A. Clustering by density

The mean shift clustering algorithm in the previous chapter usually performs sufficiently well and can choose a reasonable number of clusters. However, it is not very scalable due to computation time and still makes the assumption that clusters have a "blob"-like shape (although this assumption is not as strong as the one made by K-means).

Another clustering algorithm that also automatically chooses the number of clusters is DBSCAN. DBSCAN clusters data by finding *dense* regions in the dataset. Regions in the dataset with many closely packed data observations are considered *high-density* regions, while regions with sparse data are considered *low-density* regions.

The DBSCAN algorithm treats high-density regions as clusters in the dataset, and low-density regions as the area between clusters (so observations in the low-density regions are treated as noise and not placed in a cluster).

High-density regions are defined by *core samples*, which are just data observations with many neighbors. Each cluster consists of several core samples and all the observations that are neighbors to a core sample.

Unlike the mean shift algorithm, the DBSCAN algorithm is both highly scalable and makes no assumptions about the underlying shape of clusters in the dataset.

B. Neighbors and core samples

The exact definition of "neighbor" and "core sample" depends on what we want in our clusters. We specify the maximum distance, ε, between two data observations that are considered neighbors. Smaller distances result in smaller and more tightly packed clusters. We also specify the minimum number of points in the neighborhood of a data observation for the observation to be considered a core sample (the neighborhood consists of the data observation and all its neighbors).

In scikit-learn, we implement DBSCAN with the `DBSCAN` object (part of the `cluster` module). The object is initialized with the keyword arguments `eps` (representing the value of ε) and `min_samples` (representing the minimum size of a core sample's neighborhood).

The code below demonstrates how to use the `DBSCAN` object, with ε equal to 1.2 and a minimum size of 30 for a core sample's neighborhood.

```python
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=1.2, min_samples=30)
# predefined data
dbscan.fit(data)

# cluster assignments
print('{}\n'.format(repr(dbscan.labels_)))

# core samples
print('{}\n'.format(repr(dbscan.core_sample_indices_)))
num_core_samples = len(dbscan.core_sample_indices_)
print('Num core samples: {}\n'.format(num_core_samples))
```

RUN SAVE RESET

Close

Output 2.004s

```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  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,  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])
```

In the code above, we used `DBSCAN` to cluster the 150 data observations in `data`. The algorithm found two clusters. In this case all the data observations fit in a cluster, but in general the non-cluster observations would be labeled with `-1`.

The `core_sample_indices_` attribute represents the core sample data observations in `data` (specified by row index).