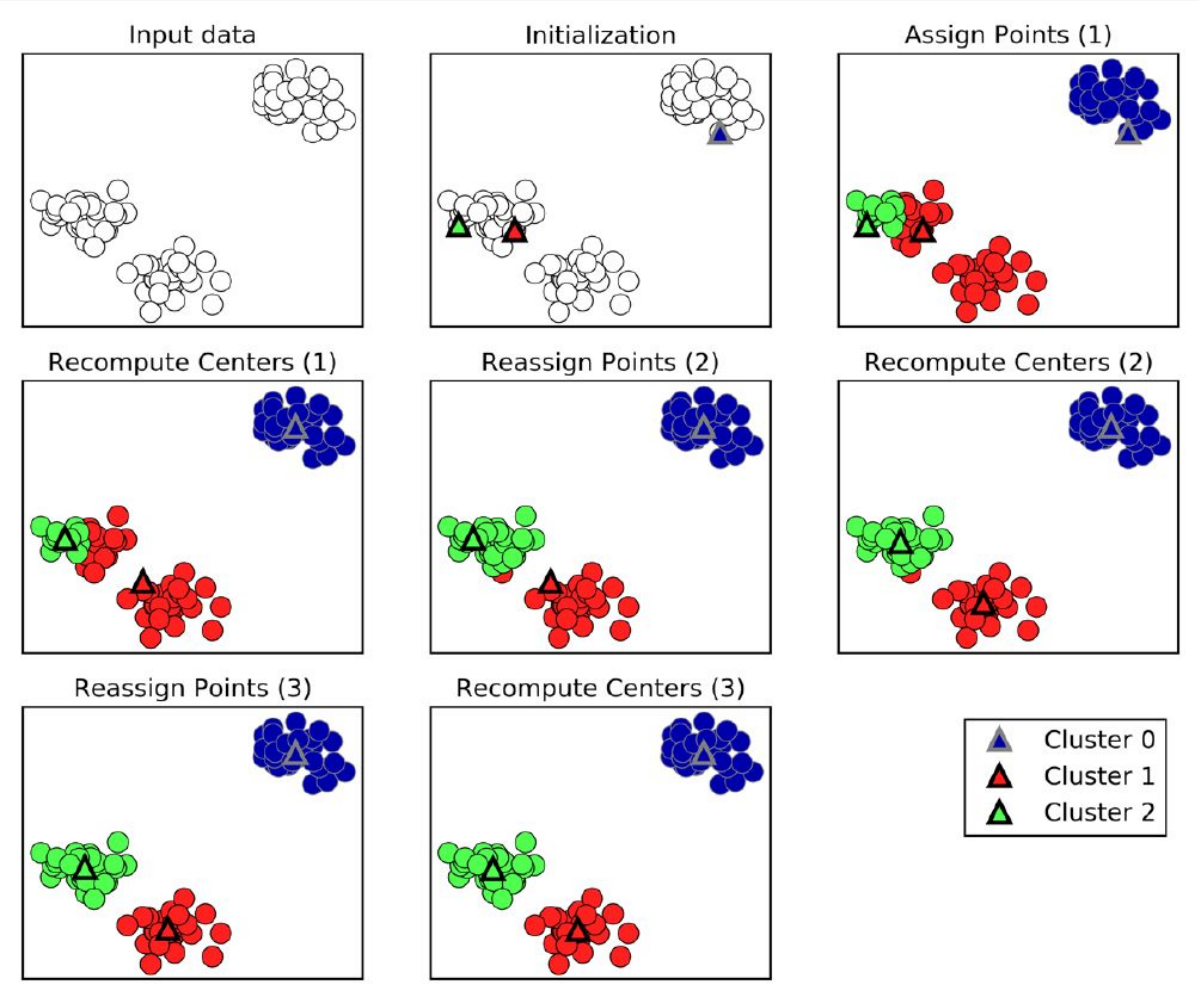

CLUSTERING
K-MEANS
AGLOMERATIVE
DBSCAN

K-Means

- Aprendizaje no supervisado
- Encontrará k clusters (conjuntos)
- Sirve para hacer predicciones
- Los clusters tienen formas “simples”
- Es rápido de ejecutar
- Depende del azar

¿cómo funciona?



¿Cómo funciona?

Para cada cluster se escoge un punto arbitrario del espacio (según la dimensión de los datos) al azar y se le asigna un color (lable) y se le llama centro

Se calcula la distancia de cada punto de nuestro dataset a todos los centros

Cada punto del dataset se pinta del color del centro mas cercano

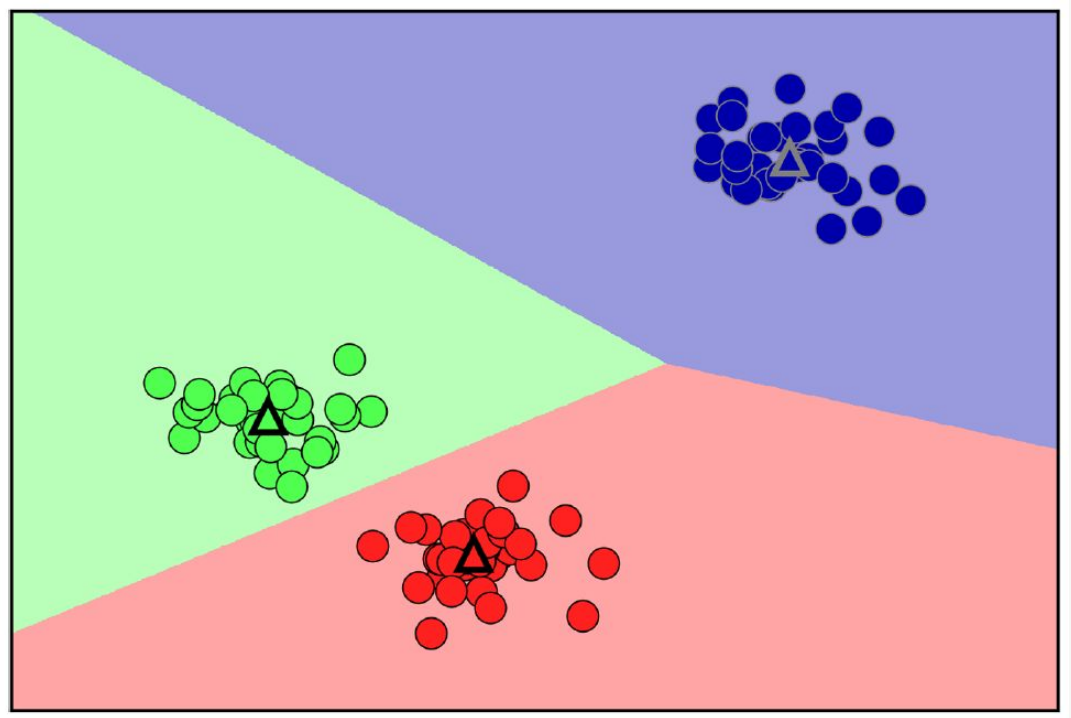
Se saca el “promedio” de todos los puntos pintados del mismo color y se calcula un nuevo centro (el centro no tiene que pertenecer al dataset)

Se despintan todos nuestros datos y se vuelve a empezar el algoritmo con los nuevos centros.

Finaliza cuando ya no hay cambios en los colores de los puntos del dataset.

Y así queda.

```python
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# generate synthetic two-dimensional data
X, y = make_blobs(random_state=1)

# build the clustering model
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

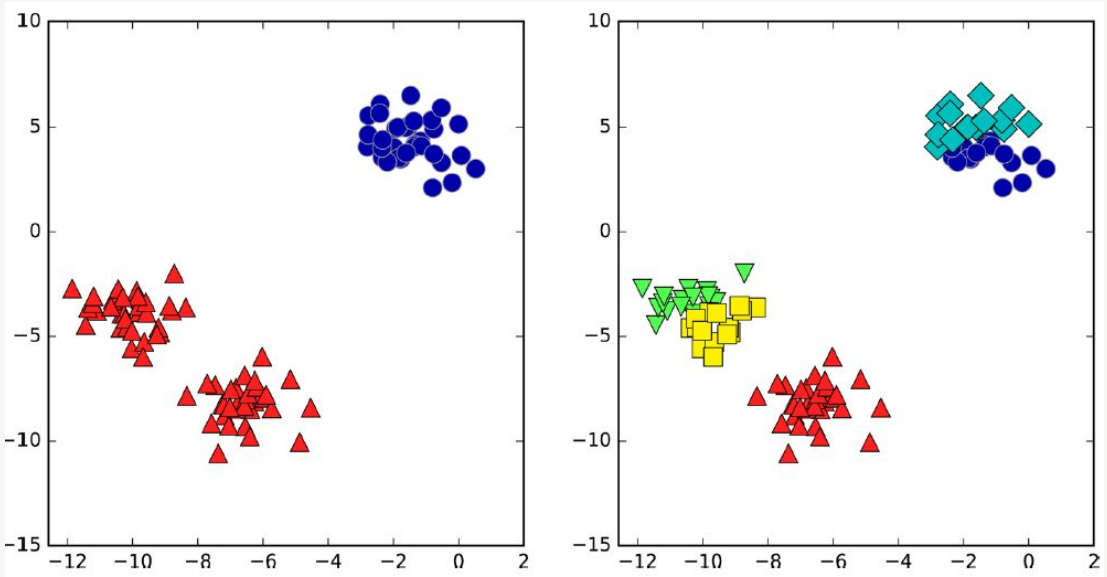


```python
print(kmeans.predict(X))
```

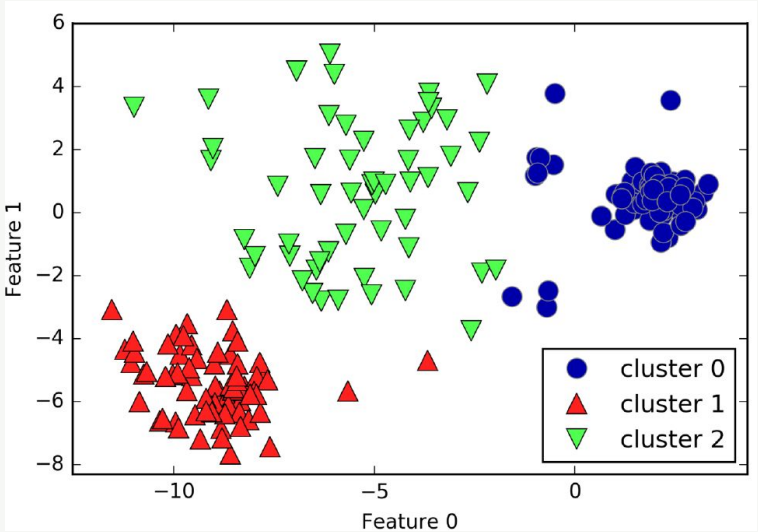
```
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 0 1 2 0 0 1 1 0 1 1 0 1 2 0 2
 2 2 0 0 2 1 2 2 0 1 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 2 0 1
 1 2 0 0 1 2 1 2 2 0 1 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1]
```

Y si pedimos mas clusters

```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# using two cluster centers:
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
assignments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[0])

# using five cluster centers:
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
assignments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[1])
```

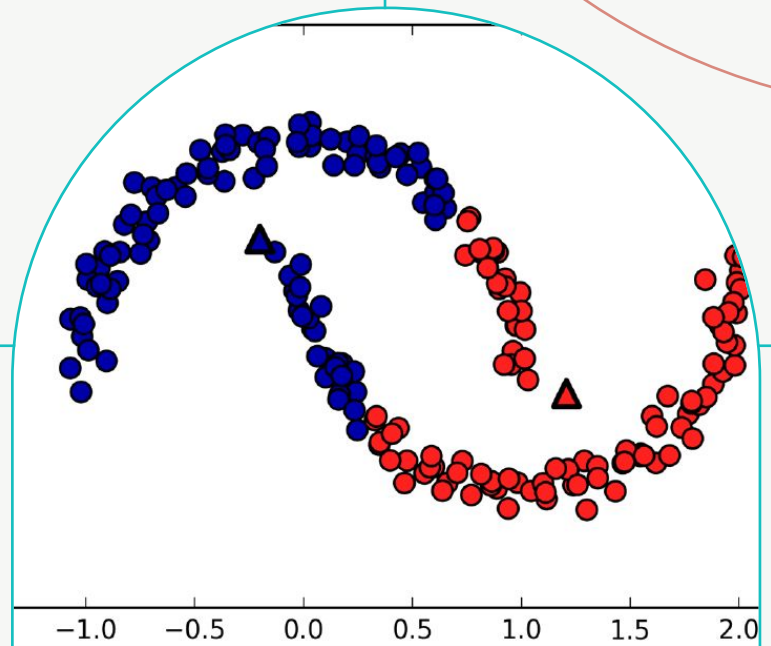


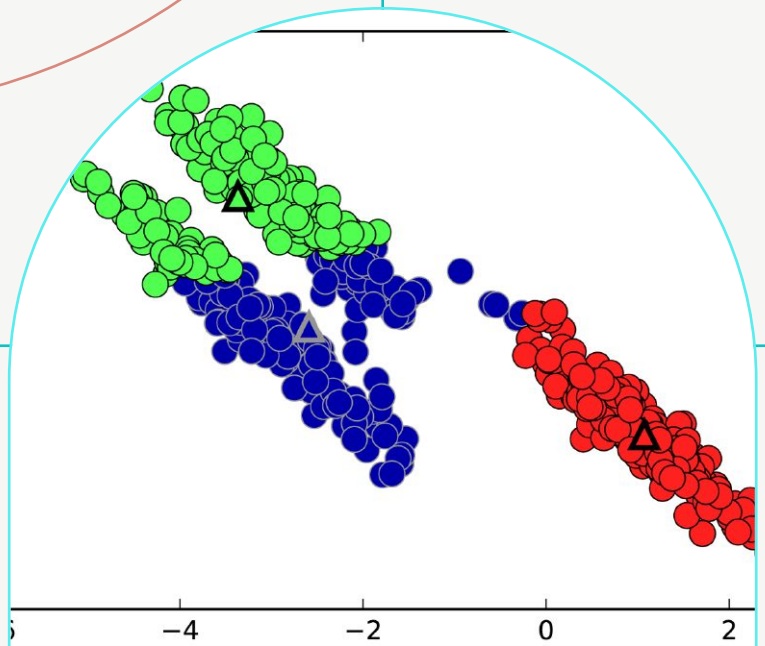
Más problemas




```
X_varied, y_varied = make_blobs(n_samples=200,
                                cluster_std=[1.0, 2.5, 0.5],
                                random_state=170)
y_pred = KMeans(n_clusters=3, random_state=0).fit_predict(X_varied)

mglearn.discrete_scatter(X_varied[:, 0], X_varied[:, 1], y_pred)
plt.legend(["cluster 0", "cluster 1", "cluster 2"], loc='best')
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

No puede tomar formas complejas



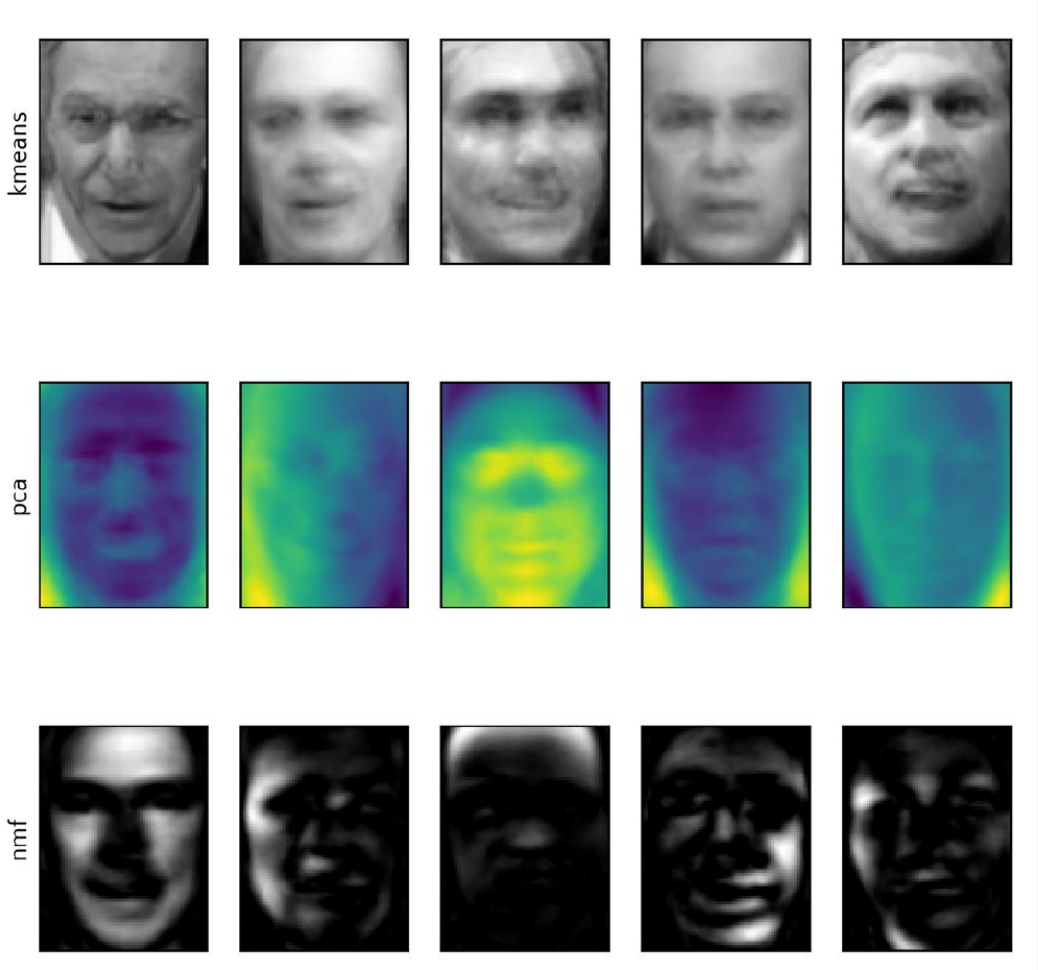


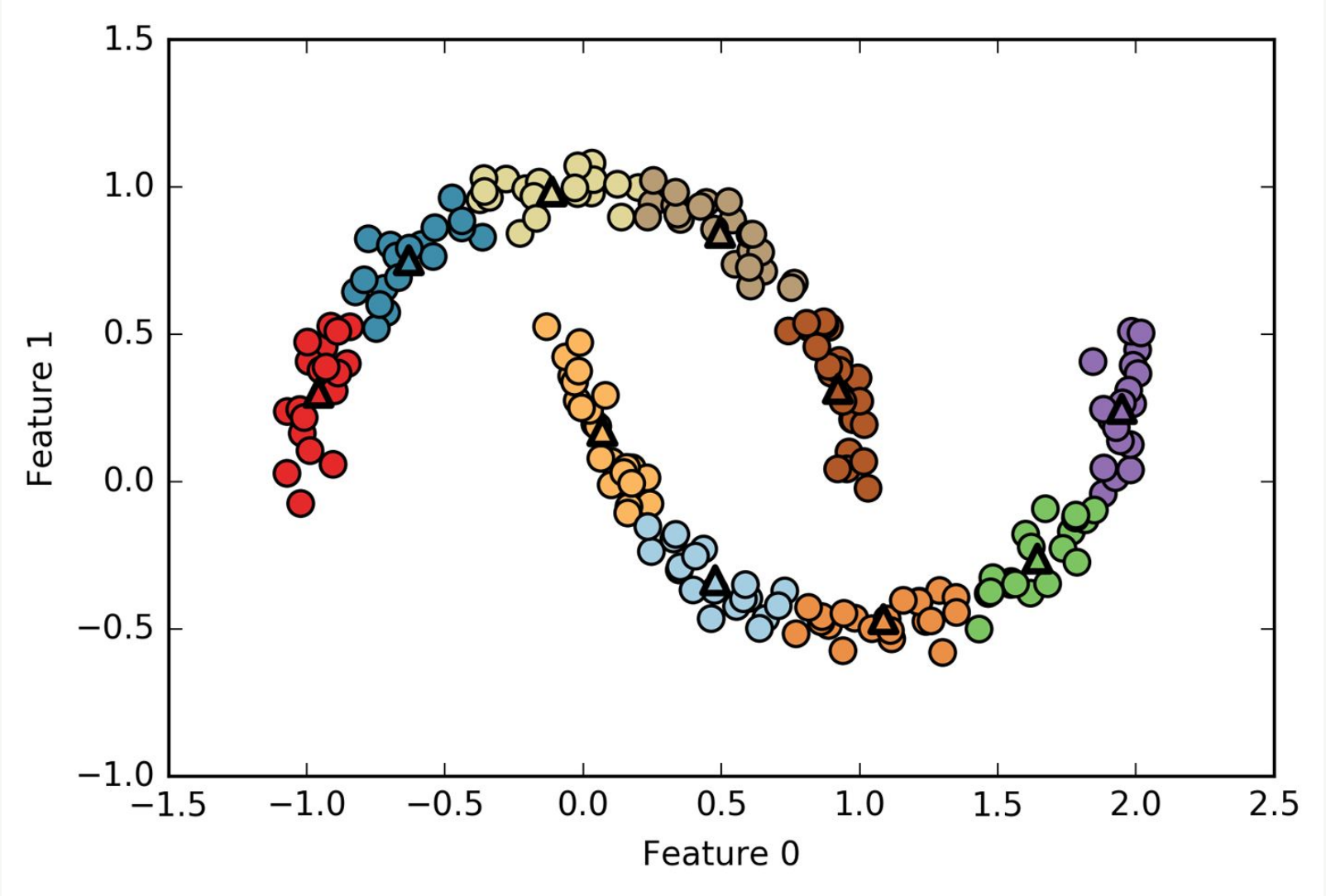
Vector de cuantizaación de k-means

Después de aplicar k-means podemos fijarnos en el centro de cada cluster, ese es un punto en el espacio y Podemos juntar a todos en un vector

Funciona como descomposición

```
X_train, X_test, y_train, y_test = train_test_split(
    X_people, y_people, stratify=y_people, random_state=0)
nmf = NMF(n_components=100, random_state=0)
nmf.fit(X_train)
pca = PCA(n_components=100, random_state=0)
pca.fit(X_train)
kmeans = KMeans(n_clusters=100, random_state=0)
kmeans.fit(X_train)

X_reconstructed_pca = pca.inverse_transform(pca.transform(X_test))
X_reconstructed_kmeans = kmeans.cluster_centers_[kmeans.predict(X_test)]
X_reconstructed_nmf = np.dot(nmf.transform(X_test), nmf.components_)
```




1.5
1.0
0.5
0.0
−0.5
−1.0
Feature 1
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Feature 0

Agglomerative Clustering

- No es un método predictivo, se enfoca en conocer el comportamiento de los datos.

- Se refiere a una colección de algoritmos de agrupamiento que se construyen sobre el mismo principio

El principio

Se considera a cada punto como su propio clúster, y luego fusiona los dos clusters “más similares” hasta que se cumpla algún criterio de detención

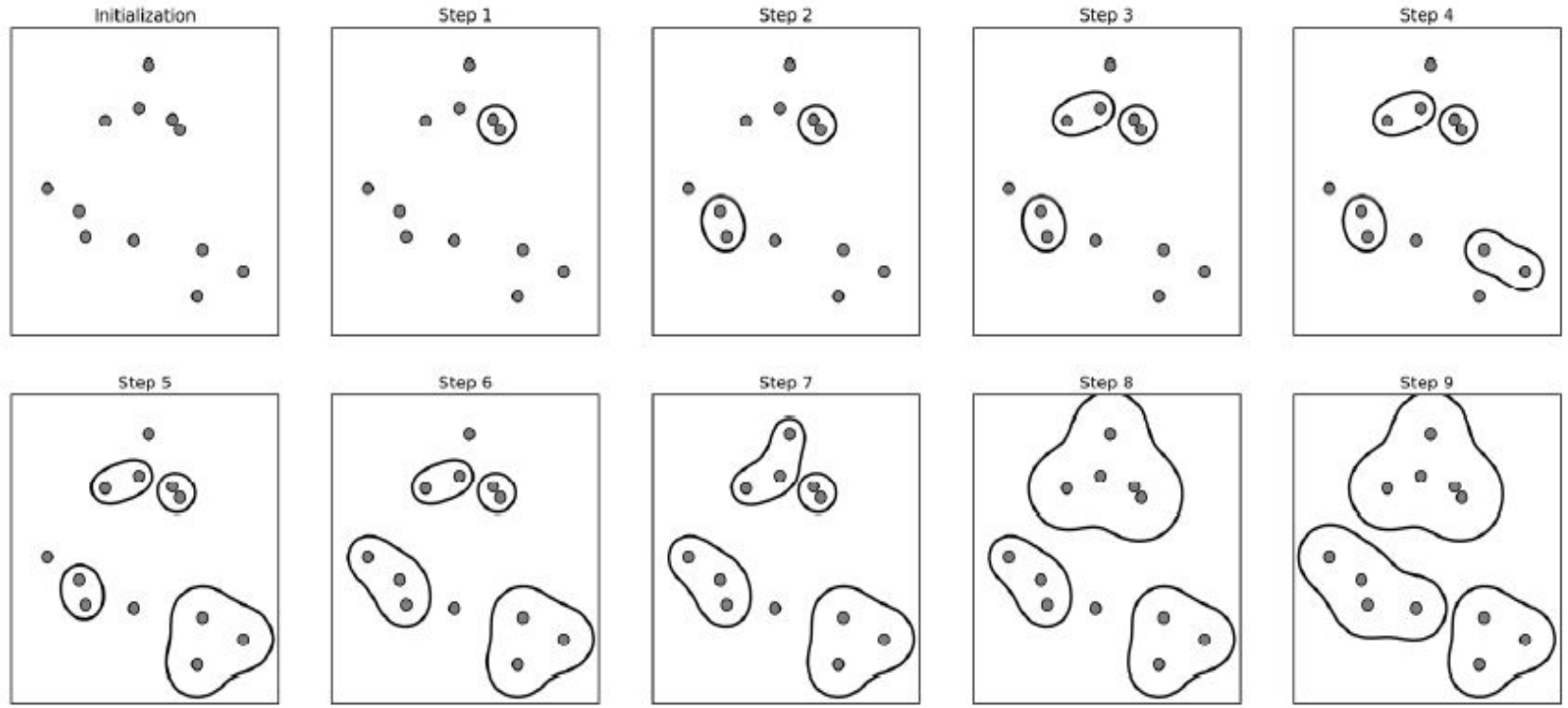


Figure 3-33. Agglomerative clustering iteratively joins the two closest clusters

Criterios de Fusion (linkage)

Ward

Menor incremento de varianza

Average

Menor promedio de las distancias entre todo los puntos

Complete

Menor máxima distancia entre los puntos de distintos clusters

Implementación del algoritmo

```
In[62]:

    from sklearn.cluster import AgglomerativeClustering
    X, y = make_blobs(random_state=1)

    agg = AgglomerativeClustering(n_clusters=3)
    assignment = agg.fit_predict(X)

    mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
    plt.xlabel("Feature 0")
    plt.ylabel("Feature 1")
```

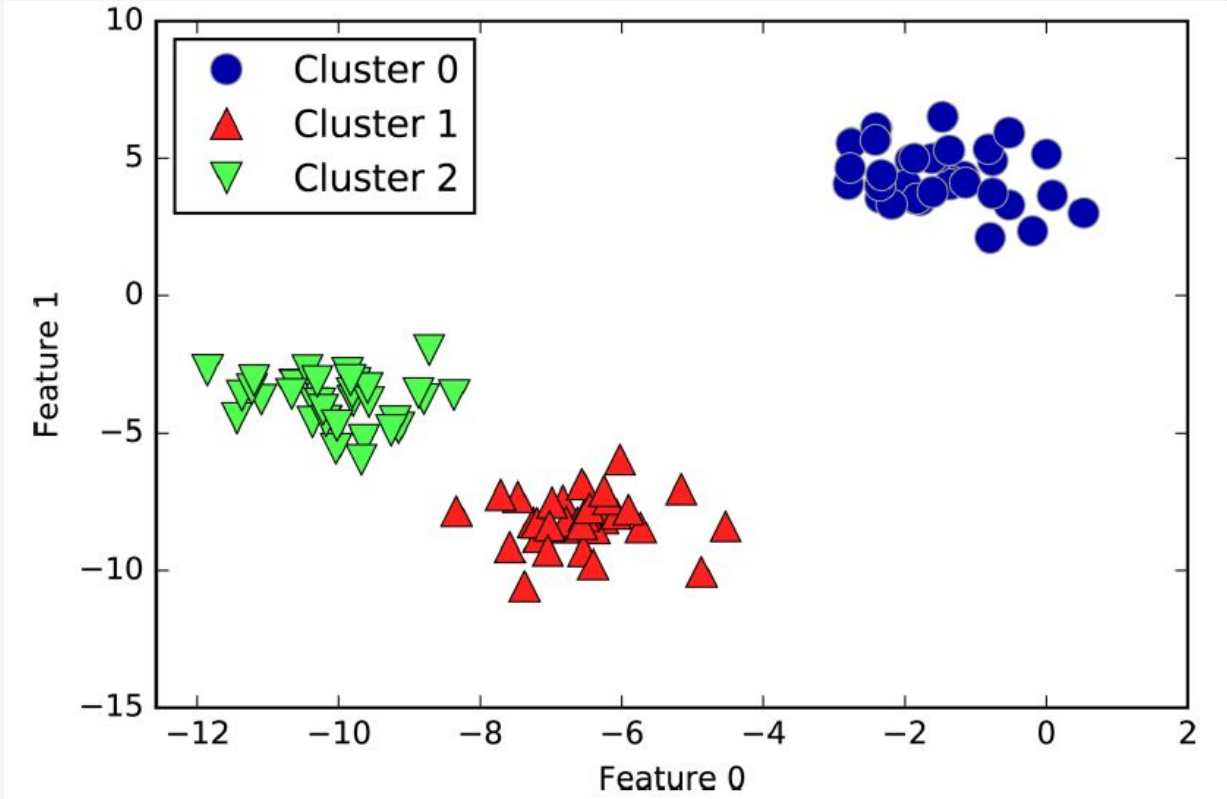
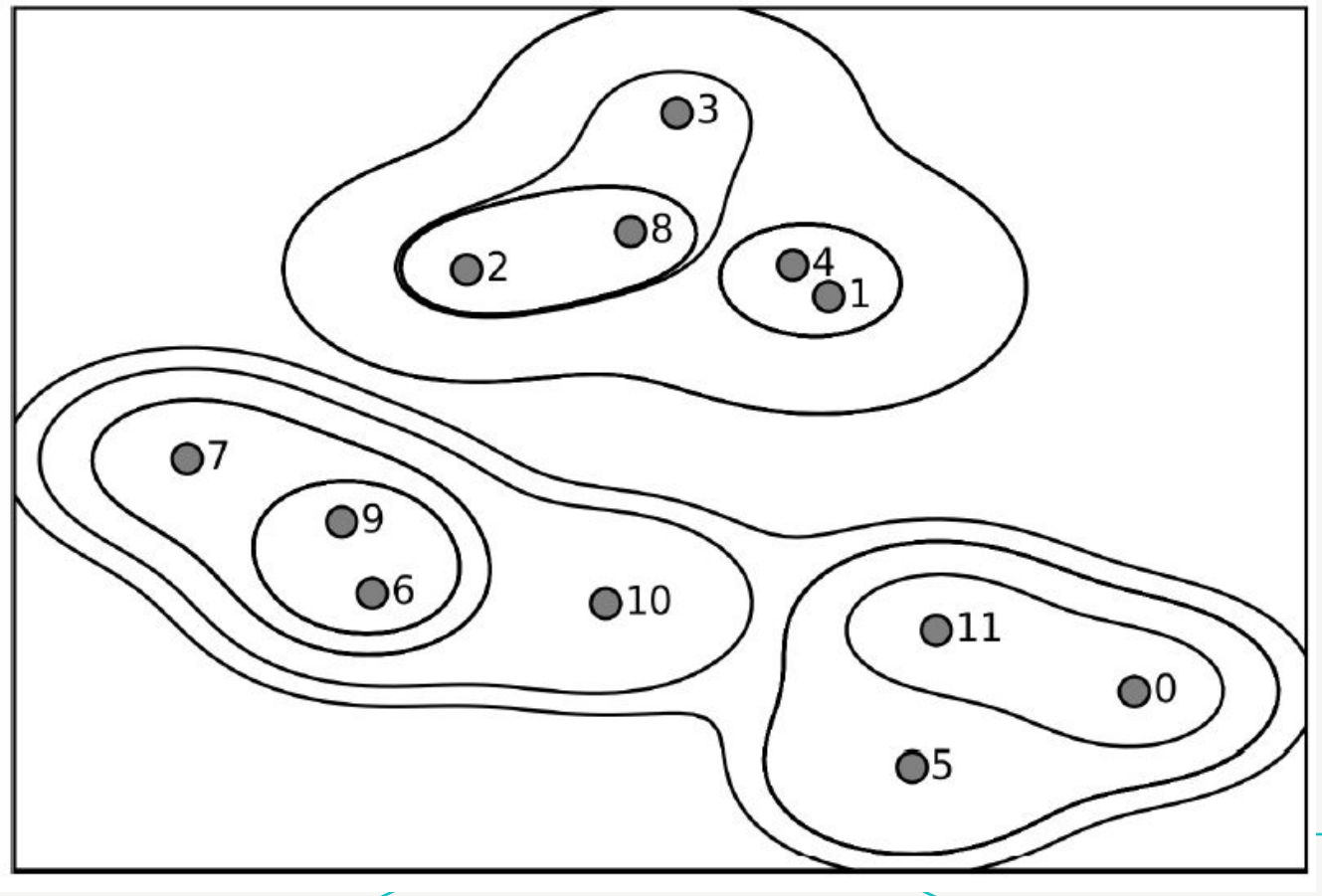


Figure 3-34. Cluster assignment using agglomerative clustering with three clusters

OBS: Se requiere especificar el número de clusters que se quieren encontrar

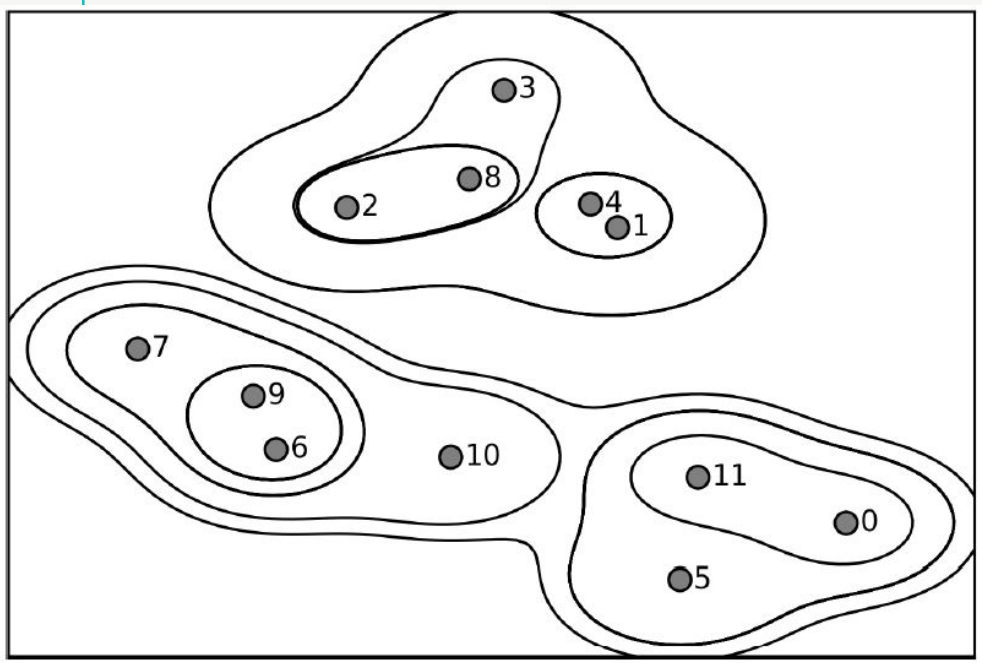
Hierarchical clustering

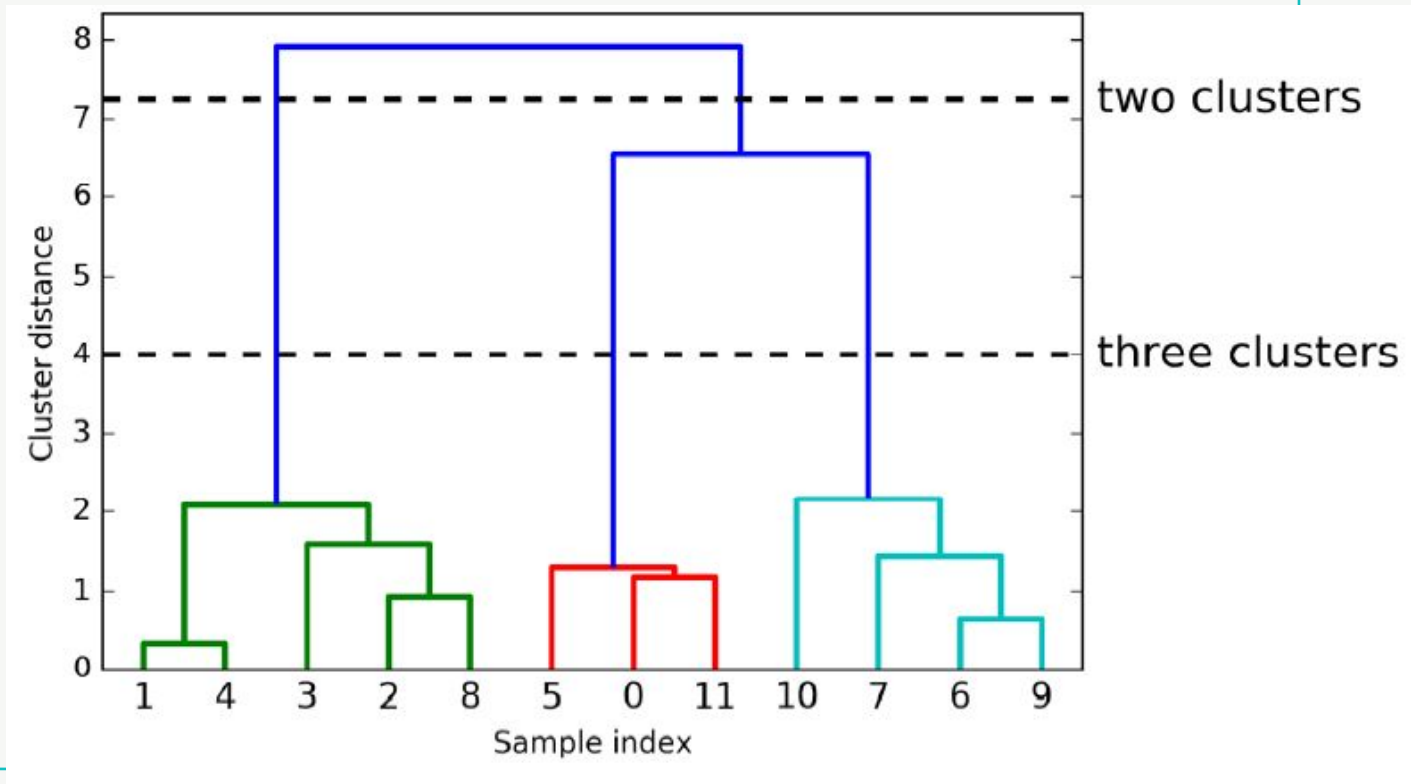
El Agglomerative Clustering produce lo que se conoce como Hierarchical clustering, el cua fusiona clusters de forma iterativa.



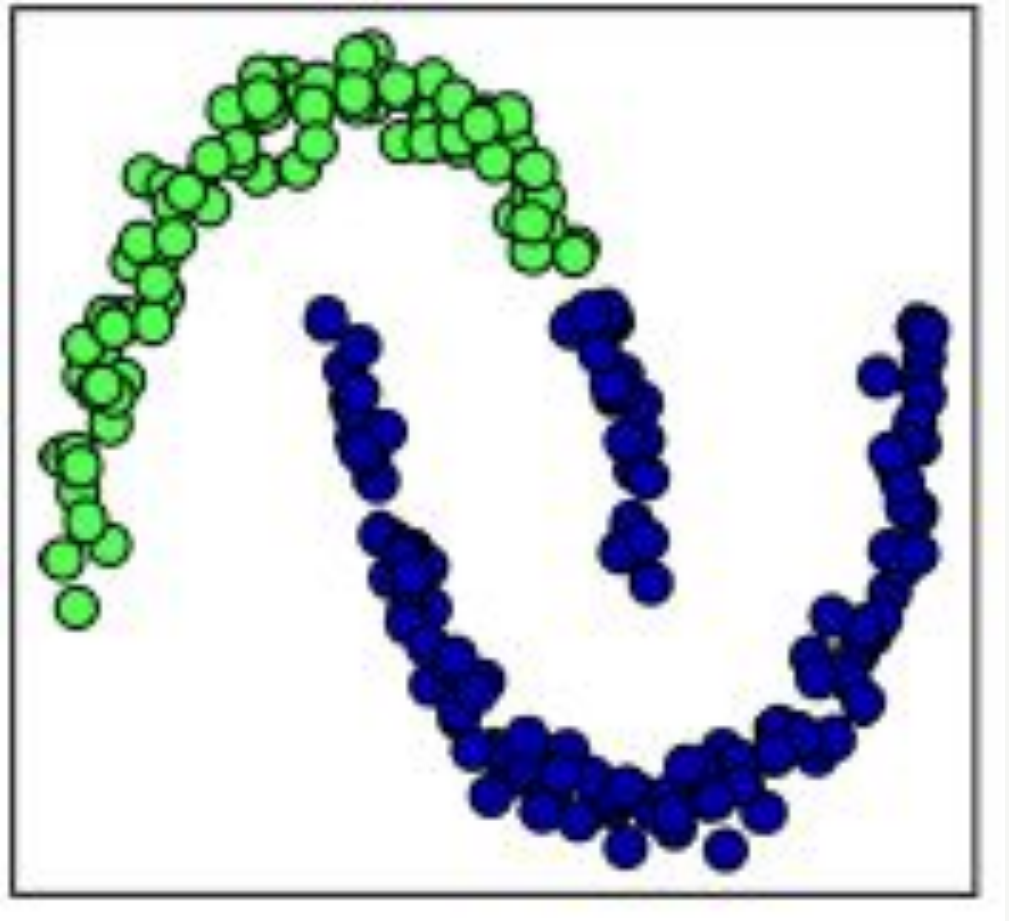
Dendrograma, una herramienta de visualización

Es un diagrama bidimensional, que ilustra las fusiones o divisiones realizadas en cada etapa sucesivas del análisis.





Sin embargo



DBSCAN

significa :“density based spatial clustering of applications with noise”

No requiere saber el número de clusters

puede generar clusters con formas complejas

no es tan rápido como los anteriores

puede trabajar con datos grandes

“reginoes densas forman un cluster, se separan por regiones vacás”

no puede hace rpredicciones

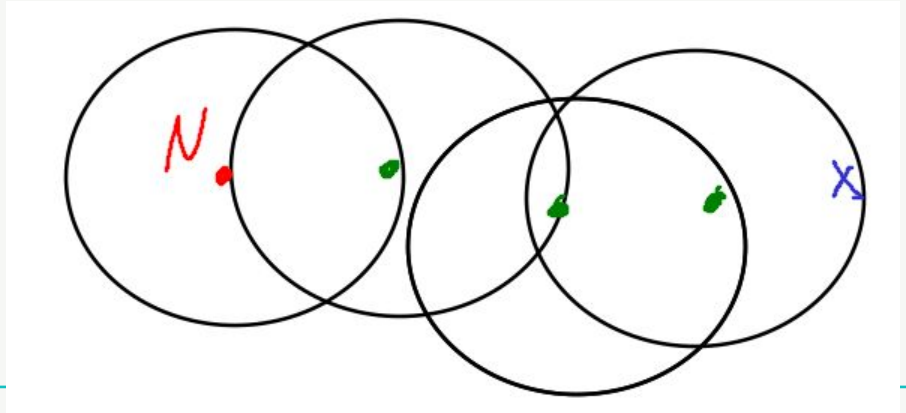
¿Cómo funciona?

Tiene dos parametros: “min_samples” and “eps”. los llamaremos n y epsilon.

Para todo punto del dataset consideramos la bola de radio épsilon alrededor de ese punto (*vecindad*), si tiene al menos de n puntos del dataset, al punto se le dá la etiqueta de “núcleo”.

Todos los puntos dentro de la vecindad de un núcleo son directamente alcanzables.

Un punto es alcanzable si hay una sucesión de puntos tal que el primero es un núcleo y hay una cadena de vecindades, una por cada punto de la sucesión que conectan al núcleo con el último punto



Si un punto no es alcanzable es llamado “ruido”

Cada núcleo define un cluster como el conjunto de puntos alcanzables desde él.

Si dos núcleos son tales que uno es alcanzable desde el otro, ambos clusters se hacen uno

el ruido no pertenece a ningún cluster.

implementación

el ruido es etiquetado como -1, a falta de predicción usamos fit_predict

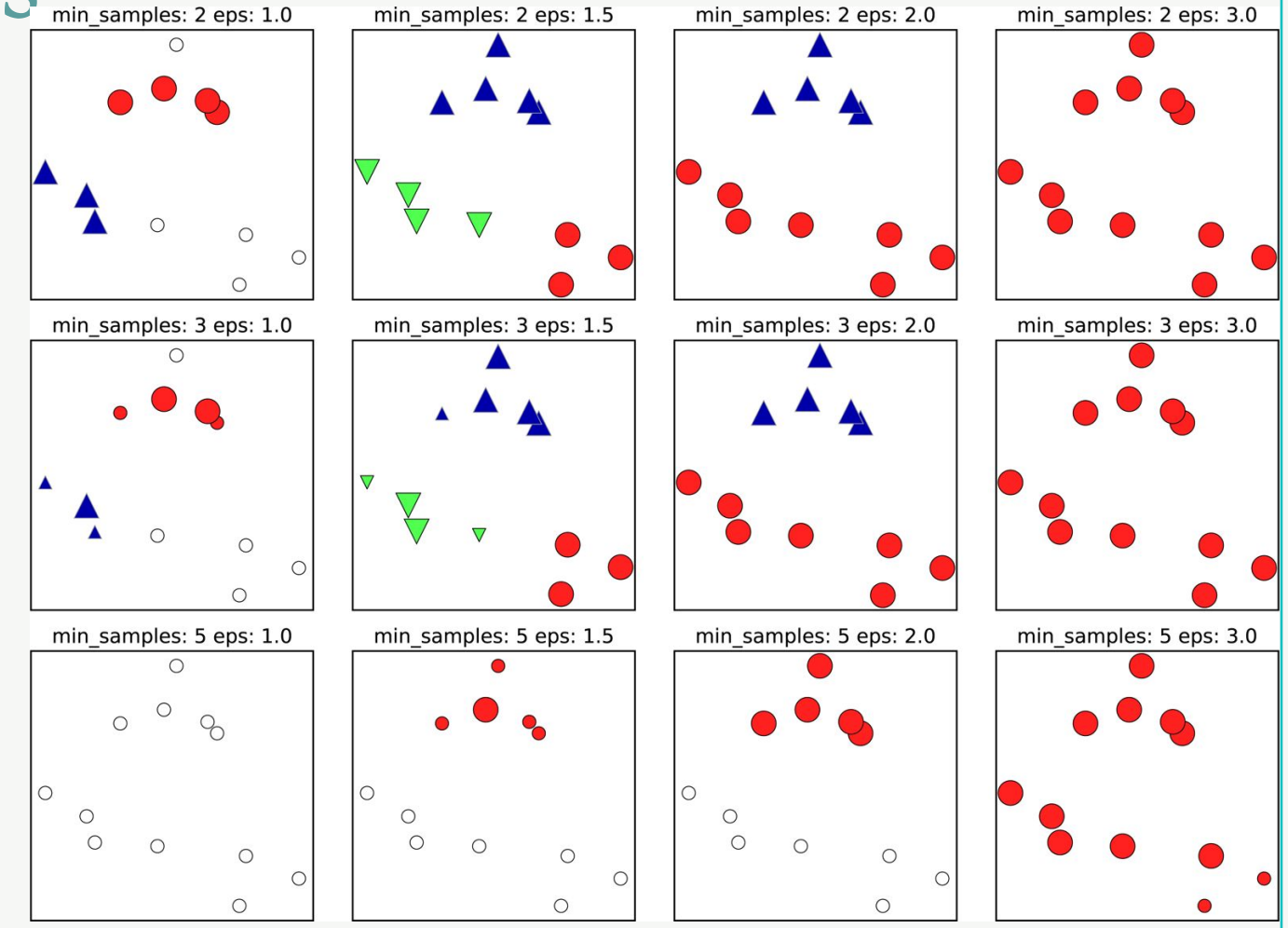
```
from sklearn.cluster import DBSCAN
X, y = make_blobs(random_state=0, n_samples=12)

dbscan = DBSCAN()
clusters = dbscan.fit_predict(X)
print("Cluster memberships:\n{}".format(clusters))
```

```
Cluster memberships:
[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
```

min_samples y eps

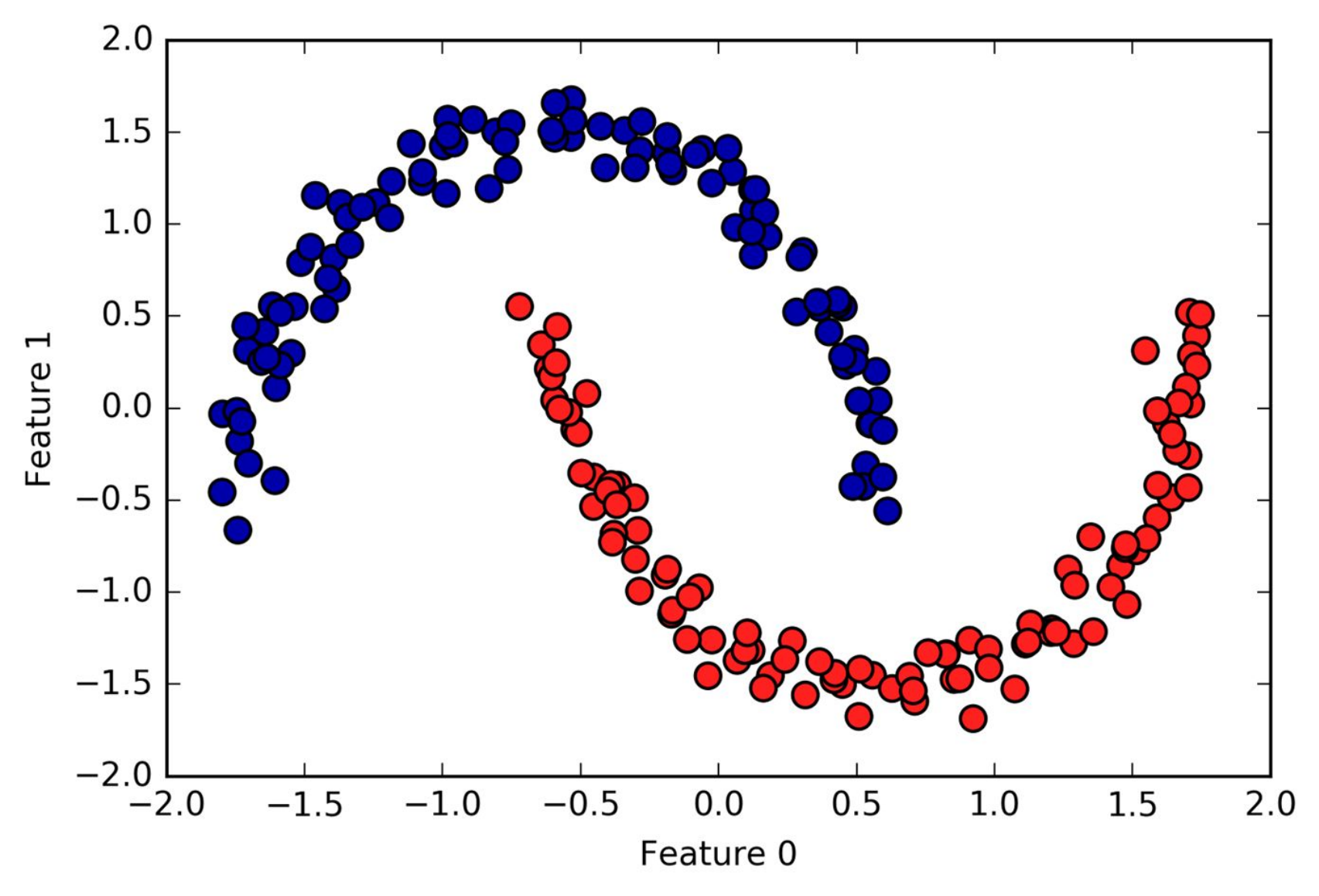
naturalmente, si modificamos n o epsilon el resultado cambiará

para este método nos sirve escalar los datos



Aplicación a dos lunas

```
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# rescale the data to zero mean and unit variance
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

dbscan = DBSCAN()
clusters = dbscan.fit_predict(X_scaled)
# plot the cluster assignments
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mglearn.cm2, s=60)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

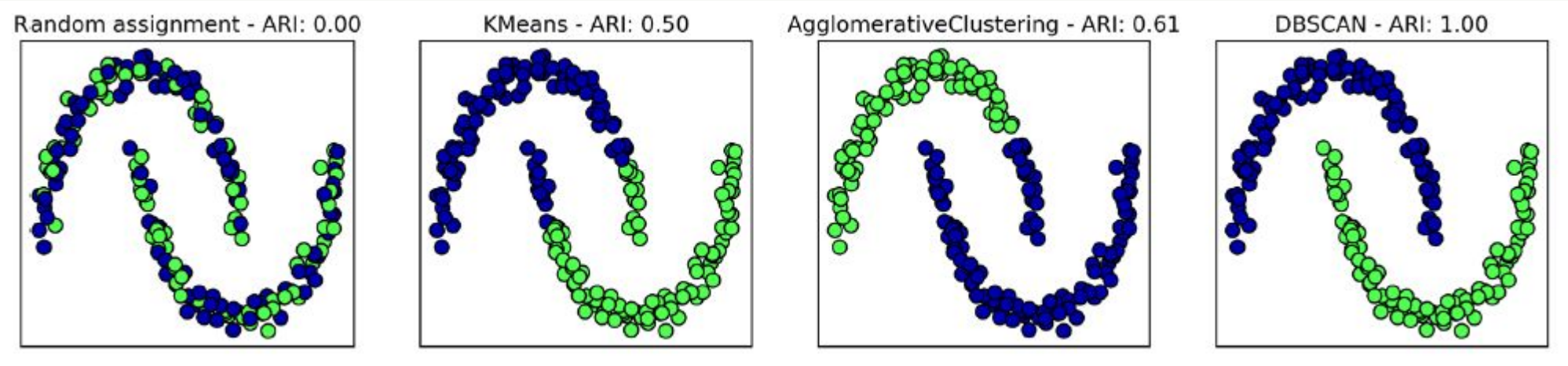

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Feature 1
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Feature 0

Comparar algoritmos de clustering

si conocemos cual es el resultado que esperamos de un algoritmo de clustering podemos calcular que tan bueno es. Para esto tenemos dos algoritmos:

adjusted rand index (ARI) y normalized mutual information (NMI), metricas en [0,1]

```
algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2)
              DBSCAN()]
```



Comparando DBSCAN, K-means y Agglomerative Clustering en faces dataset

1. Escalamos los datos con PCA

In[71]:

```
# extract eigenfaces from lfw data and transform data
from sklearn.decomposition import PCA
pca = PCA(n_components=100, whiten=True, random_state=0)
pca.fit_transform(X_people)
X_pca = pca.transform(X_people)
```

Aplicando DBSCAN

Aplicando K-means