La idea detrás de este algoritmo es tratar al conjunto de datos como una muestra de alguna distribución de probabilidad y aproximar su función de densidad probabilística. Luego ejecutar un proceso de gradiente ascendente hasta encontrar las modas de la función.

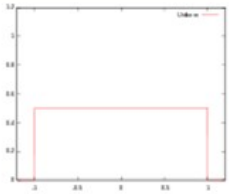
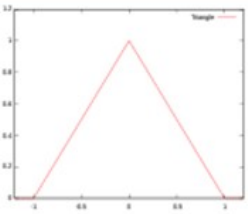
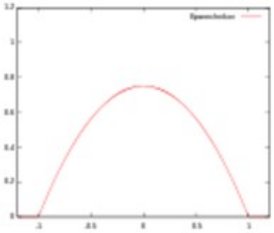
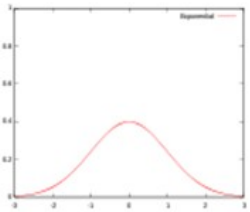
La aproximación de la función de densidad probabilística se hace por medio de la Estimación de densidad Kernel o KDE. (Kernel Density Estimation)

Aunque no es nuestra intención hacer un estudio detallado de la matemática detrás del algoritmo, hagamos algunas introducciones para que se entienda su funcionamiento.

¿Qué es un Kernel? Un Kernel, es una función que toma valores reales, es integrable y no negativa. Además debe satisfacer los siguientes requerimientos:

- Normalización: $\int_{-\infty}^{+\infty} K(u)\,du = 1$
- Simetría: $K(u) = K(-u) \forall u$

Hay varios Kernels muy usados, por ejemplo:

Kernel	Forma del Kernel
Kernel uniforme: $K(u) = \frac{1}{2}$,para $\|u\| \leq 1$	
Kernel Triangular: $K(u) = (1 - \|u\|)$, para $\|u\| \leq 1$	
Kernel Epanechnikov: $K(u) = \frac{3}{4}(1 - u^2)$ (parabolic)	
Kernel Gaussian: $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$	

Aunque muchos Kernel, nos interesa, para explicar el funcionamiento de este método los que mostramos en la tabla anterior.

Sabiendo que es un Kernel y cómo lucen, tanto matemáticamente como desde la forma de la función, estamos en condiciones de entender la manera que se encuentra la función de densidad probabilística a partir de nuestro conjunto de datos.

Para esto, supongamos que nuestro dataset está representado por {x1, x2, …,xn} , entonces la función de densidad probabilística se calcula basada en estos casos, de la siguiente forma:

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) (1)$$

Donde h es llamado bandwidth.

Para explicar esto y desarrollar la intuición de que significa esto, veamos la siguiente imagen:

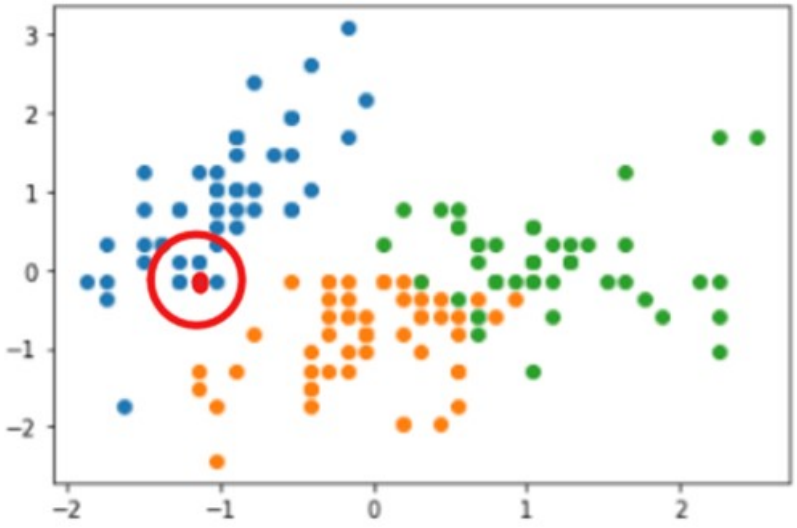


Figura 12. Ejemplo de la interpretación geométrica de la aplicación de un Kernel a un conjunto de datos.

En la figura 12, se muestra un conjunto de datos representados en el plano bidimensional. El círculo rojo representa, aproximadamente, como debe imaginarse la aplicación de la función (1) a n conjunto. El diámetro del círculo está determinado por el parámetro h. Y se le llama vecindad del punto x.

Esto quiere decir que para el cálculo de la función de densidad, basado en nuestro dataset, se aplica un círculo como este y aplica usando los casos que caen dentro de este. Luego se desplaza al otro punto y se calcula de nuevo. La suma de todos los Kernel da como resultado la función de densidad.

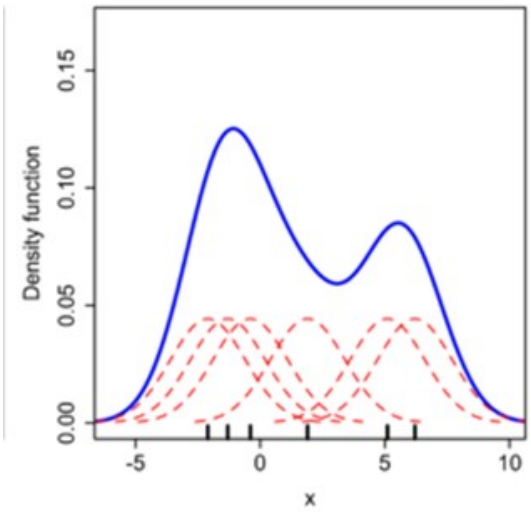


Fig.13. Función de densidad probabilística obtenida por KDE. Usando 6 datos. Cada una de las campanas discontinuas pertenece a la aplicación de un Kernel a un punto.

En la figura 13 se da otra perspectiva de cada uno de los Kernel independientes, distinguidos por líneas discontinuas rojas. Es importante notar que esta curva gausiana tiene media en el dato x. La función de densidad, entonces, se obtiene sumando las contribuciones individuales.

En la siguiente imagen, se muestra una interpretación geométrica de cómo debe, intuitivamente imaginarse una función de densidad en un espacio tridimensional.

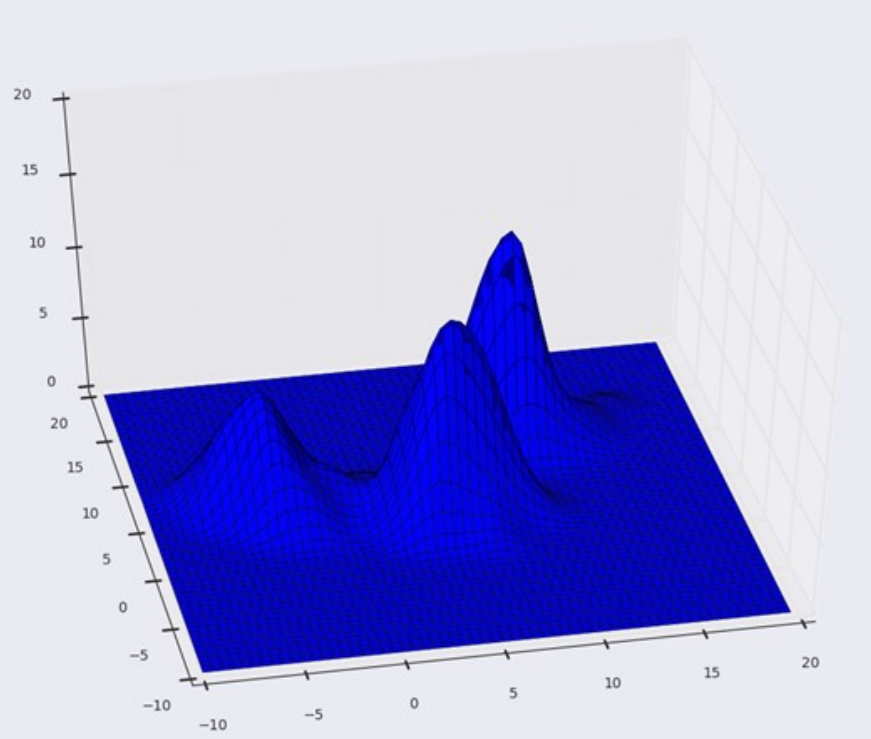


Figura 14. Ejemplo de una Función de Densidad Probabilística, con tres modas o máximos.

Es en la cresta de estos picos donde se da la máxima densidad de casos, es decir, los casos están más agrupados alrededor de estos puntos.

El cálculo de esta función es solo el primer paso en el funcionamiento de este método. Ahora, lo que se hace, es que se calcula el gradiente de cada uno de los puntos o casos, basado en la función de densidad, de la siguiente forma:

$$\nabla f(x) = \frac{2C_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left\| \frac{x - x_i}{h} \right\|^2 \right) \right] \left[\frac{\sum_{i=1}^{n} x_i g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right)}{\sum_{i=1}^{n} g\left(\left\| \frac{x-x_i}{h} \right\|^2 \right)} - x \right]$$

Donde, h es el ancho de banda o bandwidth que especifica el ancho de la ventana del Kernel, d es el número de rasgos de nuestro dataset, g es –K'(x) y es la derivada del Kernel. El primer término, si se dan cuenta, es proporcional al estimado de la función de densidad en el punto x, mientras que el segundo es el vector de la media, que siempre apunta en la dirección de máximo incremento del gradiente y es

denotado por m.

Pues el método MeanShift, ahora lo que hace es:

- Se elige un punto y se calcula el vector de la media m(x).
- Se traslada la ventana de estimación de la densidad para otro punto x donde el gradiente sea mayor y se calcula de nuevo m(x).
- Se repite hasta que se encuentra un punto x para el cual el gradiente es cero. En ese punto se dice que el algoritmo converge.

Veamos cómo se aplica en la práctica, usando el módulo Scikit-learn de Python, y luego veremos los parámetros que acepta.

```
#Clustering con Mean Shift
from sklearn.cluster import MeanShift
from sklearn.cluster import  estimate_bandwidth
import matplotlib.pyplot as plt
import statistics as stat
from numpy import unique
from numpy import where
from sklearn import datasets
from itertools import cycle
from sklearn.preprocessing import StandardScaler
# Cargamos el Iris Dataset. Tres clases, 150 casos
# 50 casos por clase, todos los atributos numéricos
idf= datasets.load_iris()
sc = StandardScaler()
X = sc.fit_transform(idf.data)
# Definimos el modelo, le pasamos como parámetro
# bandwidth y bin_seeding
bandwidth = estimate_bandwidth(X, quantile=0.2, n_samples=150)
model = MeanShift(bandwidth=bandwidth, bin_seeding=True)
# Hacemos el clustering y retornamos el clúster
# asignado a cada caso
yhat = model.fit_predict(X)
# Recuperamos los lusters
clusters = unique(yhat)
# Vamos a mostrar un Scatterplot para cada clúster
for cluster in clusters:
    print("Cluster:{0} cantidad de casos:{1}".format(cluster,len(yhat[yhat==cluster])))
    row_ix = where(yhat == cluster)
    plt.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
plt.show()
```

Figura 15. Ejemplo de uso de MeanShift, calculando el Bandwidth explícitamente. Esto no es estrictamente necesario, pues de no pasarle el parámetro o pasarle None, la misma implementación lo hace.

El resultado de la corrida de este ejemplo, se muestra en la siguiente imagen.

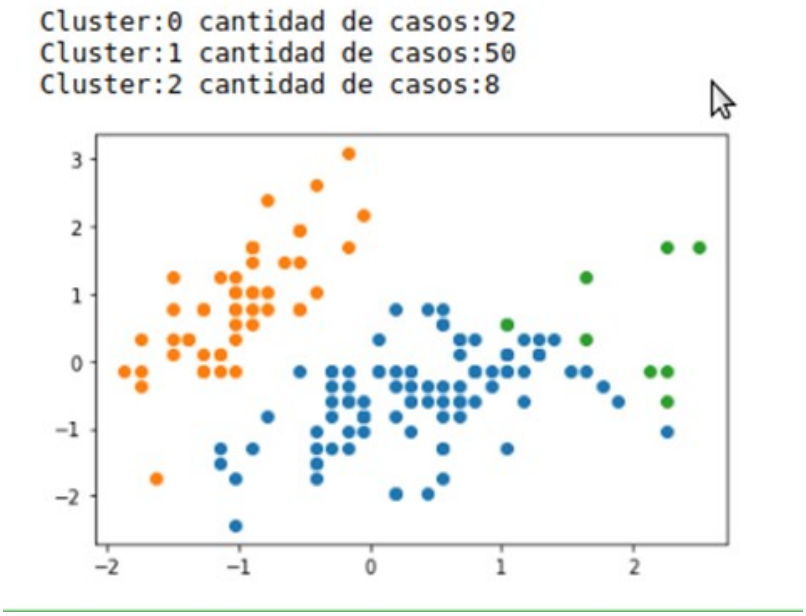


Figura 16. Resultados de la corrida de Mean Shift, sobre el Iris Dataset.

por cada una de las clases y que la corrida de MeanShift distribuye 92 casos a un solo clúster ya nos habla de que necesitamos reconsiderar los parámetros que les estamos pasando.

Veamos los parámetros necesarios para correr MeanShift y su significado.

Parámetros:

bandwidth: Espera un valor de tipo float y el valor por defecto en None. Este parámetro define el tamaño de la ventana para la estimación de KDE. Si se pasa como valor None o no se pasa ningún valor, se usa sklearn.cluster.estimate_bandwidth para calcularlo.

Este es el parámetro que define el funcionamiento del algoritmo, por tanto no veremos otros y es necesario que consulten la documentación.

Esta implementación retorna:

- cluster_centers_: Es un ndarray con forma (n_clusters, n_featires) y contiene las

coordenadas de los centros de los clústers.

- labels_: contiene un ndarray, con las etiquetas asignadas a cada uno de los casos.

Spectral clustering

Este tipo de método ha tenido un significativo éxito en la práctica al ser capaz de tener buenos resultados donde otros fallan. La idea detrás de este tipo de método de aprendizaje es ver al conjunto de datos como un grafo, donde cada caso es visto como un vértice que están unidos por medio de vértices que denotan su similaridad o distancia.

Por tanto, dado un conjunto de datos $\{x_1,x_2,\ldots,x_n\}$ y una métrica de afinidad o similaridad, como puede ser la norma, L1, L2 o euclidiana, entonces podemos construir un grafo G, tal que:

$$G=\{V,E\}$$

G es un conjunto ordenado de vértices y de aristas o arcos, donde los vértices corresponden a casos en el conjunto de casos y los arcos son las similaridades entre ellos. En la Fig. 17, se muestra como se vería un grafo de este tipo.

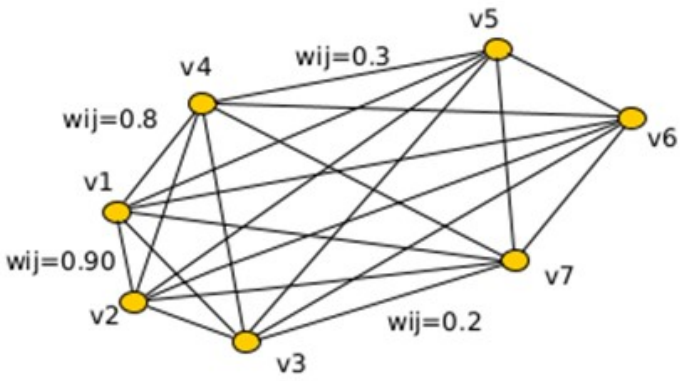


Figura 17. Ejemplo de Grafo completamente conectado de 7 vértices. Los wij representa la similaridad entre vértices.

Entonces:

$V = \{x_1, x_2, \ldots, x_n\}$ es el conjunto de vértices que representan los datos.

$E = \{mij\}$ es el conjunto de todos los pesos de los arcos que conectan dos vértices.

Asumiremos que el grafo es ponderado y que los pesos de los arcos que unen los vértices, tienen pesos no negativos.

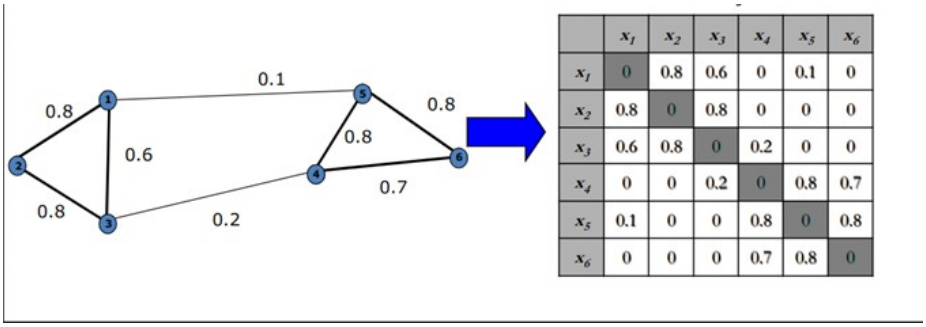
El grado de un vértice, es la suma de los pesos de los arcos que lo tienen como un extremo. De esta forma, el grado d del vértice i, es:

$$d_i = \sum_{j=0}^{n} w_{ij}$$

Hay varias maneras en las que podemos construir un grafo de similaridad. Las más tratadas en la literatura son:

- Grafo ε-vecindad o ε-neighborhood: En este tipo de grafo, solo se incluyen los vértices y cuyos arcos tienen peso menor o mayor que ε.
- Grafos de vecindad de los k-vecinos cercanos: En este se incluyen los k vértices que son vecinos cercanos de del vértice vi.
- Grafo completamente conectado: En este tipo de grafo se incluyen todos los vértices y los pesos que reflejan su similaridad o distancia.

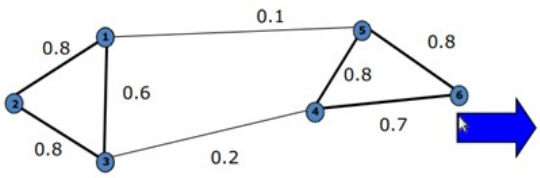
Un grafo, tiene asociada una matriz de afinidad, distancia o similaridad que contiene las similaridades entre cada uno de los vértices. Normalmente se representa como W y luce como el Figura 18.



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	0	0.8	0.6	0	0.1	0
x_2	0.8	0	0.8	0	0	0
x_3	0.6	0.8	0	0.2	0	0
x_4	0	0	0.2	0	0.8	0.7
x_5	0.1	0	0	0.8	0	0.8
x_6	0	0	0	0.7	0.8	0

Figura 18. Matriz de similaridad de un Grafo.

También, tiene asociada una Matriz llamada, matriz de grado (degree) y denotada por D, en la cual los valores de la diagonal son los grados de cada uno de los vértices del grafo G. Un ejemplo de esta matriz, se muestra en la figura 19.



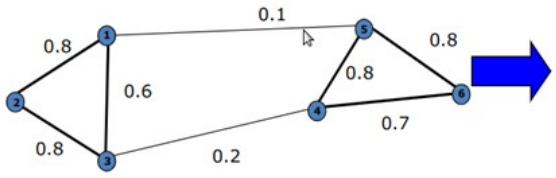
	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	0	0	0	0	0
x_2	0	1.6	0	0	0	0
x_3	0	0	1.6	0	0	0
x_4	0	0	0	1.7	0	0
x_5	0	0	0	0	1.7	0
x_6	0	0	0	0	0	1.5

Figura 19. Matriz de grado, del grafo G.

Usando estas dos matrices anteriores se construye otra, llamada Laplaciano o matriz Laplaciana del grafo G y se denota por L. La matriz L se obtiene de la siguiente forma:

$$L = D - W$$

En la fig. 20 mostramos el Laplaciano de las matrices que hemos usado en las dos imágenes anteriores.



	x_1	x_2	x_3	x_4	x_5	x_6
x_1	1.5	-0.8	-0.6	0	-0.1	0
x_2	-0.8	1.6	-0.8	0	0	0
x_3	-0.6	-0.8	1.6	-0.2	0	0
x_4	0	0	-0.2	1.7	-0.8	-0.7
x_5	-0.1	0	0	-0.8	1.7	-0.8
x_6	0	0	0	-0.7	-0.8	1.5

Figura 20. Ejemplo de un Laplaciano, calculado a partir de las dos matrices de la imagen 18 y 19. W y D.

Cada Matriz, tiene unos valores y unos vectores asociados a estas que la identifican de manera única. A estos valores y vectores se les llaman valores o vectores propios. Normalmente en la literatura se les llaman Eigenvalues y Eigenvector y el término «Eigen» se puede entender, aproximadamente como «propio». Por tanto a estos, también, se les llama valores o vectores propios de la matriz.

Los valores propios de una matriz, son aquellos que satisfacen la ecuación característica de ella.

Si tenemos una matriz $A \in R^{n*n}$, entonces, al conjunto de raíces de la ecuación:

$$det(A - \lambda I) = 0$$

Se llaman valores propios de A.

Para cada valor propio de una matriz A hay, asociado con este, un vector que se le llama vector propio de A si,

$$A\overrightarrow{x} = \lambda\overrightarrow{x}$$

Donde $\overrightarrow{x}$ es un vector propio de A y λ el valor propio asociado con él.

De esta forma, al conjunto de vectores propios del Laplaciano del grafo G se le denomina Espectro de G. Hay todo un campo de investigación llamado Teoría

espectral de grafos que se dedica al estudio de estos valores y los métodos para su obtención.

Con todo lo anterior explicado, podemos ahora, a grandes rasgos, claro está describir en que consiste el algoritmo de clustering espectral.

Spectral Clustering:

Entrada: $\{x_1,x_2,\ldots,x_n\}$ y el número k de clúster.

- Construir una matriz de similaridad a partir de la entrada, S.
- Encontrar la matriz Laplaciana de S, llamada L.
- Encontrar, los k valores y vectores propios de L.
- Construir una nueva matriz, usando los vectores propios como columnas.
- Interpretando las filas de la nueva matriz, como vectores en un nuevo espacio vectorial R^{n*k} aplicar un algoritmo de clustering K-Means.

Lo más interesante aquí, la intuición detrás del funcionamiento de este método es que proyecta nuestro conjunto de datos a un nuevo espacio vectorial donde puede realizarse la separación de los clúster de una manera más simple.

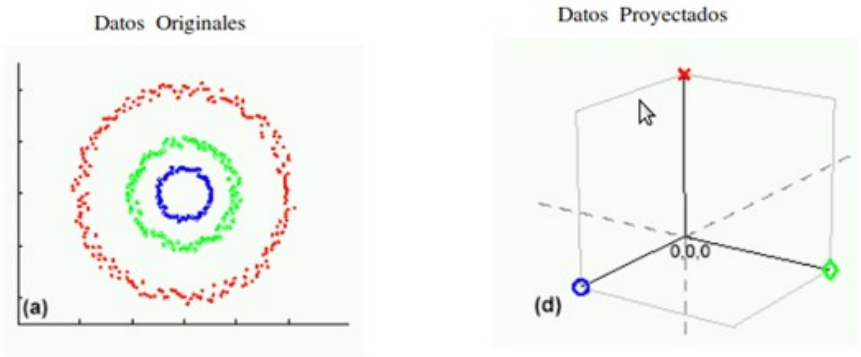


Figura 21. Este método proyecta los datos originales y luego aplica el clustering.

A grandes rasgos, esta es la idea básica detrás de este método de clustering. Veamos ahora como se usa la implementación hecha en Python y que resultados arroja al aplicarlo al dataset iris. Es importante recordar que este dataset, contiene 150 casos, divididos en tres clases y 50 casos en cada una de las clases. Eso nos da una medida, de manera intuitiva de cuan bien puede separar un modelo, los casos del dataset, hasta que en la próxima lección tratemos los métodos de evaluar los resultados de estos métodos.

```
#Clustering con Clustering Espectral
from sklearn.cluster import SpectralClustering
import matplotlib.pyplot as plt
from numpy import unique
from numpy import where
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
# Cargamos el Iris Dataset. Tres clases, 150 casos
# 50 casos por clase, todos los atributos numéricos
idf= datasets.load_iris()
sc = StandardScaler()
X = sc.fit_transform(idf.data)
# Definimos el modelo, le pasamos como parámetro
# el número de clúster que sabemos de antemano que es 3
model = SpectralClustering(3)
# Hacemos el clustering y retornamos el clúster
# asignado a cada caso
yhat = model.fit_predict(X)
# Recuperamos los lusters
clusters = unique(yhat)
# Vamos a mostrar un Scatterplot para cada clúster
for cluster in clusters:
    print("Cluster:{0} cantidad de casos:{1}".format(cluster,len(yhat[yhat==cluster])))
    row_ix = where(yhat == cluster)
    plt.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
plt.show()
```

Figura 22. Ejemplo de aplicación de Clustering Espectral, sobre el iris Dataset.

Los resultados de la aplicación de este método al iris dataset se muestran en la siguiente figura:

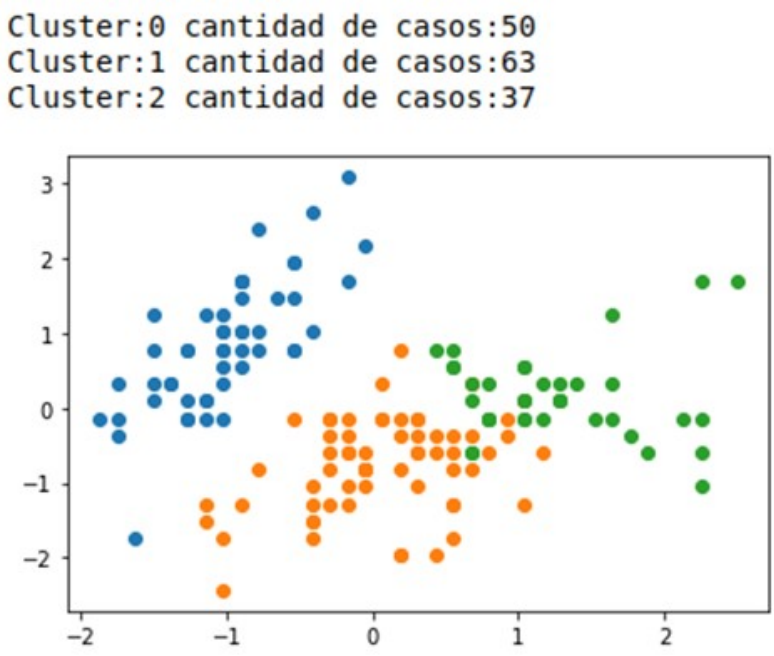


Figura 22. Resultados de la aplicación de Spectral Clustering al iris Dataset.

Veamos los parámetros que esta implementación toma.

Parámetros:

- n_clusters: Este parámetro define la cantidad de clústers en los que se dividirán los datos de entrada y es equivalente a la dimensión del subespacio de proyección. Es decir, especifica la cantidad de dimensiones que tendrá el subespacio. Espera valores enteros y el valor por defecto es 8.
- eigen_solver: Especifica la estrategia para obtener los vectores propios. Este parámetro acepta tres valores: arpack, lobpcg, amg. El valor por defecto es None que usa arpack, como solver.
- n_components: Especifica la cantidad de vectores propios usar para el subespacio de proyección. Cuando no se especifica valor se usa n_clusters.
- affinity: Define el método por el cual se construye la matriz de afinidad o similaridad.

Los valores que acepta son:

- nearest_neighbors: Construye la matriz de afinidad a partir de un grafo de los k-vecinos cercanos.
- rbf: Construye la matriz de afinidad usando un Kernel de Base Radial. (RBF) Toma como coeficiente el valor del parámetro gamma.
- precomputed: Interpreta los datos de entrada X como una matriz de afinidad ya construida, donde los valores mayores significan similaridades más altas entre los casos.
- precomputed_nearest_neighbors: Interpreta los datos de entrada X como un grafo disperso de distancias calculadas de antemano y construye una matriz de afinidad binaria a partir del valor del parámetro n_neighbors de los vecinos más cercanos a cada instancia.

- n_neighbors: Espera valores enteros y por defecto toma el valor 10. Este parámetro especifica la cantidad de vecinos cercanos a usar cuando se usa nearest_neighbors como valor para el parámetro affinity. Es decir cuando se pretende construir la matriz de afinidad por el método ε-neighborhood.

Como hemos estado haciendo con los métodos anteriores, solo describimos aquellos parámetros que son los que definen la conducta del método, el resto de los parámetros deben consultarse en la documentación de la implementación.

Este método retorna los siguientes valores:

- affinity_matrix_: Después que se ejecuta el método fix(), esta variable contiene la matriz de afinidad o matriz de similaridad.
- labels_: Contiene las etiquetas de cada uno de los casos. Su tamaño es igual al número de casos que contiene el dataset.

Hasta aquí hemos visto los elementos esenciales y la idea detrás del clustering

espectral, en la siguiente sección veremos otros métodos de clustering o métodos de aprendizaje no supervisado.

DBSCAN

Este método de aprendizaje automático se clasifica dentro de los métodos basados en densidad. La idea detrás de este método es identificar un clúster con un área densa que está separada de otros por un área menos densa.

Su nombre «Density-based spatial clustering of applications with noise», puede traducirse, más o menos, como «Agrupamiento basado en densidad espacial para aplicaciones con ruido» pero no dice mucho de las ideas que tiene detrás.

Este método ve la densidad como la cantidad de casos o puntos en un área determinada, donde la distancia entre ellos es menos que a la del resto del clúster. Este tipo de método no asume ninguna forma en el grupo de casos.

La densidad de cada punto se estima contando el número de puntos, que se encuentran a una distancia no mayor que Eps del punto. Dependiendo de dicho valor, cada punto es clasificado como central (core), frontera (border) o ruido (noise). Eps, significa épsilon.

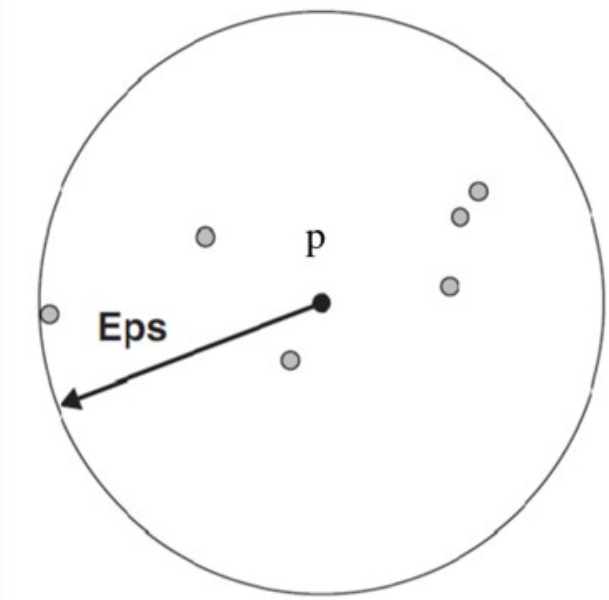


Figura 23. Eps-vecindad de un punto p.

A este círculo con centro p y radio eps se le llama vecindad del punto p. Esto nos lleva a la primera definición.

Definición #1: Eps-vecindad: Se llama eps-vecindad de un punto y denotada por $N_{eps}\,(p)$ y definida como sigue:

$$N_{eps}\,(p) = \{q \in D \vee dis\,(p,q) \leq eps\}$$

Esto se lee, la eps-vecindad de un punto p, son todos los puntos que pertenecen a D, el conjunto de datos, tal que la distancia entre p y q es menor que eps.

Sin embargo esto no explica completamente la manera en que un clúster queda definido, pues los puntos del borde tienen una eps-vecindad mucho menor que los del centro y por tanto habría que definir una cantidad mínima de puntos que defina la densidad de cada punto.

Definición #2: Directamente densidad-alcanzable: Un punto p es directamente densidad-alcanzable si:

$\forall p \in C, \exists q \vee p \in N_{eps}(q) \, y N_{eps}(q) \vee \geq MinP$

Esto se lee, el punto p es directamente densidad-alcanzable, si pertenece al clúster C y existe un punto q, tal que p pertenece a su eps-vecindad y esta tiene al menos MinP puntos.

Este tipo de relación es simétrica para los puntos centrales o core, del clúster, sin embargo no lo es si uno de los puntos es de frontera.

Teniendo en cuenta este MinP ahora podemos definir los tipos de puntos:

- **Punto central (core):** Un punto se llama central o core si su eps-vecindad contiene más de MinP puntos.
- **Punto de frontera:** Un punto es de frontera si no es un punto central; pero hay al menos un punto central dentro de su eps-vecindad.
- **Punto Ruido:** Un punto se considera Ruido si no es ni central ni frontera.

Definición #3: densidad-alcanzable: Se dice que un punto p es densidad-alcanzable desde un punto q si existe una cadena de puntos $p_1, p_2, \ldots, p_n$ donde $p_1 = q; p_n = p$ tal que p_{i+1} es directamente densidad-alcanzable desde p_i .

Definición #4: densidad-conectado: Un punto p es densidad-conectado a un punto q, si existe un punto o tal que, tanto p como q sean densidad-alcanzables desde o.

Ahora, con los conceptos anteriores se puede definir la noción de clúster basado en densidad.

Definición #5: Clúster: Si denotamos por D al conjunto de datos o dataset, un clúster C en D, es un subconjunto no vacío de D que satisface las siguientes propiedades:

- $\forall p, q \, si \, p \in C$ y q es densidad-alcanzable desde p, entonces $q \in C$.
- $\forall p, q \in C$: p es densidad-conectado a q.

Definición #6: Ruido: Hagamos que C={C1 , C2,…,Ck} sea el conjunto de clúster del dataset D, atendiendo a los parámetros eps y MinP, entonces se puede definir el ruido como el conjunto de puntos en D que no pertenecen a ningún Ck clúster.

Es decir, el ruido puede definirse como:

$$ruido = \{p \in D \vee p \notin C_k\}$$

Esto, en palabras llanas significa que se considera ruido al conjunto de puntos que no pertenecen a ningún clúster.

Algoritmo DBSCAN

- Etiquetar todos los puntos como central, frontera o ruido.
- Eliminar los puntos ruido.
- Unir los puntos centrales que estén a una distancia menor que Eps.
- Cada grupo de puntos centrales conectados forma un clúster.
- Asignar cada punto frontera al clúster de uno de sus puntos centrales.

De manera general, hemos descrito cómo se realiza el proceso de agrupamiento por medio de DBSCAN. Claro, no hemos hablado de la manera de escoger los valores de eps y MinP; sin embargo este tema no lo veremos en esta lección.

En la siguiente imagen veremos un ejemplo de aplicación de DBSCAN al Iris Dataset, usando la implementación de este en scikit-learn de Python.

```python
#Clustering DBSCAN
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
from numpy import unique
from numpy import where
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
# Cargamos el Iris Dataset. Tres clases, 150 casos
# 50 casos por clase, todos los atributos numéricos
idf= datasets.load_iris()
sc = StandardScaler()
X = sc.fit_transform(idf.data)
# Definimos el modelo, le pasamos como parámetro
# eps y min_samples
model = DBSCAN(eps=0.68, min_samples=15)
# Hacemos el clustering y retornamos el clúster
# asignado a cada caso
yhat = model.fit_predict(X)
# Recuperamos los clusters
clusters = unique(yhat)
# Vamos a mostrar un Scatterplot para cada clúster
for cluster in clusters:
    print("Cluster:{0} cantidad de casos:{1}".format(cluster,len(yhat[yhat==cluster])))
    row_ix = where(yhat == cluster)
    plt.scatter(X[row_ix, 0], X[row_ix, 1])
# show the plot
plt.show()
```

Figura 24. Ejemplo de aplicación de DBSCAN al Iris Dataset.

```
Cluster:0 cantidad de casos:44
Cluster:1 cantidad de casos:45
Cluster:2 cantidad de casos:61
```

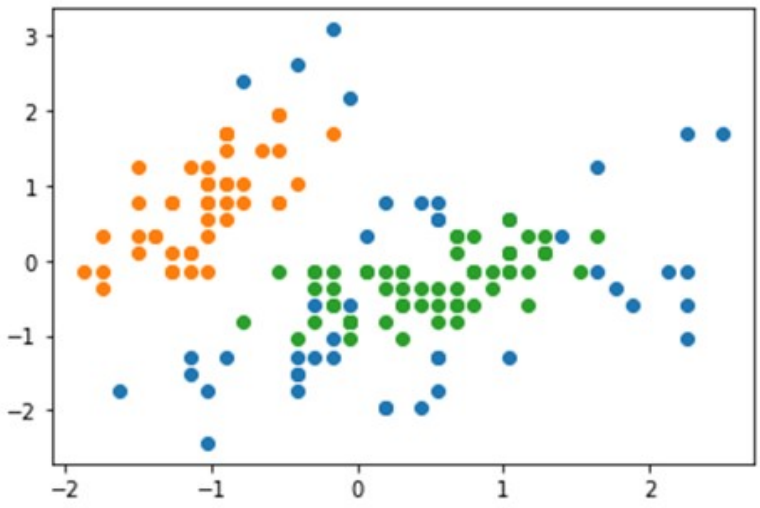


Figura 25. Resultados de la aplicación de DBSCAN al Iris Dataset.

Es importante que tengamos en cuenta que los parámetros son escogidos de manera empírica y no siguiendo ninguno de los métodos propuestos para ello.

Veremos ahora los parámetros o argumentos necesarios para utilizar el algoritmo. Como siempre, en esta sección solo veremos aquellos parámetros que son más importantes para la operación del algoritmo, dejando el resto para que sean consultados en la documentación del método en caso de ser necesario.

Parámetros:

- eps: Este parámetro espera un valor de tipo float y el valor por defecto es 0.5. Este es el principal parámetro de DBSCAN y define el tamaño de la ε-vecindad . Puede interpretarse como la distancia máxima entre dos casos o puntos para que una se considere en la vecindad de la otra.

- min_samples: Espera valores enteros y el valor por defecto es 5. Representa el MinP que hemos descrito antes y define el número mínimo de casos o puntos en la vecindad para que un punto sea considerado como central o core.

- metric: Espera un valor de cadena o string y el valor por defecto es 'euclidean'. Define la métrica que se va a usar para calcular la distancia entre puntos. Puede tomar los valores siguientes:

  - scikit-learn: 'cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan'.

  - scipy.spatial.distance: 'braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule'

Retorna:

- core_sample_indices_: Contiene los índices de los puntos core, que se encontraron.

- labels_: Contiene las etiquetas de todos los puntos o casos, asignadas por DBSCAN.

- Hasta aquí hemos revisado, aunque de manera muy general el método de aprendizaje no supervisado DBSCAN, no entramos en detalles sobre el algoritmo;

pero, si se necesita entender más profundamente es necesario estudiar con detalles la documentación disponible que es abundante.

La mayoría de los métodos de aprendizaje no supervisado, tal como K-means y otros, no son muy eficientes a la hora de tratar con grandes conjuntos de datos. Estos no escalan bien en términos de tiempo de corrida y uso de memoria en la medida que los conjuntos de datos aumentan su tamaño y aquí es donde entra en escena el algoritmo BIRCH, que estudiaremos brevemente en la siguiente sección.

BIRCH

Estas siglas significan «Balanced Iterative Reducing and Clustering using Hierarchies» y podría traducirse, más o menos como «Agrupamiento y reducción iterativa equilibrada usando jerarquías». Aunque el nombre no arroja mucha luz sobre la forma en que se ejecuta el clustering, lo cierto es que da algunas pistas.

Una de las características interesantes y únicas de este método es que no hace el agrupamiento directamente sobre el conjunto de datos, sino que hace una especie de sumario de este, sobre el que luego puede aplicarse cualquiera de los algoritmos ya estudiado para realizar el clustering. Esto es especialmente importante cuando se está trabajando un dataset grande, tanto en número de casos como de rasgos.

Este, se puede categorizar como un algoritmo de clustering jerárquico y solo necesita un paso sobre el dataset para encontrar buenos clústers.

La idea detrás de este método es capturar la estructura del dataset y sumarizarla en un árbol que se da en denominar árbol CF. La estructura central sobre la que descansa la construcción de dicho árbol, es llamada CF o Clustering Feature y contendrá información sobre un determinado grupo de puntos. Así que es importante que definamos que se entiende por **Clustering Feature.**

Definición #1: Clustering Feature (CF):

Si X es un conjunto de datos tal que $X = \{x_1, x_2, \ldots, x_n\}$ entonces Clustering Feature (CF) es una estructura triple:

$$CF = (N, \overrightarrow{LS}, SS)$$

Donde N, es el número de puntos de datos en el clúster que trata de caracterizar el CF.

LS, por otra parte, es la suma lineal de todos los N puntos de X, calculada como sigue:

$$\overrightarrow{LS} = \sum_{i=1}^{N} \overrightarrow{x_i}$$

Y SS, sería la suma del cuadrado de los x de X, tal como mostramos a continuación:

$$SS = \sum_{i=1}^{N} (\overrightarrow{x_i})^2$$

Cada CF, caracteriza a un clúster del conjunto de datos X.

Habiendo definido en que consiste la estructura central que soporta la idea básica de este algoritmos, es necesario que definamos las maneras en que se miden las distancias y se calculan los elementos y luego que exploremos la manera en que se construye el árbol y por tanto la manera en que se ejecuta el agrupamiento.

Para entender la forma en que procede este algoritmo hay que definir algunos conceptos como el del Centroide, el Radio y el Diámetro de un Clúster, entonces:

Definición #2: Centroide de un Clúster $\overrightarrow{X_0}$: Se considera el punto central de un clúster y se obtiene de la siguiente forma:

$$\overrightarrow{X_0} = \frac{1}{N} \sum_{i=1}^{N} \overrightarrow{x_i}$$

Definición #3 Radio de un Clúster R: Se puede considerar como la distancia media entre los puntos que pertenecen al clúster y el centroide $\overrightarrow{X_0}$ y se puede obtener por medio de la siguiente expresión:

$$R = \left[\frac{1}{N} \sum_{i=1}^{N} (\overrightarrow{x_i} - \overrightarrow{x_0})^2 \right]^{\frac{1}{2}}$$

Definición #4 Diámetro de un Clúster D: Esta medida puede considerarse como la suma de las distancias, tomados dos a dos, entre todos los puntos de un clúster y se puede definir como sigue:

$$D = \left[\frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1}^{N} (\overrightarrow{x_i} - \overrightarrow{x_j})^2 \right]^{\frac{1}{2}}$$

Definición #5 Distancia Euclidiana entre dos centroides $\overrightarrow{X_0}$ y $\overrightarrow{X_1}$, que llamaremos D_0 :

$$D_0 =$$

Definición #6 Distancia Manhattan entre dos centroides $\overrightarrow{X_0}$ y $\overrightarrow{X_1}$, que llamaremos D_1 :

$$D_1 = \left| \overrightarrow{x_0} - \overrightarrow{x_1} \right|$$

También es necesario que definamos otro grupo de medidas que son necesarias para entender cómo se construye el árbol CF de este método.

Para ello supongamos que tenemos un par de conjuntos de puntos o clústers $\overrightarrow{X_i}$ y $\overrightarrow{X_j}$, donde $i = 1,2,\ldots,N_1$ y $j = N_1 + 1, N_1 + 2, \ldots, N_1 + N_2$, entonces podemos definir:

Definición #7: Distancia media inter-clúster, que llamaremos D2:

$$D_2 = \left[\frac{1}{N_1 N_2} \sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\overrightarrow{x_i} - \overrightarrow{x_j})^2 \right]^{\frac{1}{2}}$$

Definición #8: Distancia media intra-clúster, que llamaremos D3:

$$D_3 =$$

Definición #8: Distancia Incremento de la Varianza, que llamaremos D4:

$$D_4 = \sum_{k=1}^{N_1+N_2}$$

Si consideremos cada CF como un vector, entonces, se puede definir la forma en que estos se pueden sumar.

Teorema #1: Aditividad de los CF: Si asumimos que $CF_1 = (N_1, \overrightarrow{LS_1}, SS_1)$ y $CF_2 = (N_2, \overrightarrow{LS_2}, SS_2)$ son vectores de dos clústers disjuntos, entonces el vector resultante de la suma de estos vectores es:

$$CF_1 + CF_2 = (N_1 + N_2, \overrightarrow{LS_1} + \overrightarrow{LS_2}, SS_1 + SS_2)$$

Veamos ahora que aspecto tendría un árbol CF y cómo se construye y de paso, vemos la manera que se hace el agrupamiento.

Bueno, lo primero es que este árbol, tiene asociado dos parámetros que definen la forma de este y son: El factor de ramificación, que llamaremos B y el umbral, al cual llamaremos T.

Teniendo en cuenta esto, es preciso que definamos lo que llamaremos Regla T y que de alguna forma define el tamaño del árbol.

Definición #9: Regla T: Llamaremos Regla T al hecho de que ninguna de las entradas en una hoja puede tener radio o diámetro mayor que T. Donde T es el

parámetro T o umbral.

Cuanto T es menor, el diámetro o radio de las entradas es menor y por tanto absorben menos puntos y son necesarias más entradas, por tanto el árbol crece.

Un árbol CF, tiene un nodo raíz, que contendrá uno o varios CF. También tendrá uno o varios nodos ramas o noleaf y uno o varios nodos hojas. En la Fig. 26, se muestra una representación de cómo luciría un árbol CF. Lo que está por encima de la línea es el dataset y lo que está por debajo de ella es la abstracción que hace el algoritmo de los datos reales.

Es importante que quede claro que el algoritmo no trae a la memoria los datos reales, sino sus características que sumariza de manera implícita en cada uno de los CF.

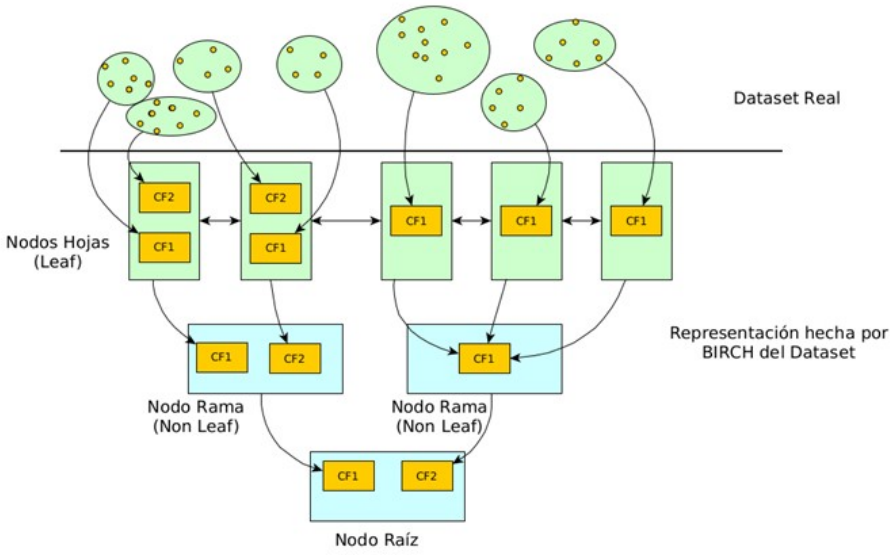


Figura 26. Representación de como luciría un árbol CF construido por BIRCH.

El nodo raíz y los nodos ramas, no pueden contener más de B, entradas, recordar que B es el factor de ramificación y es un parámetro de entrada al algoritmo. Los nodos hojas no pueden contener más de L entradas y estas deben cumplir la Regla T. L es el tamaño de las hojas o nodos hojas.

Es muy sencillo calcular las características de un clúster a partir de la información contenida en cada CF, tal que:

Dado un $CF = (N, \overrightarrow{LS}, SS)$, entonces el Centroide, el Radio y el Distancia media intra-clúster serían:

$$C = \frac{\overrightarrow{LS}}{N}$$

$$R = \sqrt{\frac{SS}{N} - (\frac{\overrightarrow{LS}}{N})^2}$$

$$D_2 = \sqrt{\frac{N_1 SS_2 + N_2 * SS_1 - 2 * \overrightarrow{LS}_1 * \overrightarrow{LS}_2}{N_1 N_2}}$$

Veamos cómo sería el algoritmo general que usa este método para realizar el agrupamiento.

$$D_2 = \sqrt{\frac{N_1 SS_2 + N_2 * SS_1 - 2 * \overrightarrow{LS}_1 * \overrightarrow{LS}_2}{N_1 N_2}}$$

Algoritmo BIRCH:

- Fase 1. Construir el árbol CF.
- Fase 2. Condensar el árbol a un rango requerido. (opcional)
- Fase 3. Agrupamiento Global.
- Fase 4. Refinamiento del agrupamiento. (opcional)

Fase 1: Construir el árbol CF:

- Crea e inicializa un árbol t_0 en memoria con un pequeño valor T.

- Comienza a insertar puntos en el árbol.(Ver el algoritmo de inserción)
- Si la memoria se desborda antes de que todos los puntos sean insertados, se crea un nuevo árbol t_1 con un T mayor. Se reinsertan todos los puntos que estaban en el anterior y se continúan insertando los nuevos.
- Opcionalmente se pueden chequear outliers mientas se construye el árbol:
  - Si la entrada de la hoja, contiene mucho menos puntos que la media se escriben a disco en vez de añadirlas al árbol.
  - Si el espacio en disco se acaba, se exploran todos los outliers y se reabsorben aquellos que no aumenten el tamaño del árbol.
- Si la memoria no se desborda, se ha terminado el proceso y se pasa a la Fase siguiente.

Fase 2: Condensar el árbol a un rango requerido: Aunque esta etapa es opcional, también puede ser muy útil, pues se puede utilizar como un puente entre la salida de la primera fase y la tercera. Pues serviría para adaptar dicha salida a los requerimientos particulares de los algoritmos de agrupamiento externos que quisiéramos usar.

Los desarrolladores de este método se dieron cuenta que hay métodos de aprendizaje que trabajan mejor en determinados rangos del tamaño de la entrada y usan esta etapa para adaptar la salida de la primera fase a estos rangos de tamaño. Sin embargo como ya se ha dicho, esta etapa es opcional.

Fase 3: Agrupamiento Global: En esta fase se puede someter los datos de salida de la fase 1, directamente a un algoritmo de clustering como K-means o puede usarse la salida de la Fase 2. Para la aplicación de un algoritmo externo hay varias posibilidades, entre las que se encuentra: aplicar el algoritmo de forma directa y sin modificar sobre los CF o modificar uno de los algoritmos para aplicarlo a este tipo de

entrada.

Fase 4: Esta fase es opcional, pero es usada para refinar el conjunto de salida, eliminando posibles outliers y terminando de afinar los clústers.

Inserción de una entrada en el árbol CF:

Dada una entrada «ent»:

- Identificar la hoja apropiada: Comenzando desde la raíz, descender de forma recursiva el árbol, eligiendo el nodo hijo más cercano de acuerdo a las medidas D_0, D_1, D_2, D_3, D_4 .
- Modificando la Hoja elegida: Cuando se alcanza el nodo hoja, encontrar la entrada más cercana y probar si puede “absorver” la nueva entrada sin violar la Regla T. Si se viola, el vector CF se actualiza, de lo contrario se añade la nueva entrada. Si no hubiera espacio para añadir más entradas en la hoja, entonces hay que dividirlo. Esta división es realizada, usando como semilla el par de entradas más alejadas y redistribuyendo las restantes entradas, basado en el criterio de las más cercanas.
- Modificando el camino hasta la hoja: Luego de insertar la entrada en la hoja, hay que actualizar la información en todos los CF de los nodos de las ramas en todo el camino desde la hoja hasta el nodo raíz. En caso de que no haya división de los CF, solo implica añadir vectores a un CF recalculando N, LS y SS. Sin embargo en caso de que haya que dividir el CF, pues hay que añadirlos para reflejar la nueva hoja.

Hasta aquí hemos visto las ideas generales que están detrás del método de aprendizaje automático conocido como BIRCH, ahora veremos un ejemplo de cómo usarlo sobre el Iris Dataset, como lo hemos estado haciendo con los métodos anteriores.

Recordamos que usamos la implementación de este método en Python y el módulo scikit-learn.

Siempre recordamos, como lo hemos hecho anteriormente, que el Iris Dataset, tiene un total de 150 casos, distribuidos en tres clases y 50 casos en cada una de ellas. Todos los rasgos son numéricos.

En este caso, hemos usado un factor de ramificación igual 8 y un umbral de 0,3; pero no se ha usado ningún método específico para su selección.

```
#Clustering BIRCH
from sklearn.cluster import Birch
import numpy as np
import matplotlib.pyplot as plt
from numpy import unique
from numpy import where
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
# Cargamos el Iris Dataset. Tres clases, 150 casos
# 50 casos por clase, todos los atributos numéricos
idf= datasets.load_iris()
sc = StandardScaler()
X = sc.fit_transform(idf.data)
# Definimos el modelo, le pasamos como parámetro
# B y T.  (B=branching_factor y T= threshold)
model = Birch(branching_factor=8, threshold=0.3)
# Hacemos el clustering y retornamos el clúster
# asignado a cada caso
yhat = model.fit_predict(X)
# Recuperamos los clusters
clusters = unique(yhat)
# Vamos a mostrar un Scatterplot para cada clúster
i=0
for cluster in clusters:
    print("Cluster:{0} cantidad de casos:{1}".format(i,len(yhat[yhat==cluster])))
    row_ix = where(yhat == cluster)
    plt.scatter(X[row_ix, 0], X[row_ix, 1])
    i+=1
# show the plot
plt.show()
```

Fig. 27. Ejemplo de implementación del método BIRCH sobre el Iris Dataset.

En la siguiente imagen, se muestran los resultados de la corrida de este modelo.