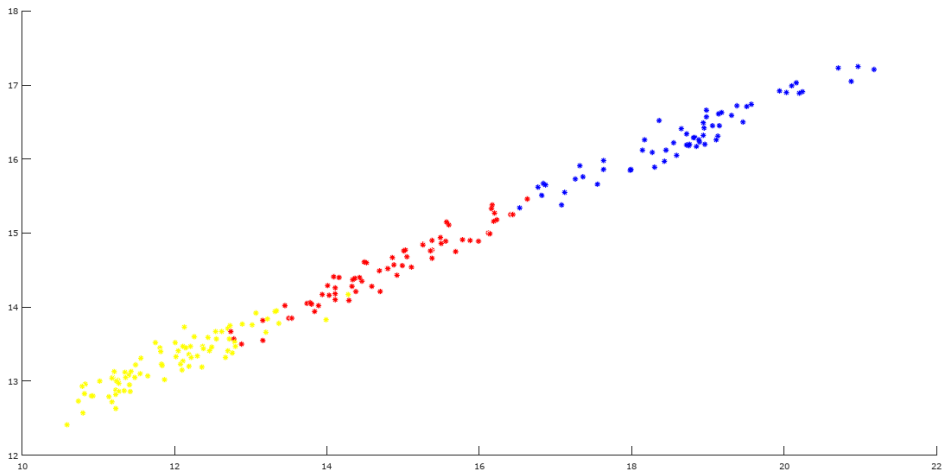

Figura 14. Resultado de clasificación observando dos dimensiones de los datos.

Siempre es complicado verificar los resultados con más de tres dimensiones, pero podría ayudar a comprender mejor como se distribuyen los datos.

Comparación

Podemos comparar ambos algoritmos tanto en implementación como en resultados. Con respecto a la implementación es justo decir que el algoritmo k-medias es más sencillo e intuitivo, ya que con solo inicializar centroides y repitiendo las dos etapas especificadas obtenemos los resultados deseados. De la misma manera es sencillo seguirle los pasos e incluso observar los resultados en tablas y gráficas, si las dimensiones de los datos son pequeñas. Los SOM fueron un tanto más complicados en su implementación; aunque tenía un mayor número de pasos a seguir, estos eran claros y no presentaban mayor problema, el problema surgía al utilizar varias variables para modificar radios de afectación a nodos y velocidades de aprendizaje. Fue necesario ajustar valores y realizar varias pruebas hasta encontrar un funcionamiento decente del algoritmo. También fue complicado mostrar los resultados y mapas finales ya que, una vez más, las dimensiones de los datos impedían una comprensión clara de la distribución de los mismos, por lo que se optó por comparar los datos de cada clase con los mapas finales y observar (mediante cuadros a escala de grises) su distribución.

En cuanto a los resultados, es claro que el algoritmo k-medias presentó una mayor efectividad en su clasificación. Esto depende también de la distribución de los datos, ya que en ambos casos la clase tres de semillas fue separa perfectamente y no parecían tener "infiltrados" de otras clases en ella.

Efectividad	**k-medias**	**SOM**
Clase 1	82.857	57.14
Clase 2	85.714	72.85
Clase 3	100	100

Es importante también notar que, aunque los SOM parecen tener menor efectividad a la hora de clasificar, tienen una ventaja sobre k-medias ya que no necesita saber en primera instancia el número de clases que presentan los datos, lo cual puede ser de gran ayuda en procesos en los que no se tiene este número.

Análisis de discriminante Gaussiano

El análisis de discriminante Gaussiano presenta un enfoque diferente a problema de clasificación de datos; ya que en lugar de encontrar una función que realice el trabajo de límite entre dos clases, como lo hace la regresión lineal o regresión logística, se crea un modelo o "molde" de cada clase que tenga sus características como la media o varianza, para posteriormente calcular la probabilidad de que un nuevo dato pertenezca a cualquiera de ellas.

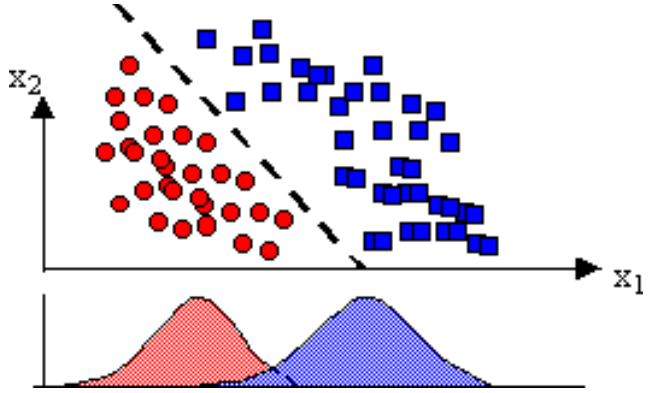

Figura 15 Distribución normal asociada a cada clase.

Este método de clasificación se usa comúnmente cuando los datos pueden ser aproximados con una distribución normal. En primer lugar se necesita un conjunto de datos de entrenamiento con sus respectivas etiquetas, para armar el modelo de cada clase y obtener una función que nos diga a qué clase tiene mayor probabilidad de pertenecer un nuevo dato.

Para realizar el algoritmo seguimos los pasos que nosotros nos planteamos:

- Tomar los datos de entrenamiento y calcular la media de cada característica de cada clase.
- Calcular la matriz de covarianza de ambas clases.
- Obtener la función de distribución normal de cada clase con las medias respectivas y la matriz de covarianza.

Para validar los resultados simplemente obtenemos la probabilidad de que cada dato pertenezca a una clase y tomamos la mayor.

Cabe resaltar que aunque el conjunto de datos presenta tres clases, se tomaron las que separaban mejor según los algoritmos de aprendizaje no supervisado; esto por simplicidad y ya que el algoritmo originalmente funciona para calcular probabilidades entre dos clases.

Se utiliza el teorema de Bayes para predecir la clase a la que pertenece un nuevo dato.

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

Es decir, la probabilidad que una semilla sea tipo 1, dado un nuevo vector de características, está dada por la probabilidad de que esas características pertenezcan o se ajusten mejor a los datos de las semillas tipo 1, esto multiplicado por la probabilidad de que una semilla sea tipo 1 (0.5 ya que solo hay dos clases y son equiprobables) y dividido por la probabilidad total.

Dado que este cálculo se realiza para las dos clases por cada semilla de prueba, ignoramos la división entre la probabilidad total ya que la proporción se mantiene y si una probabilidad es mayor que otra no afecta el multiplicar o dividir ambas probabilidades por un escalar.

Finalmente nuestra clasificación depende simplemente de la distribución de cada clase, que está dada por la siguiente función.

$$p(x|y=0) = \frac{1}{(2\pi)^{1/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)}$$

$$p(x|y=1) = \frac{1}{(2\pi)^{1/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)}$$

Donde Σ es la matriz de covarianza de los datos y μ es el vector de medias de cada clase.

$$\Sigma = \sum_{i=1}^{m} (x^i - \mu_{y^i})(x^i - \mu_{y^i})^T$$

La codificación fue sencilla debido a las características de Octave, que puede manejar vectores y matrices con facilidad.

```
clases = datos(71:end, :);
clase1 = datos(71:140,:);
clase2 = datos(140:end,:);
media1 = mean(clase1);
media2 = mean(clase2);
```

Figura 16 Clases y vector de medias de cada clase.

Primero se tomaron las dos clases y se obtuvo su vector de medias y se calculó la matriz de covarianza en un ciclo for; aunque el software tiene su propia función para obtener la matriz de covarianza se decidió obtenerla manualmente.

```
for i = 1 : length(clases)
  if clases(8) == 2
    covarianza = covarianza + (clases(i,1:7) - media1(1:7))'*(clases(i,1:7) - media1(1:7) );
  else
    covarianza = covarianza + (clases(i,1:7) - media2(1:7))'*(clases(i,1:7) - media2(1:7) );
  end
end
```

Figura 17 Cálculo de la matriz de covarianza.

Con estos datos se generaron las funciones de distribución normal de cada clase.

```
normal1 = (1/(((2*pi)^(n/2))*sqrt(det(covarianza))))*exp((-1/2)*(x - media1)*inv(covarianza)*(x - media1)');
normal2 = (1/(((2*pi)^(n/2))*sqrt(det(covarianza))))*exp((-1/2)*(x - media2)*inv(covarianza)*(x - media2)');
```

Figura 18 Distribución normal de cada clase.

El último paso fue que cada dato de semilla se evaluó en cada función normal, y dependiendo la que resulte con mayor probabilidad fue clasificada.

Ya que el algoritmo fue validado con los mismos datos con los que fue entrenado se tuvo un 100% de efectividad, clasificando ambas clases de semillas de manera correcta y sin ningún error.

Análisis de Ada Boost

Este es un Meta-Algoritmo que utiliza clasificadores que tienen una precisión muy baja pero lo suficientemente buenas para aportar un poco de conocimiento sobre lo que se quiere clasificar.

Uniendo varios clasificadores débiles podemos crear uno fuerte que obtiene una precisión superior a sus partes.

El algoritmo añade un clasificador débil en cada iteración, tratando de mejorar el desempeño contra los datos que fueron mal clasificados.

En cuanto al desempeño con los datos obtuvimos un 100% de efectividad, esto es debido a que las clases que fueron escogidas para este algoritmo pueden ser divididas de manera perfecta en algunas de sus dimensiones.

También podemos decir que siendo algoritmos de aprendizaje supervisado, para ser probados deberían utilizarse datos distintos a los usados para entrenarles.

El Ada Boost resulta relativamente sencillo de entender a nivel funcional, pero es más complejo de programarse, ya que requerimos poder crear los clasificadores que se usaran para formar parte del clasificador fuerte.

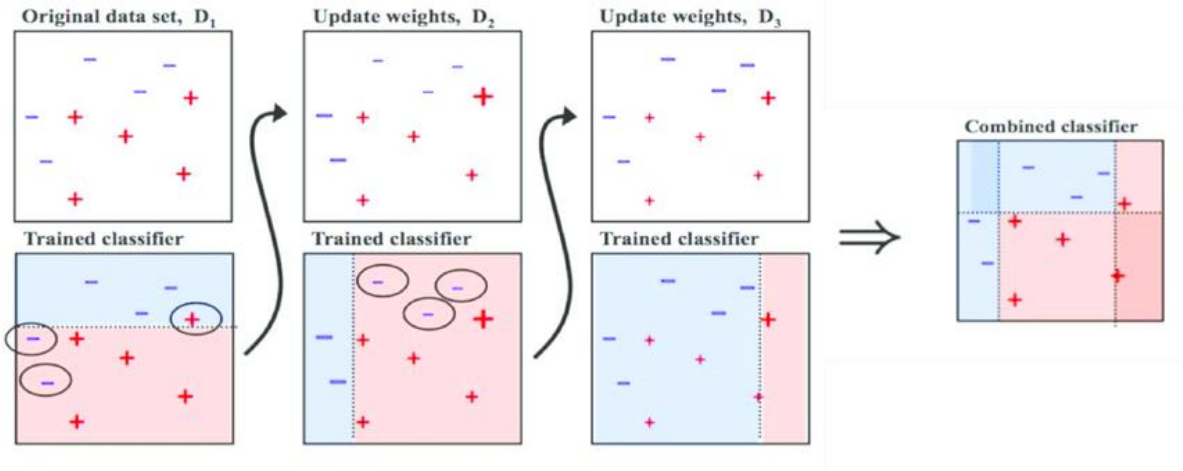

Figura 19 Funcionamiento del Ada Boost

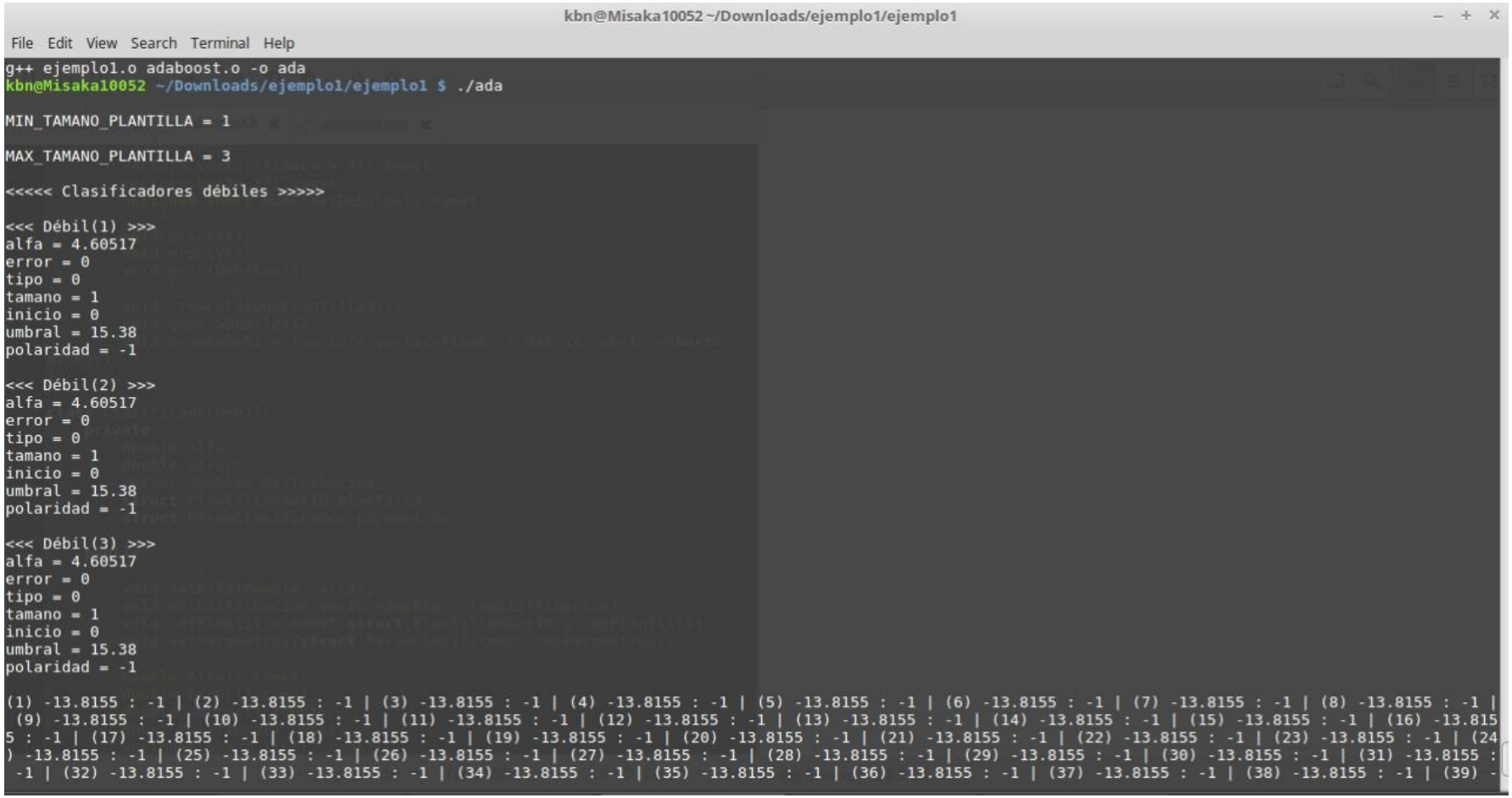

Figura 20 Resultado de Clasificadores Débiles

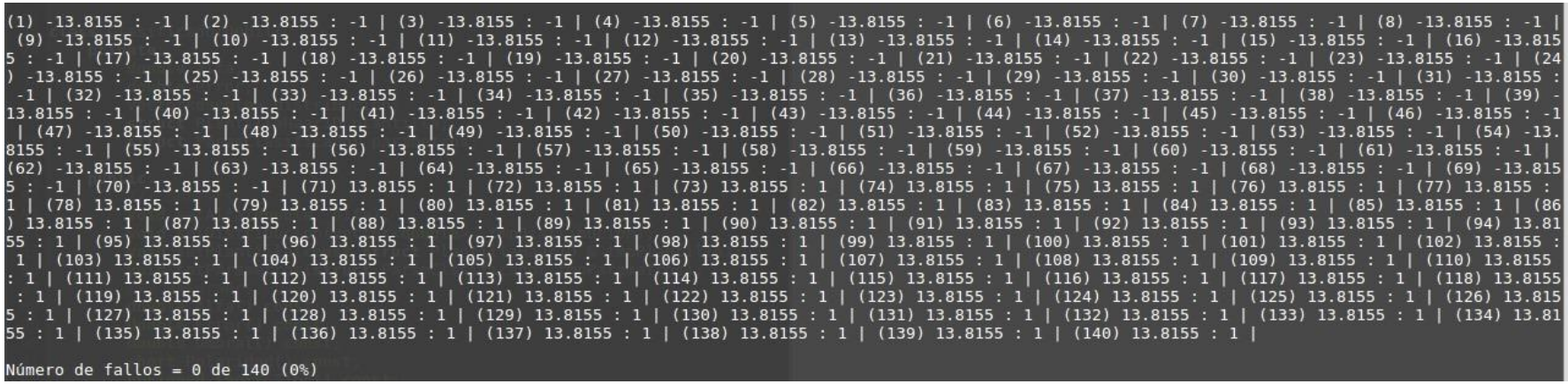

Figura 21 Número de fallos