



Tarea 1: Técnicas de *Clustering*

Francisca Ramírez

Juan Pablo Muñoz

17 de abril del 2019

Introducción

En esta tarea se exploran distintas técnicas de reconocimiento de patrones basadas en *clustering* vistas en cátedra. Para ello, se cuenta con tres pequeños *datasets* con distintas características, que servirán para contrastar la aptitud que cada técnica posee para cada caso.

Luego de la experimentación, se responden las dos preguntas conceptuales planteadas en el enunciado.

Parte I

Primero, se prepara la ingesta de datos.

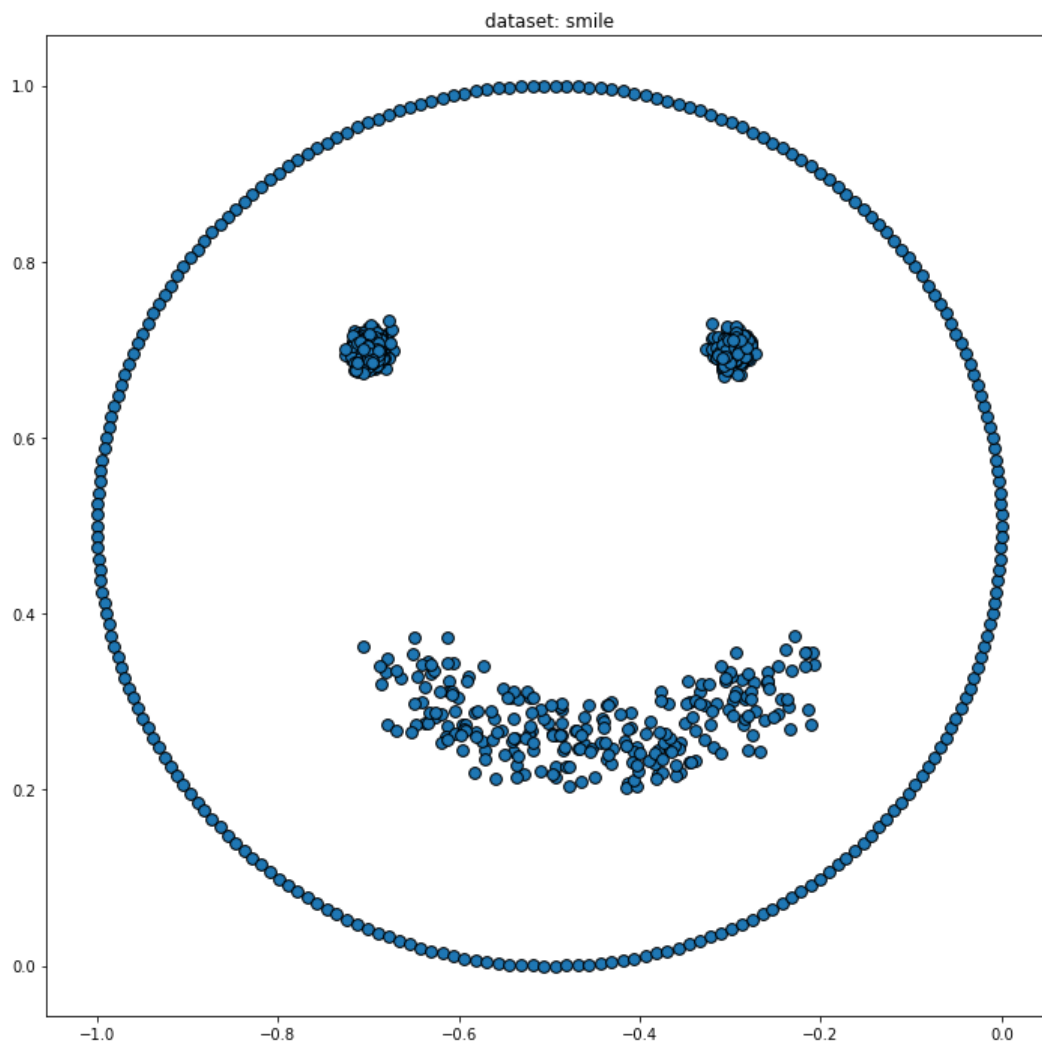
```
In [1]: 1 import os.path
2 import numpy as np
3
4 def ingest_dataset(txt_dir):
5     dataset = list()
6     if os.path.exists(txt_dir):
7         with open(txt_dir, 'r') as f:
8             for line in f.readlines():
9                 data_point = line.split()
10                 x_coord, y_coord = float(data_point[0]), float(data_point[1])
11                 dataset.append([x_coord, y_coord])
12     return np.array(dataset)
```

Y se instancian los tres datasets.

```
In [2]: 1 smile = ingest_dataset('smile.txt')
2 mouse = ingest_dataset('mouse.txt')
3 spiral = ingest_dataset('spiral.txt')
```

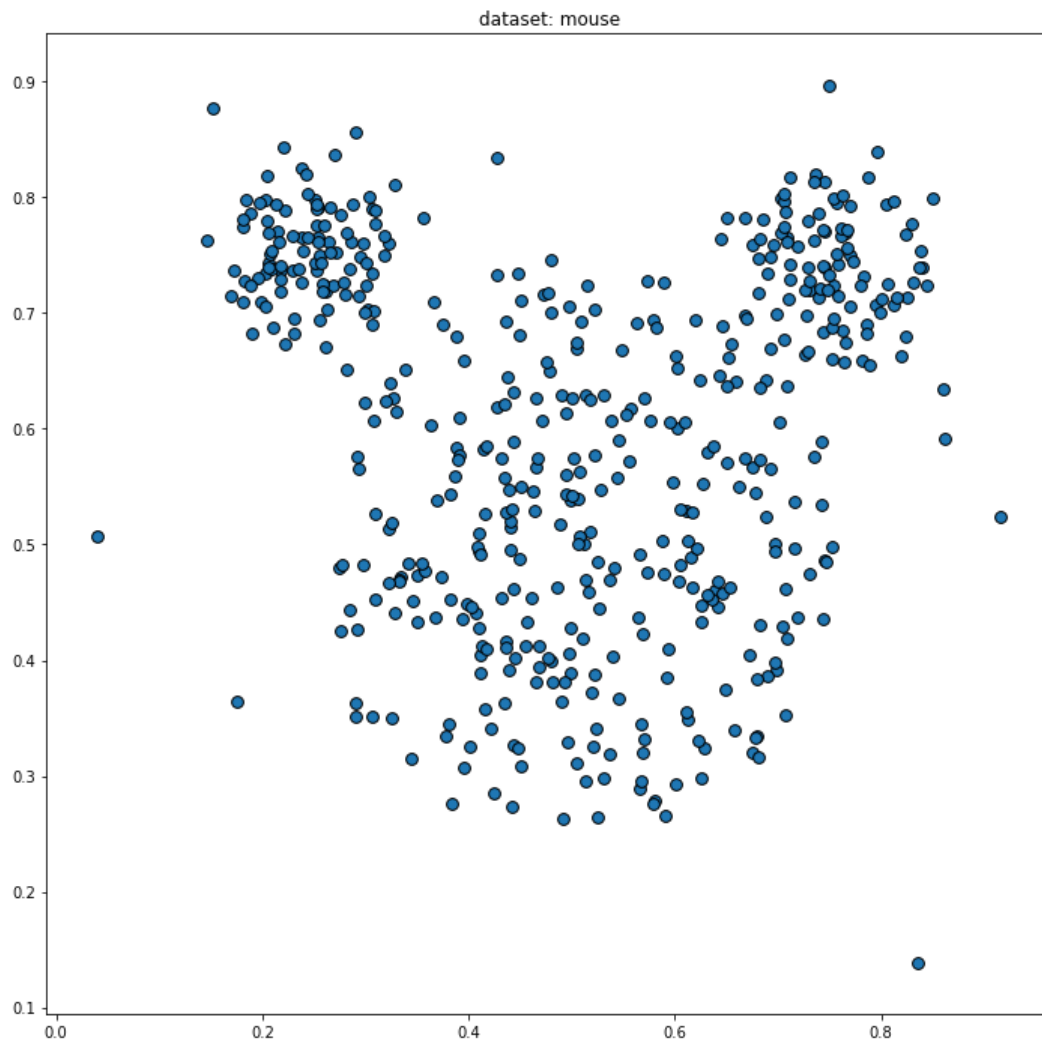
El hecho de que estos datasets sean 2-dimensionales nos permite visualizarlos fácilmente y obtener una clara idea cómo se distribuyen.

```
In [4]: 1 # Smile
2
3 import matplotlib.pyplot as plt
4
5 plt.figure(figsize=(12,12))
6 plt.scatter(smile[:, 0], smile[:, 1], marker='o',
7             edgecolors='k', s=60)
8 plt.title('dataset: smile')
9 plt.show()
```



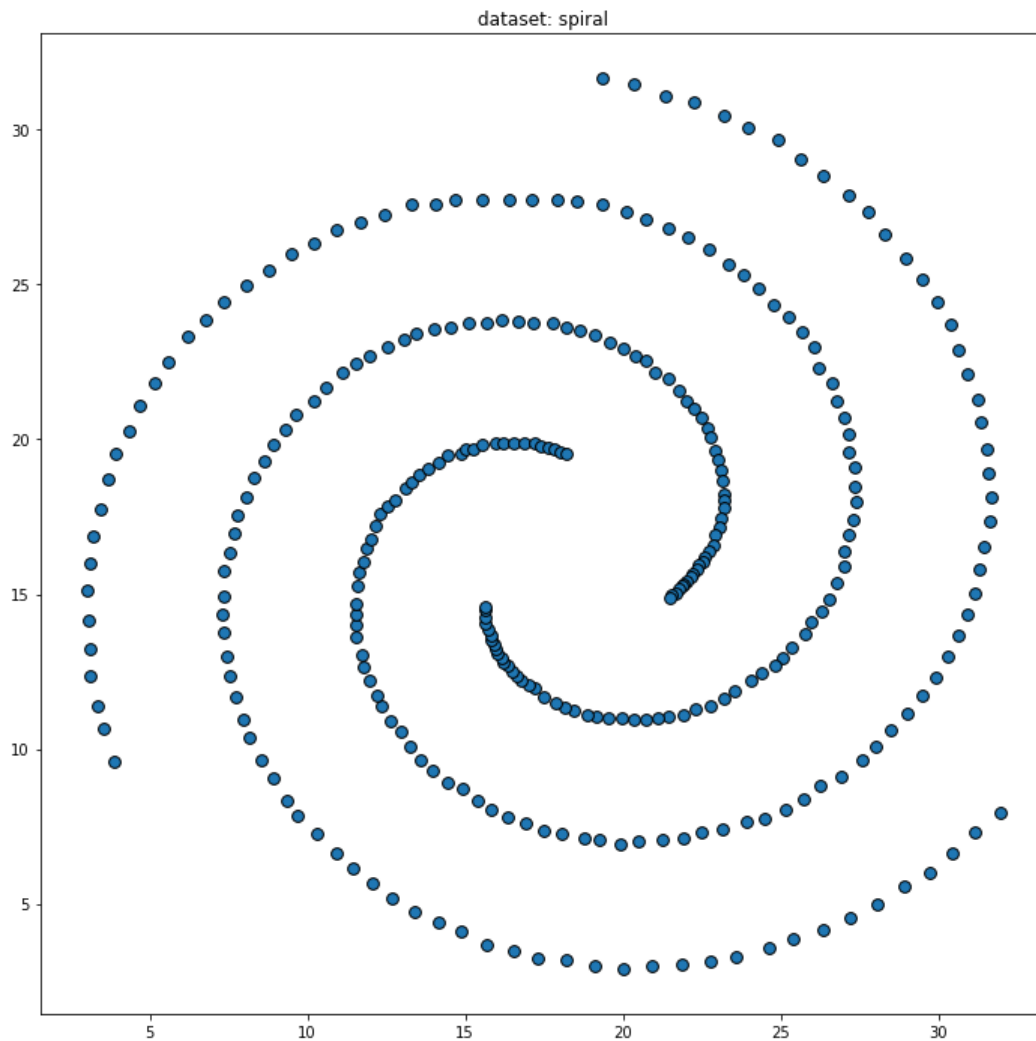
Del gráfico, es evidente que este dataset posee 4 clusters: dos de ellos tienen una alta densidad y tienen forma convexa o globular (los ojos), uno tiene densidad media y es semi-convexo (la sonrisa) y el último es una distribución aparentemente uniforme en forma de circunferencia, que encierra el espacio que los tres clusters anteriores ocupan.

```
In [5]: 1 # Mouse
2 plt.figure(figsize=(12,12))
3 plt.scatter(mouse[:, 0], mouse[:, 1], marker='o',
4             edgecolors='k', s=60)
5 plt.title('dataset: mouse')
6 plt.show()
```



Este dataset posee tres clusters circulares, uno de densidad mediana (cabeza) y dos más pequeños de densidad media-alta (orejas). Los tres clusters tienen forma convexa y tienen poca a nula intersección entre sí.

```
In [6]: 1 # Spiral
2 plt.figure(figsize=(12,12))
3 plt.scatter(spiral[:, 0], spiral[:, 1], marker='o',
4             edgecolors='k', s=60)
5 plt.title('dataset: spiral')
6 plt.show()
```



La distribución de este dataset es no-convexa y resulta intuitivo distinguir cada espiral como un posible cluster distinto. En este caso, no es la densidad de los puntos lo que nos indica la presencia de un cluster, sino el patrón que estos describen.

Los tres datasets tienen un pequeño tamaño y dimensionalidad, por lo que los algoritmos a probar no mostrarán diferencias apreciables en cuanto a tiempos de ejecución, siendo este aspecto uno muy importante a la hora de realizar comparaciones entre distintas técnicas. Por lo tanto, sólo será posible comparar en términos de calidad de resultados.

A continuación, se procede a aplicar las técnicas de *clustering*.

1. K-Means

In [7]:

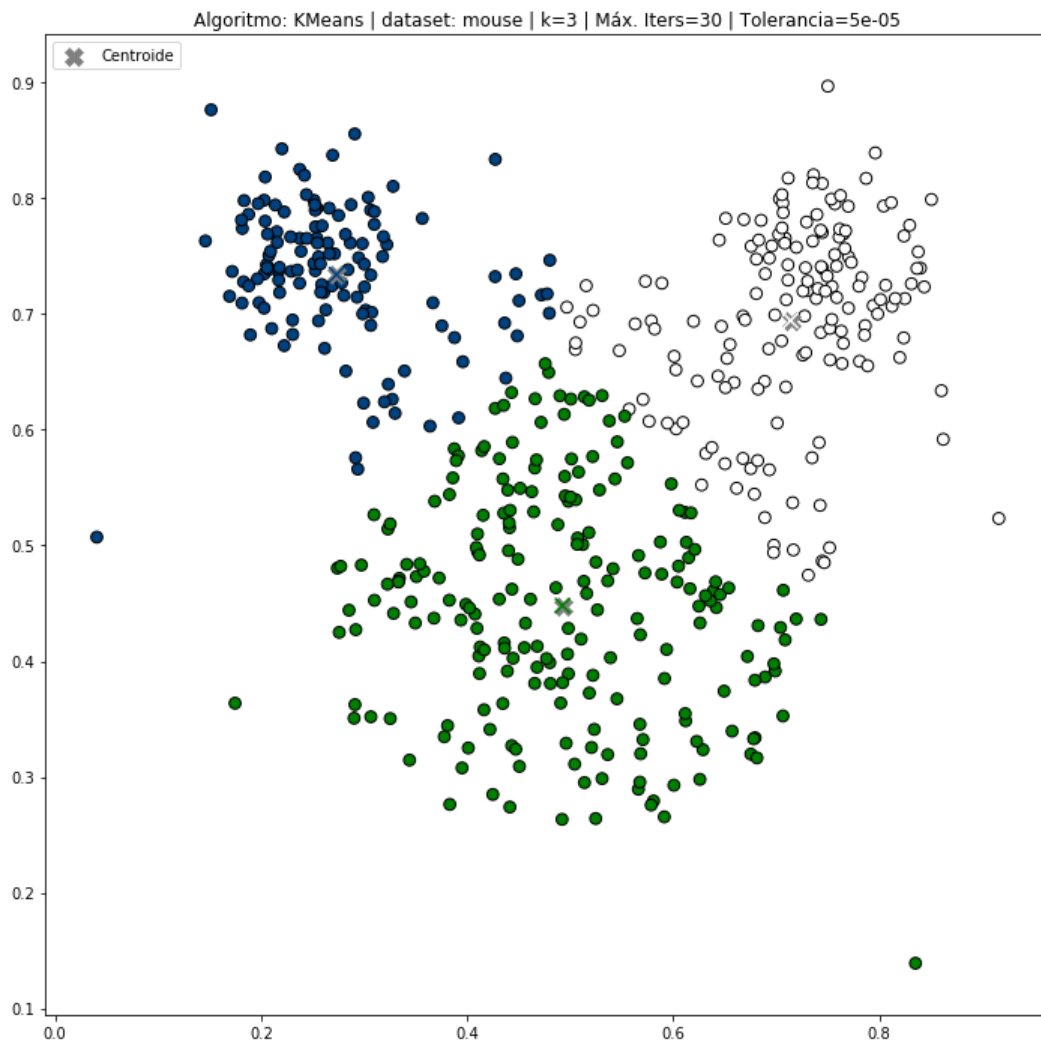
```
1 from sklearn.cluster import KMeans
2 import matplotlib.pyplot as plt
3 from ipywidgets import interact
4 from ipywidgets import FloatSlider
5
6 def apply_kmeans(dataset, k, max_iterations=300, tolerance=1e-4):
7     kmeans = KMeans(
8         n_clusters=k,
9         init='random',
10        n_init=1,
11        max_iter=max_iterations,
12        tol=tolerance,
13        random_state=0,
14    )
15    kmeans.fit(dataset)
16    return kmeans.cluster_centers_, kmeans.labels_
17
18 @interact(
19     dataset_name=['smile', 'mouse', 'spiral'],
20     k=(2,10, 1),
21     max_iterations=(10, 100, 10),
22     tolerance=FloatSlider(
23         min=5e-5,
24         max=5e-4,
25         step=5e-5,
26         continuous_update=False,
27         readout=True,
28         readout_format='.5f'
29     ),
30 )
31 def plot_kmeans(dataset_name, k, max_iterations, tolerance):
32     if dataset_name == 'smile':
33         dataset = smile
34     elif dataset_name == 'mouse':
35         dataset = mouse
36     elif dataset_name == 'spiral':
37         dataset = spiral
38     centroids, labels = apply_kmeans(dataset, k, max_iterations, tolerance)
39     plt.figure(figsize=(12,12))
40     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
41                 edgecolors='k', s=60, cmap=plt.cm.ocean)
42     plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=150,
43                 linewidths=.5, c='gray', cmap=plt.cm.ocean, label='Centroide')
44     plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=100,
45                 linewidths=2, c=list(range(len(centroids))),
46                 cmap=plt.cm.ocean)
47     plt.title('Algoritmo: KMeans | dataset: {} | k={} | Máx. Iters={} | Tolerancia={}'.format(dataset_name, k, max_iterations, tolerance))
48     plt.legend(loc='upper left')
```

dataset_na...

k

max_iterati...

tolerance



Smile

El parámetro k , debe ser 4 para obtener una solución óptima según lo mencionado anteriormente, sin embargo, los cluster formados no son los esperados, esto ocurre porque k-means no puede manejar la diferencia de distribuciones como uno esperaría. Dado que k-means no funciona bien con distribución de datos esférica el cluster exterior y los ojos no son detectados correctamente, incluso, el cluster circular exterior pertenece a cuatro cluster diferentes, eso ocurre porque los centroides se mueven en cada iteración hasta quedar ubicados en posiciones donde cada punto minimice la distancia con su respectivo centroide.

Mouse

El parámetro k debe ser 3 para obtener los cluster esperados según lo mencionado anteriormente. Para este caso k-means logra parcialmente formar los cluster esperados aun que *la cabeza* no pertenece a un solo cluster, pues la densidad de los datos no permite la separación completa de las *orjas* en cluster diferentes al de *la cabeza*. Si el centroide de *las orjas* quedara en el centro de las mismas, habrán datos por ejemplo en la parte posterior de la cabeza que menor distancia al cluster de las orjas que al cluster de la cabeza, y esto obliga a los centroides a moverse.

Spiral

Se espera la detección de tres cluster (uno para cada espiral), por lo tanto, el parámetro k debe ser igual a 3 si queremos formar tres clusters. En este caso, K-means no forma ningún cluster de forma correcta por las mismas razones mencionadas anteriormente. Sin importar la posición de los centroides, la distribución de los datos genera que los puntos del espiral sean cercanos a más de un centroide.

En resumen, K-means tiene poca calidad en estos datasets, porque sus distribuciones no cumplen con los supuestos que este algoritmo asume: clusters globulares y de densidad similar.

2. Agglomerative Hierarchical Clustering

```

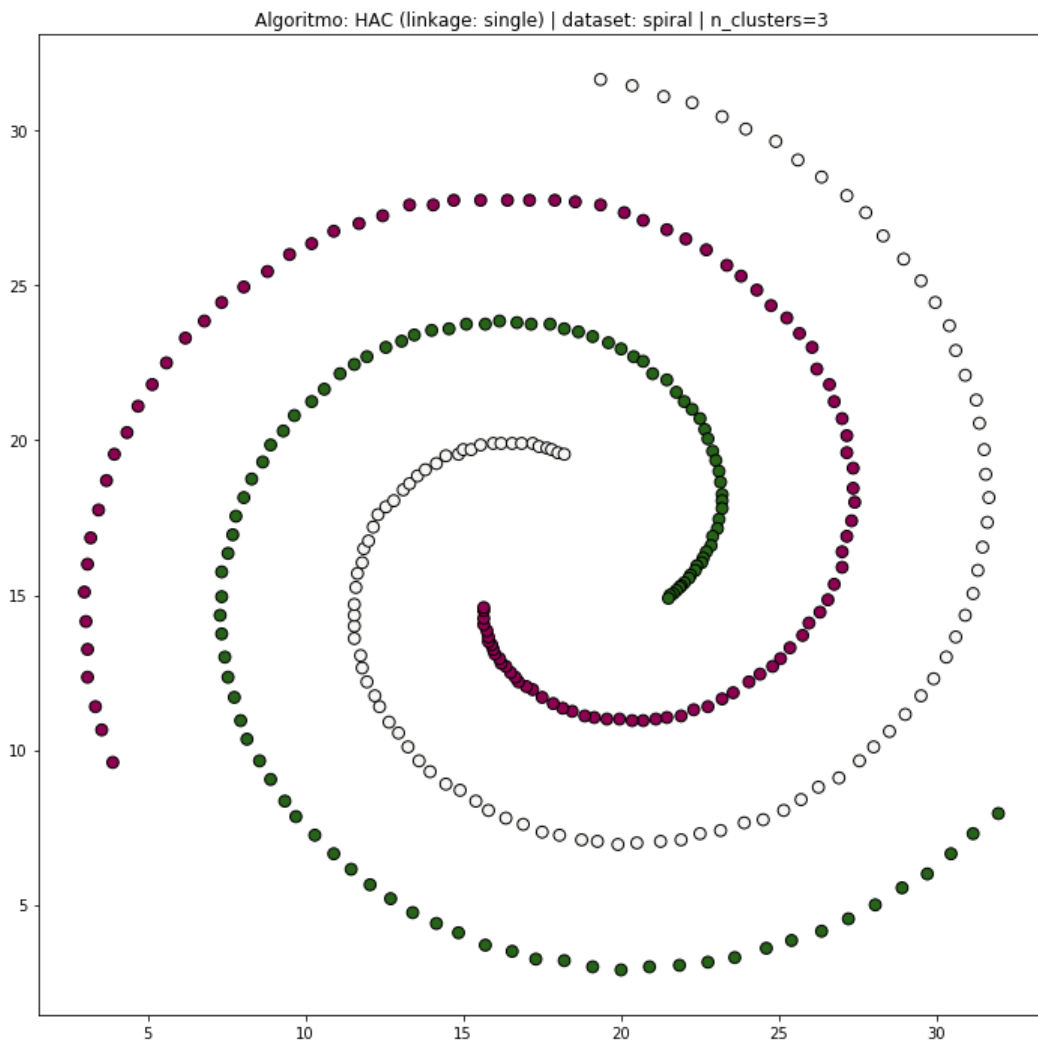
In [8]: 1 from sklearn.cluster import AgglomerativeClustering
2
3 def apply_hac(dataset, linkage, n_clusters):
4     hac = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage)
5     hac.fit(dataset)
6     return hac.labels_
7
8 @interact(
9     dataset_name=['smile', 'mouse', 'spiral'],
10    linkage=['single', 'complete'],
11    n_clusters=(2,10, 1),
12 )
13 def plot_hac(dataset_name, linkage, n_clusters):
14     if dataset_name == 'smile':
15         dataset = smile
16     elif dataset_name == 'mouse':
17         dataset = mouse
18     elif dataset_name == 'spiral':
19         dataset = spiral
20
21     labels = apply_hac(dataset, linkage, n_clusters)
22     plt.figure(figsize=(12,12))
23     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
24                edgecolors='k', s=60, cmap=plt.cm.PiYG)
25     plt.title('Algoritmo: HAC (linkage: {}) | dataset: {} | n_clusters={}'.format(linkage, dataset_name,
26

```

dataset_na...

linkage

n_clusters



Análisis Agglomerative Hierarchical Clustering

Smile

Single-link: Esta versión de HAC hace que cada cluster integre para sí el objeto o cluster exterior que tenga la distancia mínima con alguno de sus objetos miembro. Esto hace que HAC single-link tenga un comportamiento local, por lo que es apto para clústers de distinta densidad y sin estructura definida. Eligiendo el dataset smile, se nota que el algoritmo es capaz de separar los cuatro clusters sin importar sus diferencias en forma y densidad.

Complete-link: En esta versión, los clusters se unen de manera que se "minimiza el diametro" de su unión. Esta propiedad hace que complete-link tenga un comportamiento más global que single-link, haciendo que se prefiera la construcción de clusters más compactos. En el dataset smile, observamos que, mientras los clusters internos (ojos, sonrisa) se encuentran correctamente, es el cluster externo (contorno) el que se ve afectado. Debido a que este cluster es demasiado grande, complete-link prefiere romperlo e integrar partes de éste a los otros clusters.

Mouse

Single-link: En este caso se observa que, definiendo $n_clusters=3$, el algoritmo determina que todo el ratón (dos orejas y cabeza) pertenecen a un solo cluster, mientras que los otros dos clusters encontrados pertenecen a objetos "ruido" o outliers. Esto se debe al comportamiento local de single-link, que lo hace incapaz de identificar objetos outliers como tales.

Complete-link: El resultado mejora pero queda lejos de ser adecuado. Notamos que la presencia de outliers provoca que la medida de similitud o cercanía inter-cluster pueda ser muy afectada, perjudicando la decisión de unión de clusters.

Spiral

Single-link: eligiendo $n_clusters=3$, se obtiene la identificación esperada de clusters. Este resultado tiene la misma explicación que se dio para el caso del dataset smile.

Complete-link: HAC complete-link falla al intentar identificar los espirales correctamente. En este tipo de casos, donde el dataset está distribuido de acuerdo a una alta estructuralidad, complete-link, al tener un comportamiento global, prefiere descartar la estructura local de los datos en favor de generar clusters compactos o de mínimo diámetro.


En resumen, HAC single-link resultó útil para datasets de estructura compleja, como smile y spiral. En mouse, se comportó mal, probablemente por la presencia de ruido. Complete-link tuvo mal resultado en los tres datasets, pues no considera la estructura local de los objetos, y tampoco es robusto ante la presencia de ruido.


3. DBSCAN

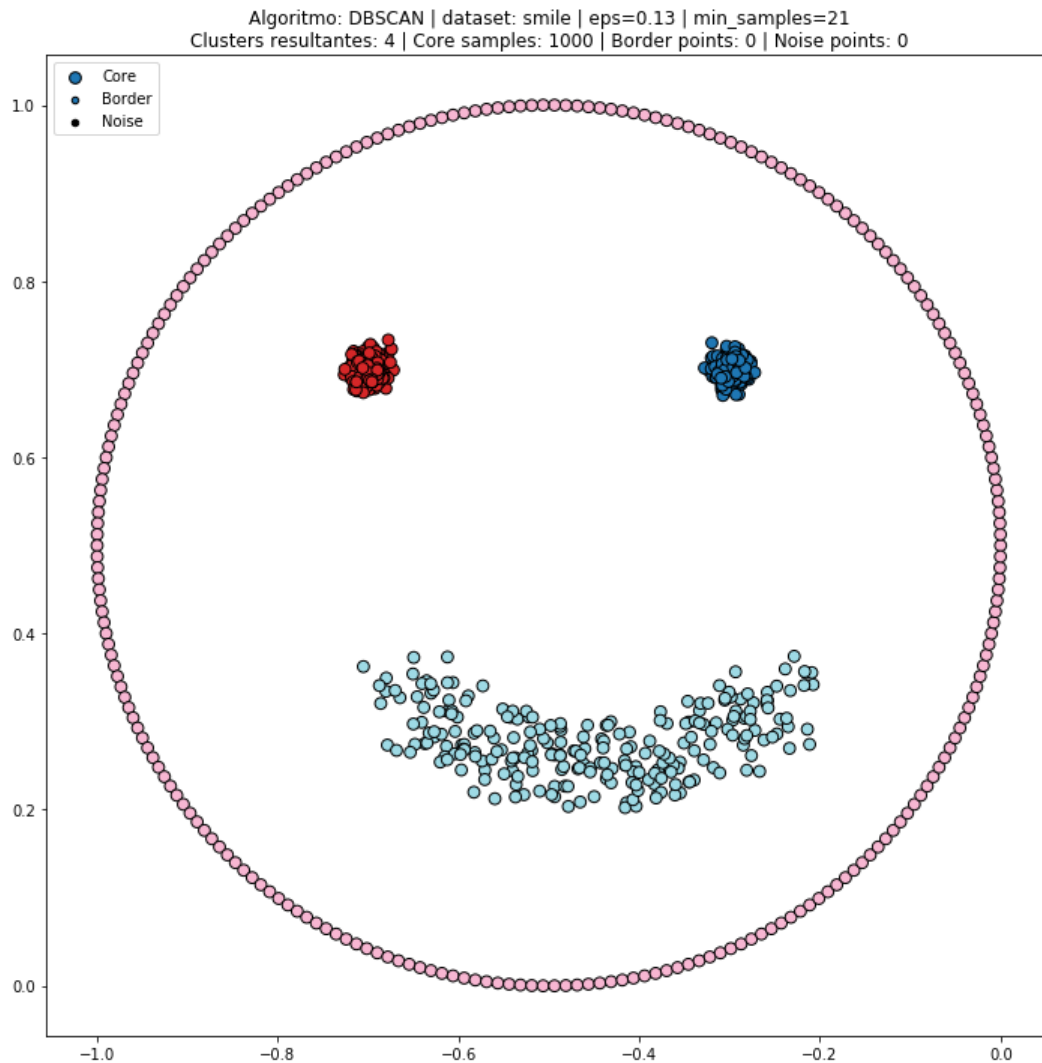
In [9]:

```
1 from sklearn.cluster import DBSCAN
2
3 def apply_dbscan(dataset, min_pts, eps):
4     dbscan = DBSCAN(eps=eps, min_samples=min_pts)
5     dbscan.fit(dataset)
6     core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
7     core_samples_mask[dbscan.core_sample_indices_] = True
8     noise_points_mask = (dbscan.labels_ == -1)
9     border_points_mask = np.zeros_like(dbscan.labels_, dtype=bool)
10    border_points_mask[~core_samples_mask & ~noise_points_mask] = True
11
12    # Number of clusters in Labels, ignoring noise if present.
13    n_clusters_ = len(set(dbscan.labels_)) - (1 if -1 in dbscan.labels_ \
14                                             else 0)
15
16    n_noise_ = list(dbscan.labels_).count(-1)
17    return dbscan.labels_, n_clusters_, n_noise_, core_samples_mask, \
18           border_points_mask, noise_points_mask
19
20 @interact(
21     dataset_name=['smile', 'mouse', 'spiral'],
22     min_pts=(1,50, 1),
23     eps=(0.01, 5.0, 0.01),
24 )
25 def plot_dbscan(dataset_name, min_pts, eps):
26     if dataset_name == 'smile':
27         dataset = smile
28     elif dataset_name == 'mouse':
29         dataset = mouse
30     elif dataset_name == 'spiral':
31         dataset = spiral
32
33     labels, n_clusters, n_noise, core_samples_mask, border_points_mask, \
34     noise_points_mask = apply_dbscan(dataset, min_pts, eps)
35     core_points = dataset[core_samples_mask]
36     border_points = dataset[border_points_mask]
37     noise_points = dataset[noise_points_mask]
38     n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
39     plt.figure(figsize=(12,12))
40     # Plot core samples
41     plt.scatter(core_points[:, 0], core_points[:, 1], marker='o',
42                c=labels[core_samples_mask], edgecolors='k', s=60,
43                cmap=plt.cm.tab20, label='Core')
44
45     # Plot border points
46     plt.scatter(border_points[:, 0], border_points[:, 1], marker='o',
47                c=labels[border_points_mask], edgecolors='k', s=20,
48                cmap=plt.cm.tab20, label='Border')
49
50     # Plot noise points
51     plt.scatter(noise_points[:, 0], noise_points[:, 1], marker='o',
52                c='black', edgecolors='k', s=20,
53                cmap=plt.cm.tab20, label='Noise')
54
55     plt.title('Algoritmo: DBSCAN | dataset: {} | eps={} | min_samples={}\n\
56     Clusters resultantes: {} | Core samples: {} | Border points: {} | Noise points: {}'.
57               .format(dataset_name, eps, min_pts, n_clusters,
58                       len(core_points), len(border_points),
59                       len(noise_points)))
60     plt.legend(loc='upper left')
```

dataset_na... smile

min_pts  21

eps  0.13



Análisis DBSCAN

Smile

Los parámetros para lograr la solución esperada, son los siguientes:

- eps: 0.13. Si se usa un valor mayor, se van a detectar menos cluster por que cada vez más puntos estarán dentro del radio del core. Si se usa un valor menor, los puntos de contorno no logran estar dentro del radio de ninguno de los core.
- min_pts: 21. Si se usan valores mayor a 21, los puntos del contorno no entran en ningún radio de los core. Para valores menores se mantienen los cluster.

El algoritmo DBSCAN funciona muy bien en este set de datos ya que es capaz de detectar formas al separar los cluster según su densidad, sin embargo hay que ser minucioso a la hora de elegir los parámetros para lograr un resultado óptimo.

Mouse

Para este set de datos, el algoritmo será incapaz de detectar los cluster esperados independiente de los parámetros por que la distribución de los datos no tiene una diferencia tan marcada, se podrá diferenciar las *orejas* de la *cabeza*, pero no las orejas entre sí. Sin embargo, este algoritmo podría ser útil para detectar los outliers en el set de datos.

Spiral

Una posible combinación de parámetros para lograr la solución esperada es la siguiente:

- min_pts: 3, si aumenta la cantidad mínima de puntos, aparecerán puntos noise y será necesario aumentar el eps.
- eps: 1.11, si el min_pts se mantiene, este valor podrá aumentar sin cambios en los cluster, en cambio si disminuye, comenzarán a aparecer puntos noise y a se comenzarán a formar más de tres cluster

En términos generales, si aumenta el número de min_pts, hay que aumentar también el eps, esto ocurre porque en el centro de los espirales los datos están mayormente concentrados.

El algoritmo funciona bien para este set de datos y puede detectar la forma de los espirales por la concentración de los datos (densidad), pero el resultado obtenido es muy sensible a los parámetros ingresados.

4. Mean-shift

In [13]:

```
1 from sklearn.cluster import MeanShift
2
3 def apply_meanshift(dataset, bandwidth):
4     meanshift = MeanShift(bandwidth=bandwidth)
5     meanshift.fit(dataset)
6     return meanshift.cluster_centers_, meanshift.labels_
7
8 @interact(
9     dataset_name=['smile', 'mouse', 'spiral'],
10    bandwidth=(0.1, 5, 0.005),
11 )
12 def plot_kmeans(dataset_name, bandwidth):
13     if dataset_name == 'smile':
14         dataset = smile
15     elif dataset_name == 'mouse':
16         dataset = mouse
17     elif dataset_name == 'spiral':
18         dataset = spiral
19     centroids, labels = apply_meanshift(dataset, bandwidth=bandwidth)
20     plt.figure(figsize=(12,12))
21     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
22                 edgecolors='k', s=60, cmap=plt.cm.tab20)
23     plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=150,
24                 linewidths=.5, c='gray', cmap=plt.cm.tab20, label='Centroide')
25     plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=100,
26                 linewidths=2, c=list(range(len(centroids))),
27                 cmap=plt.cm.tab20)
28     plt.title('Algoritmo: Mean-shift | dataset: {} | bandwidth={}\nClusters resultantes: {}'.format(datas
29     plt.legend(loc='upper left')
```

dataset_na...

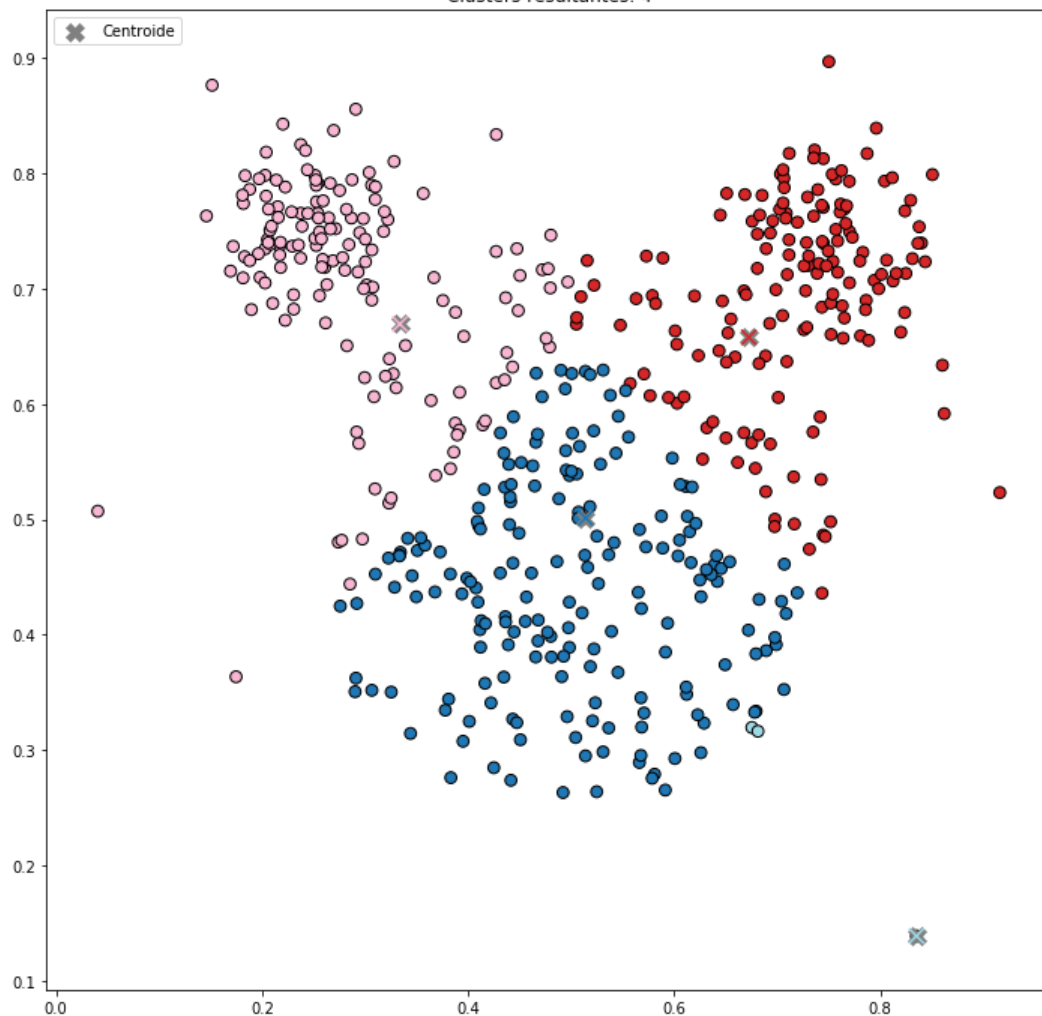
mouse

bandwidth



0.22

Algoritmo: Mean-shift | dataset: mouse | bandwidth=0.22
Clusters resultantes: 4



En general este algoritmo es malo para los tres set de datos, ya que éste funciona bien para densidades uniformes y los dataset no cumplen esta propiedad. El dataset que tiene mayor uniformidad en sus datos es mouse y es el único que logra obtener cluster parecidos a los esperados, aunque no se logra coincidir con el número de ellos.

Smile

No existe un intervalo de valores para el parámetro bandwidth que detecte cuatro clusters. Con bandwidth igual a 0.3 se tienen tres clusters que logran separar la boca y los ojos pero no el contorno. Este es el resultado que más se acerca a lo buscado, pero de todas formas no es bastante alejado de lo esperado. Esto ocurre porque la densidad de los datos no es uniforme y el contorno circular no tiene una zona de mayor concentración a la que se pueda mover el vector.

Mouse

Con este algoritmo no hay un valor para el parámetro bandwidth que logre si quiera un acercamiento a lo esperado, por ejemplo con bandwidth igual a 0.1 se obtiene diez clusters, mientras que para 0.2 se obtienen cuatro clusters. Esto ocurre porque la densidad del set de datos no está bien clara como para lograr que los vectores se concentre sólo en la *cabeza* y en las *orejas*.

Spiral

En este algoritmo con un bandwidth igual a 6 se logra obtener el número de clusters esperados (tres), sin embargo, la forma de los clusters está bien alejada de lo que se busca. Esto ocurre por la naturaleza del algoritmo, que hace que al formar clusters moviendo un centroide a zonas de mayor concentración, tome un comportamiento similar al de k-means.

5. Spectral clustering

In [17]:

```
1 from sklearn.cluster import SpectralClustering
2
3 def apply_spectral_rbf(
4     dataset,
5     n_clusters,
6     random_state=0,
7     n_init=1,
8     gamma=1.0,
9     affinity_matrix_method='rbf',
10 ):
11     spectral = SpectralClustering(
12         n_clusters=n_clusters,
13         random_state=random_state,
14         n_init=n_init,
15         gamma=gamma,
16         affinity=affinity_matrix_method,
17     )
18     spectral.fit(dataset)
19     return spectral.labels_
20
21 def apply_spectral_nearest_neighbors(
22     dataset,
23     n_clusters,
24     n_neighbors,
25     random_state=0,
26     n_init=1,
27     affinity_matrix_method='nearest_neighbors',
28 ):
29     spectral = SpectralClustering(
30         n_clusters=n_clusters,
31         random_state=random_state,
32         n_init=n_init,
33         affinity=affinity_matrix_method,
34         n_neighbors=n_neighbors,
35     )
36     spectral.fit(dataset)
37     return spectral.labels_
38
39 @interact(
40     dataset_name=['smile', 'mouse', 'spiral'],
41     affinity=['RBF', 'K-nearest neighbors'],
42     n_clusters=(2,10, 1),
43     gamma=(0.1, 100.0, 0.1),
44     n_neighbors=(1, 200, 1),
45 )
46 def plot_spectral(
47     dataset_name,
48     affinity,
49     n_clusters,
50     gamma,
51     n_neighbors,
52 ):
53     if dataset_name == 'smile':
54         dataset = smile
55     elif dataset_name == 'mouse':
56         dataset = mouse
57     elif dataset_name == 'spiral':
58         dataset = spiral
59     if affinity == 'RBF':
60         labels = apply_spectral_rbf(
61             dataset=dataset,
62             n_clusters=n_clusters,
63             gamma=gamma,
64             affinity_matrix_method='rbf',
65         )
66     elif affinity == 'K-nearest neighbors':
67         labels = apply_spectral_nearest_neighbors(
68             dataset=dataset,
69             n_clusters=n_clusters,
70             n_neighbors=n_neighbors,
71             affinity_matrix_method='nearest_neighbors',
72         )
73     plt.figure(figsize=(12,12))
74     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
75                 edgecolors='k', s=60, cmap=plt.cm.PiYG)
76     if affinity == 'RBF':
77         plt.title('Algoritmo: Spectral Clustering | Affinity: RBF\
78 (gamma={}) | dataset: {} | n_clusters={}'.format(round(gamma, 2),
79                                                     dataset_name, n_clusters))
80     elif affinity == 'K-nearest neighbors':
81         plt.title('Algoritmo: Spectral Clustering | Affinity: {}-nearest\
```

```

82 neighbors | dataset: {} | n_clusters={}'.format(n_neighbors, dataset_name,
83                                                  n_clusters))
84

```

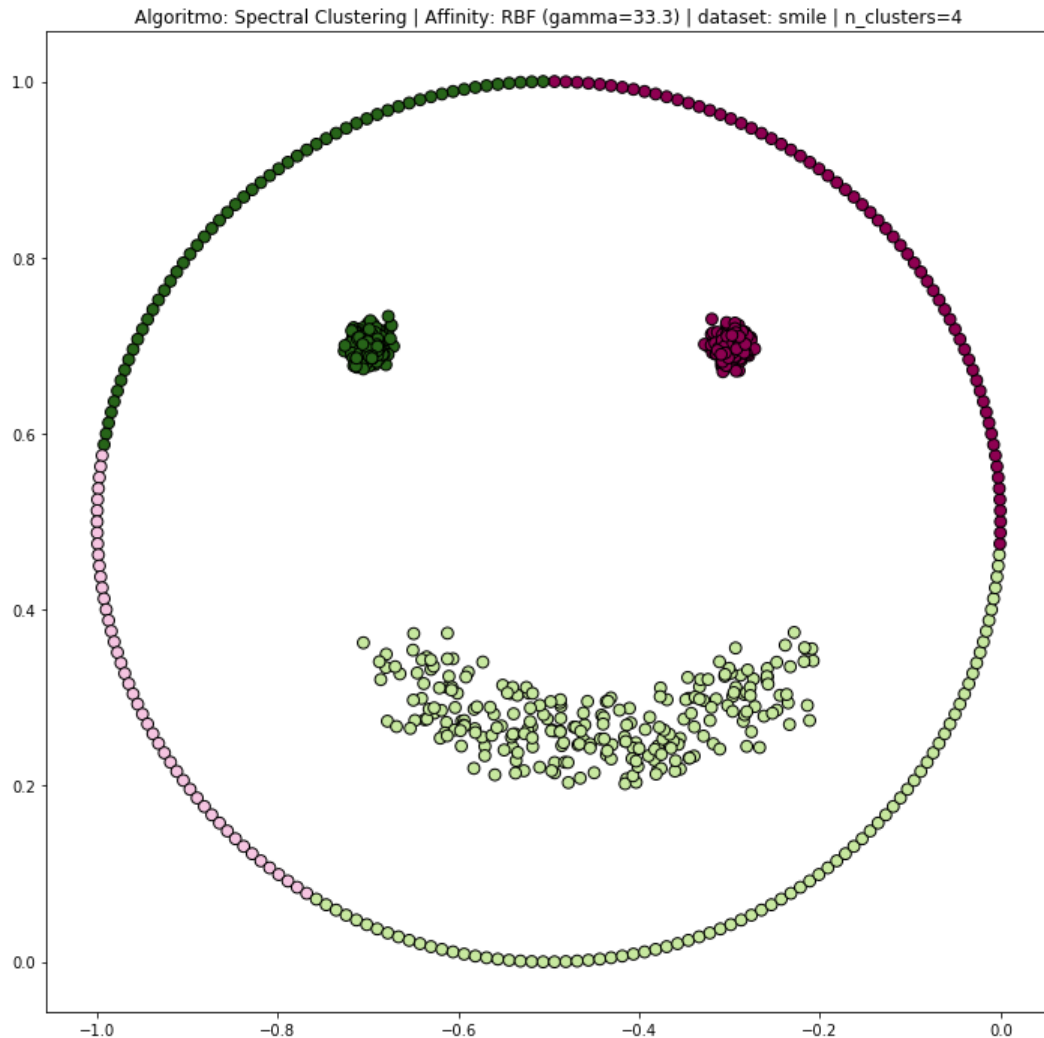
dataset_na... smile

affinity RBF

n_clusters 4

gamma 33.30

n_neighbors 22



Análisis Spectral clustering

Para este algoritmo se definieron dos métodos de construcción de matriz de afinidad: primero, un kernel RBF (Radial Basis Function kernel, o Gaussian kernel), y segundo, los k vecinos más cercanos. Una matriz de afinidad debe estar bien definida, esto es, objetos que se encuentran cerca entre sí deben tener una afinidad mayor a objetos lejanos entre sí.

RBF: Para construir una matriz de afinidad basada en el kernel RBF, se necesita la matriz de distancias entre objetos D . A cada distancia entre pares de objetos d_{ij} , se le aplica la función kernel K :

$$K(d_{ij}) = e^{(-\gamma d_{ij}^2)}$$

Donde γ (gamma) es un parámetro libre que define el "tamaño" del kernel.

Como resultado, cada distancia d_{ij} ahora tendrá un valor en el intervalo $[0, 1]$, donde los valores cercanos a 0 corresponden a los pares de objetos más lejanos entre sí, mientras que los cercanos a 1 son pares de objetos muy cercanos.

K-nearest neighbors: esta matriz de afinidad se define:

$$d_{ij} = \begin{cases} 1 & \text{si } objeto_j \in kNN(objeto_i) \\ 0 & \text{si no.} \end{cases}$$

Smile

RBF: El resultado de aplicar esta matriz de afinidad nos dice que el algoritmo tiene un comportamiento no local respecto a la distribución de los objetos, y que el resultado prácticamente no varía al cambiar el tamaño del kernel gamma. Notamos que los ojos logran identificarse bien, no así la sonrisa, que resulta dividida a la mitad, de forma parecida con el contorno, que es dividido en cuatro partes iguales. Se concluye que el kernel RBF no es adecuado para objetos cuya distribución varía globalmente.

K-nearest neighbors: Con esta matriz de afinidad se llega al resultado esperado. Sin embargo, en este caso particular, observamos una gran sensibilidad respecto a la calidad del resultado y a la cantidad de vecinos k con la que se genera la matriz de afinidad. En nuestro caso, $k = 5$ fue el mínimo valor que produjo la separación de clusters deseada. Si se varía k , se observa que el resultado varía considerablemente, produciendo el resultado esperado en algunos casos, y resultados incorrectos en otros.

Mouse

RBF: Para este dataset, esta matriz de afinidad permite un resultado parecido al esperado. Con valores de gamma pequeños, la división inter-cluster es bastante nítida. Si se aumenta, esta división se hace menos sensible a cambios de densidad en los puntos donde los clusters se intersectan. Debido a que las orejas tienen mayor densidad que la cabeza, observamos que con RBF parte de la cabeza se asigna a los clusters de las orejas, lo que no es deseable.

K-nearest neighbors: Definiendo esta matriz, el resultado es el esperado. Al reflejar mejor la relación local entre objetos, la matriz kNN permite a Spectral clustering definir de mejor manera la separación entre los clusters de las orejas con el de la cabeza. El resultado es de mejor calidad con k entre 10 y 40 aproximadamente. Para k muy grande, el algoritmo comienza a considerar la estructura no local de los datos, por lo que comienza a dar resultados más parecidos a RBF.

Spiral

RBF: En este caso, se obtiene la identificación de clusters deseada si gamma está entre 0.5 y 14.0 aproximadamente, lo que representa una buena sensibilidad del parámetro respecto al resultado obtenido. Creemos que la diferencia de este caso con la mala calidad obtenida en el dataset smile se debe a que en spiral, los datos se distribuyen de manera más uniforme, por lo que no existen regiones con mayor densidad de objetos que puedan afectar la decisión del algoritmo en la formación de clusters.

K-nearest neighbors: Si k es igual a 3 o 4, entonces Spectral clustering llega al resultado deseado, no así con cualquier otro valor de k . La alta sensibilidad se repite, como ocurrió con smile.

En resumen, observamos que Spectral clustering es una técnica bastante flexible y que es capaz de obtener resultados de calidad si se configura correctamente. En particular, usando RBF para definir similitud entre objetos parece tener resultados buenos a nivel local si gamma es pequeño, y a nivel global si se incrementa. El mal resultado obtenido en smile nos lleva a concluir que RBF no rinde bien con clusters de densidad variable.

Con K-nearest neighbors como relación de afinidad entre objetos, se obtiene un resultado apto para identificar clusters a nivel local y con densidad variable. Aumentando la cantidad de vecinos k hace que Spectral clustering tenga un comportamiento menos local. La desventaja de esta métrica de afinidad es que el resultado es en general bastante sensible al parámetro k .

6. Fuzzy C-Means

In [18]:

```
1 from skfuzzy import cluster as fuzzy
2 from ipywidgets import IntSlider
3
4 # Info: https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html
5 def apply_cmeans(dataset, c, m, error, maxiter, seed=0):
6     # El argumento 'data' de cmeans exige que el dataset venga transpuesto!
7     cntr, u, u0, d, jm, p, fpc = fuzzy.cmeans(
8         data=dataset.T,
9         c=c,
10        m=m,
11        error=error,
12        maxiter=maxiter,
13        seed=seed,
14    )
15    return cntr, u, u0, d, jm, p, fpc
16
17 @interact(
18     dataset_name=['smile', 'mouse', 'spiral'],
19     n_clusters=(2, 10, 1),
20     p_exponent=(1.1, 3.0, 0.1),
21     error=FloatSlider(
22         min=5e-4,
23         max=5e-2,
24         step=5e-4,
25         continuous_update=False,
26         readout=True,
27         readout_format='.4f'
28     ),
29     max_iterations=(1, 1000, 1),
30 )
31 def plot_cmeans(
32     dataset_name,
33     n_clusters,
34     p_exponent,
35     error,
36     max_iterations,
37 ):
38     if dataset_name == 'smile':
39         dataset = smile
40     elif dataset_name == 'mouse':
41         dataset = mouse
42     elif dataset_name == 'spiral':
43         dataset = spiral
44     cntr, u, u0, d, jm, p, fpc = apply_cmeans(
45         dataset=dataset,
46         c=n_clusters,
47         m=p_exponent,
48         error=error,
49         maxiter=max_iterations,
50     )
51     # El color de cada punto es asignado segun el cluster al cual pertenezca
52     # con mayor porcentaje
53     labels = u.argmax(axis=0)
54     plt.figure(figsize=(12,12))
55     plt.scatter(dataset[:, 0], dataset[:, 1], marker='o', c=labels,
56                 edgecolors='k', s=60, cmap=plt.cm.ocean)
57     plt.scatter(cntr[:, 0], cntr[:, 1], marker='X', s=150,
58                 linewidths=.5, c='gray', cmap=plt.cm.ocean, label='Centroide')
59     plt.scatter(cntr[:, 0], cntr[:, 1], marker='x', s=100,
60                 linewidths=2, c=list(range(n_clusters)),
61                 cmap=plt.cm.ocean)
62     plt.title('Algoritmo: Fuzzy C-Means | dataset: {} | n_clusters={} | \
63 p={} | Máx. Iters={} | error={} \n Fuzzy partition coefficient (FPC): {}'.format(
64         dataset_name,
65         n_clusters,
66         round(p_exponent, 2),
67         max_iterations,
68         round(error, 5),
69         round(fpc, 3),
70     ))
71     plt.legend(loc='upper left')
```

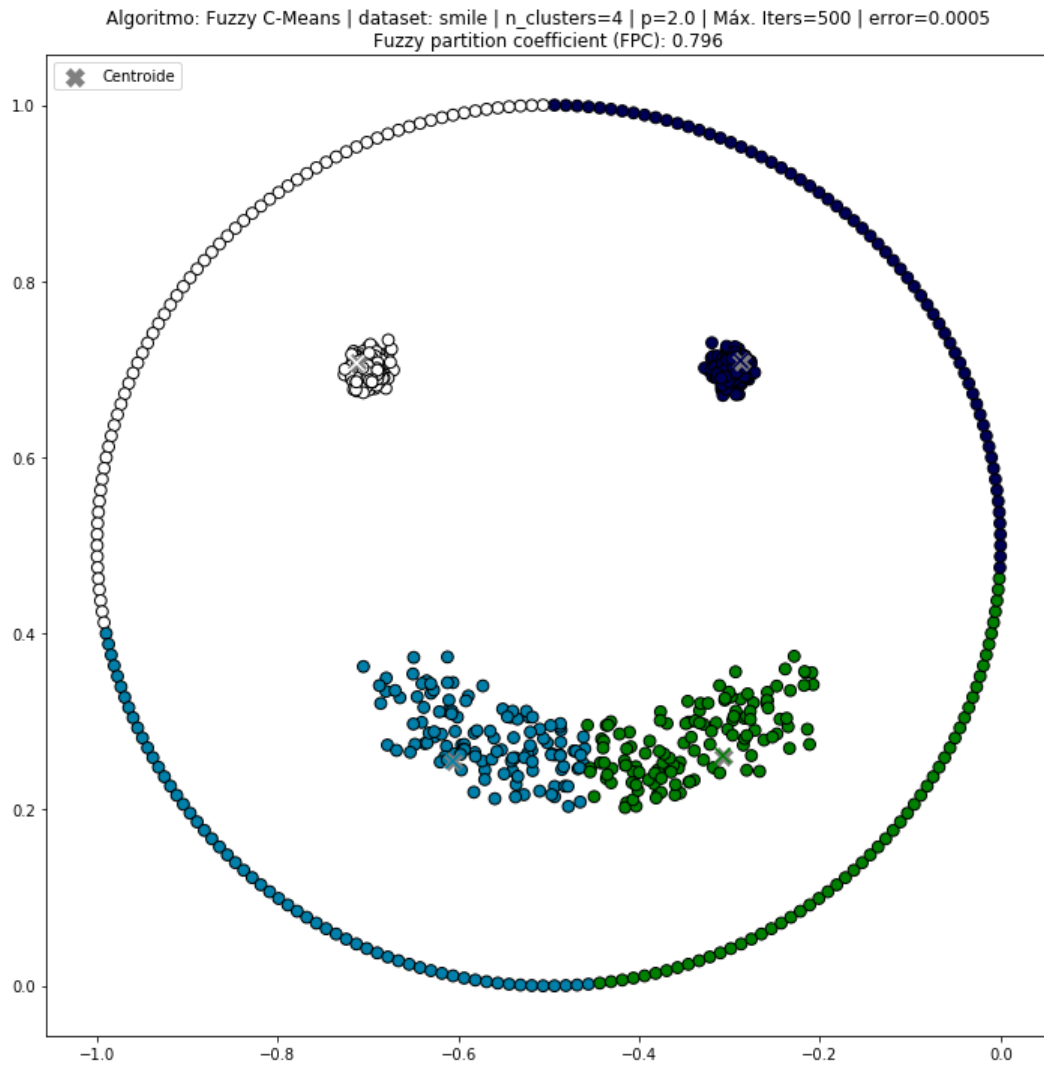
dataset_na...

n_clusters 4

p_exponent 2.00

error 0.0005

max_iterati... 500



Análisis Fuzzy C-Means

En general se puede observar que este algoritmo tiene el mismo comportamiento que k-means y el único parametro que afecta el resultado es `n_cluster` (numero de cluster), esto ocurre porque la gráfica no muestra el nivel de pertenencia de cada punto, que es la idea principal del algoritmo. Entonces, considerando lo anterior, los parámetros ideales para cada dataset seran como los de k-means:

Smile:

- `n_clusters`: 4

Mouse:

- `n_cluster`: 3

Spiral:

- `n_cluster`: 3

En los tres dataset, al cambiar el valor de `p_exponent` o del error se verán leves cambios en la posición del centroide lo que afectara en los puntos de borde de cada cluster, pero la forma general de los cluster se mantendrá intacta.

Conclusiones.

Finalmente considerando la eficiencia y resultados de los cluster se puede concluir que:

- Para el dataset smile el algoritmo que da mejores resultados es DBSCAN.
- Para el dataset mouse el algoritmo que da mejores resultados podría ser k-means o fuzzy c-means.
- Para el dataset spiral el algoritmo que da mejores resultados es DBSCAN.

Parte II

(a) Se tiene un conjunto de datos con 100 objetos. Se le pide realizar clustering utilizando K-means, pero para todos los valores de k , $1 \leq k \leq 100$, el algoritmo retorna que todos los clusters estan vacíos, excepto uno. ¿En que situación podría ocurrir esto? (analice los datos y no los parametros del algoritmo, i.e., iteraciones). ¿Qué resultado tendría single-link y DBSCAN para este tipo de datos?

Resp.: La situación anteriormente descrita puede ocurrir si los 100 datos son iguales, es decir, se ubican en el mismo punto, de esta forma, los datos serán más cercanos a un solo centroide y pertenecerán a un cluster mientras que los demás cluster no tendran datos.

Si los datos estuviesen un poco mas distribuidos en el espacio, la unica forma de que ocurra esta situación es que de los centroides iniciales solo uno este cerca de los datos y el resto esto lo suficientemente alejado para no alcanzar los datos.

El resultado de DBSCAN será un solo cluster con todos los puntos dentro independiente del eps.

(b) Considerando single-link y complete-link hierarchical clustering, ¿es posible que un objeto esté más cerca (en distancia Euclidiana) de los objetos de otros clusters en relación a los de su propio cluster? Si fuese posible, ¿en que enfoque (single y/o complete) esto podría ocurrir? Justifique con un ejemplo en cada caso.

Resp.: En cada iteración de single-link, un punto se incluye en cualquiera sea el cluster al cual pertenezca su punto más cercano. Al construirse los clusters de esta manera, no existe forma de que algún par de puntos de distintos clusters sean más cercanos entre sí que con los puntos de sus respectivos clusters.

En cambio, en complete-link esto sí puede ocurrir. Considerar, por ejemplo, la siguiente distribución de los objetos A, B, C, D, E en una dimensión:

A-B-C--D---E

Donde la distancia es directamente proporcional a la cantidad de caracteres entre los objetos. Si queremos un resultado de 2 clusters, se obtiene:

[A-B-C]--[D---E]

De aquí notamos que D dista de C (que está en un cluster distinto) sólo en 2 unidades, mientras que la distancia con E (que está en el mismo cluster) es de 3. Así, con este ejemplo, queda demostrada la posibilidad de que un objeto de un cluster puede tener como objetos más cercanos a aquellos de otros clusters.

In []:

1