Advanced Artificial Intelligence

Lab 10

Outline

- Introduction details of different clustering algorithms
- Compare different clustering algorithms on a concrete problem
- Exercise

Introduction details of different clustering algorithms

Partitional clustering

- *K*-means & *K*-means++ clustering

Spectral clustering

Hierarchical clustering

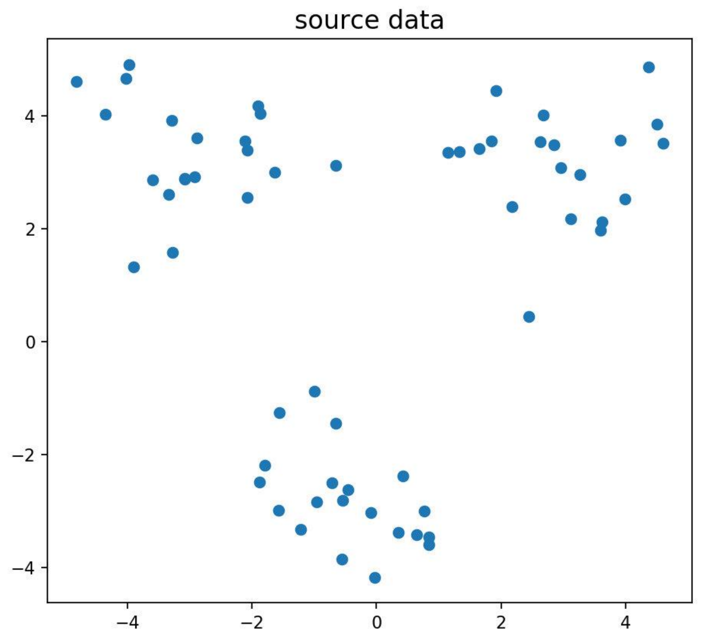
- Bottom-Up (Agglomerative)
- top-down (Divisive)

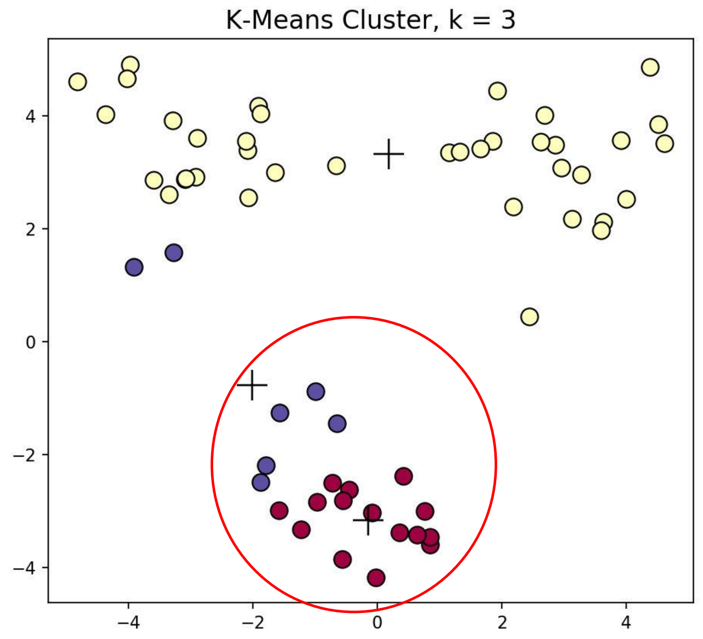
K-means Clustering

- **Algorithm** *K-means clustering*
- 1. Decide on a value for *K*, the number of clusters.
- 2. Initialize the *K* cluster centers (randomly, if necessary).
- 3. Decide the class memberships of the *N* objects by assigning them to the nearest cluster center.
- 4. Re-estimate the *K* cluster centers, by assuming the memberships found above are correct.
- 5. Repeat 3 and 4 until none of the *N* objects changed membership in the last iteration.

K-means Clustering

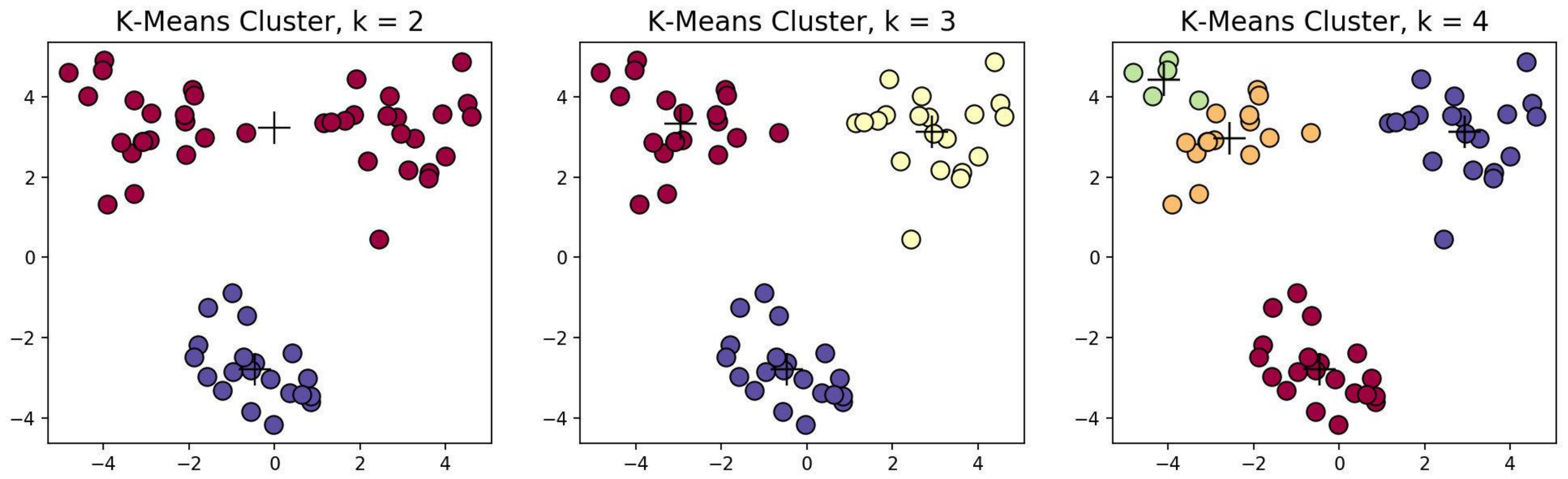
When the **initial centers** are different, the final clustering result is different. In this example, the data below (in the red circle) should be in two clusters, due to the unreasonable selection of the initial centers.





K-means Clustering

The value of *K* also has a great impact on the results.



K-means Clustering

Strength:

- Simple and efficient
- Fast convergence

Weakness:

- Need to determine the value of k in advance
- Sensitive to the initial centroid point
- Sensitive to abnormal data

K-means++ Clustering

- *K*-means++ clustering is an optimized algorithm for the selection of initial centroid points in k-means. The flow of the algorithm is similar to *K*-means clustering, the only change is the **selection of the initial centroid**.

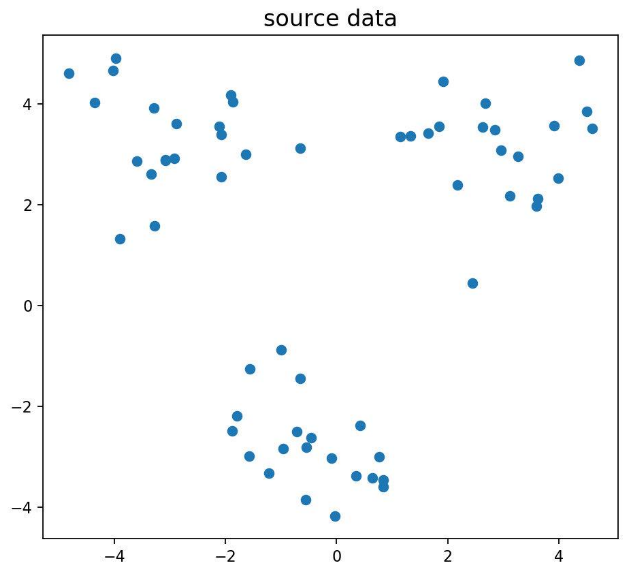
K-means++ Clustering

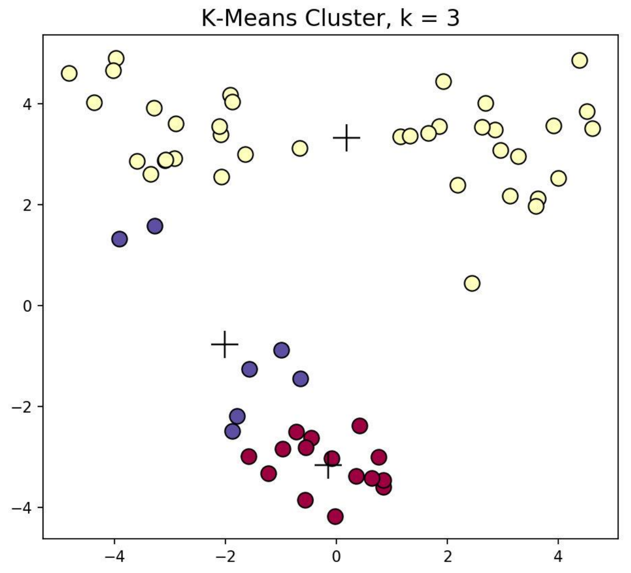
- The algorithm is as follows:

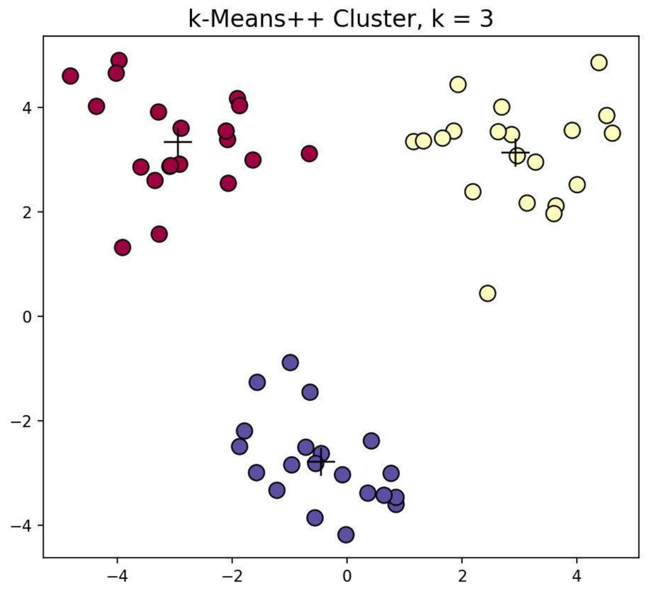
1. Choose one center uniformly at random among the data points.
2. For each data point x not chosen yet, compute D(x), the **distance** between x and the nearest center that has already been chosen.
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to D(x)^2.
4. Repeat Steps 2 and 3 until k centers have been chosen.
5. Now that the initial centers have been chosen, proceed using standard *K*-means clustering.

K-means++ Clustering

Optimization of initial centroid point selection

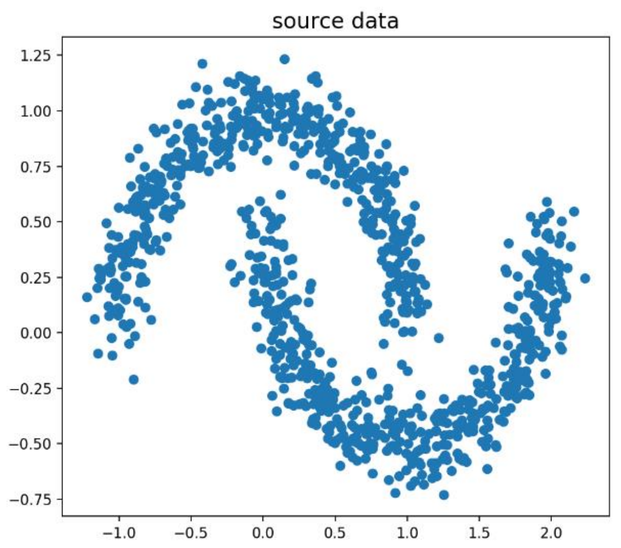


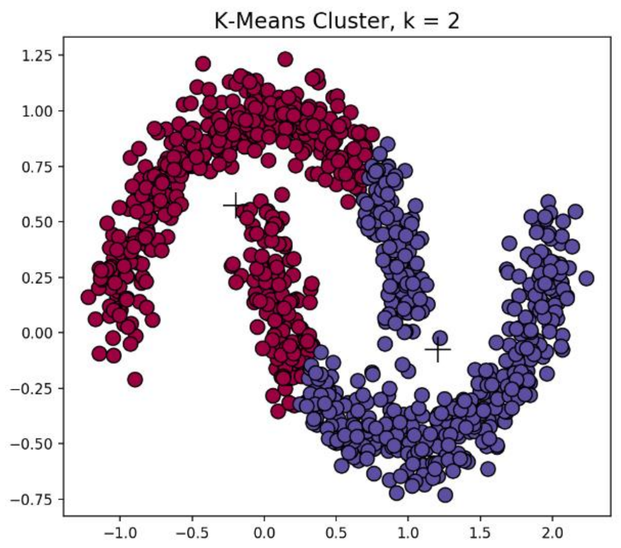




K-means & *K*-means++ Clustering

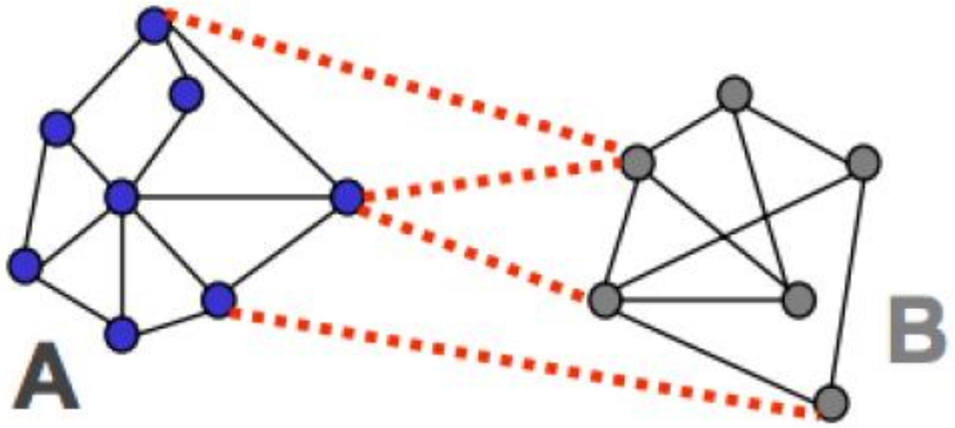
- The *K*-means clustering algorithm has a good effect on convex data. The data can be divided into spherical clusters according to distance, but it can do nothing for data points with non-convex shapes.





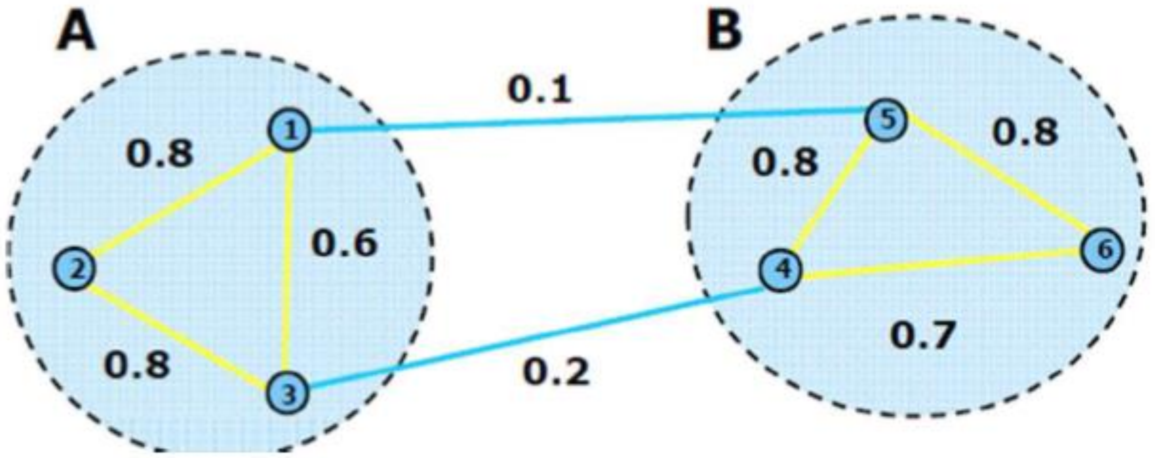
Spectral clustering

- Data points are represented as nodes in a weighted graph.
- The edges are weighted by their pairwise similarity.
- Core idea: partition the nodes into two subsets A and B s.t. the cut size (the sum of the weights assigned to the edges connecting between nodes in A and B) is minimized.



Spectral clustering

$$CutSize(A,B) = \sum_{x \in A} \sum_{x' \in B} weight(x, x')$$



Spectral clustering

- It is common to use a Gaussian Kernel to compute similarity between objects

$$weight(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp\frac{-\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|^2}{\sigma^2}$$

One could create:

- a fully connected graph;
- k-nearest neighbour graph (each node is only connected to its k-nearest neighbours
- ϵ-neighborhood

Normalized Cut

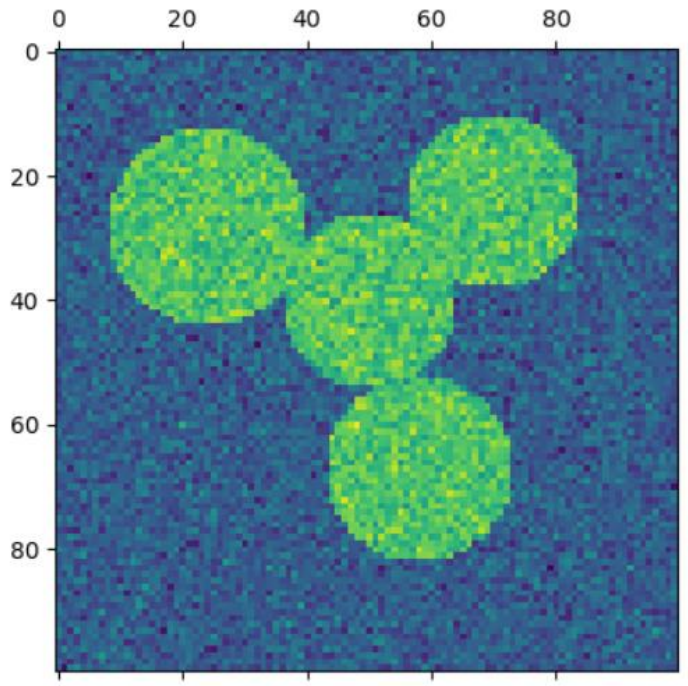
- An efficient approximate graph-cut based clustering algorithm.
- Instead of looking at the value of total edge weight connecting $A$and B, the proposed measure computes the cut cost as a fraction of the total edge connections. This is called **disassociation measure the normalized cut**:

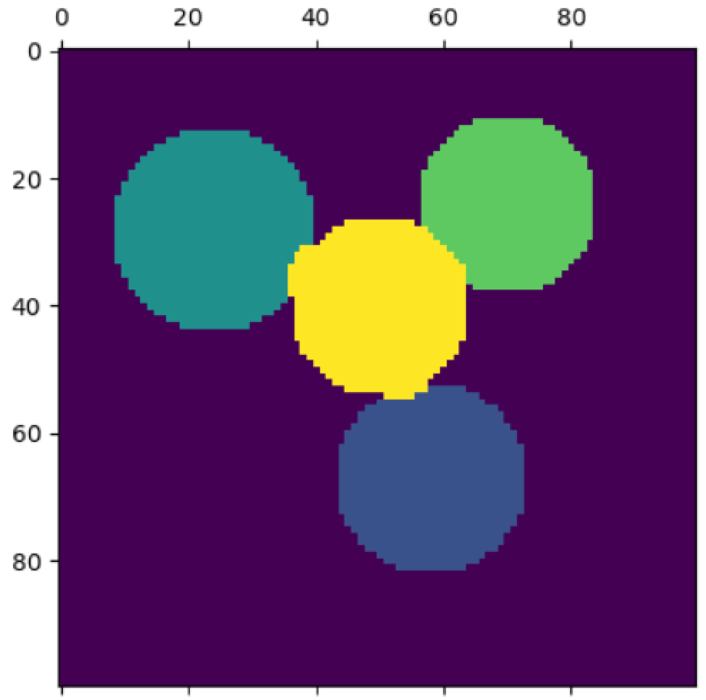
$$Ncut(A,B) = \frac{CutSize(A,B)}{assoc(A,V)} + \frac{CutSize(A,B)}{assoc(B,V)},$$

where $assoc(A,V) = \sum_{x \in A, x' \in V} weight(x,x')$ is the total connection from nodes in A to all nodes in the graph. $assoc(B,V)$ is similarly defined.

An Example of Spectral Clustering

- In this example, an image with connected circles is generated and spectral clustering (normalized graph cuts) is used to separate circles.

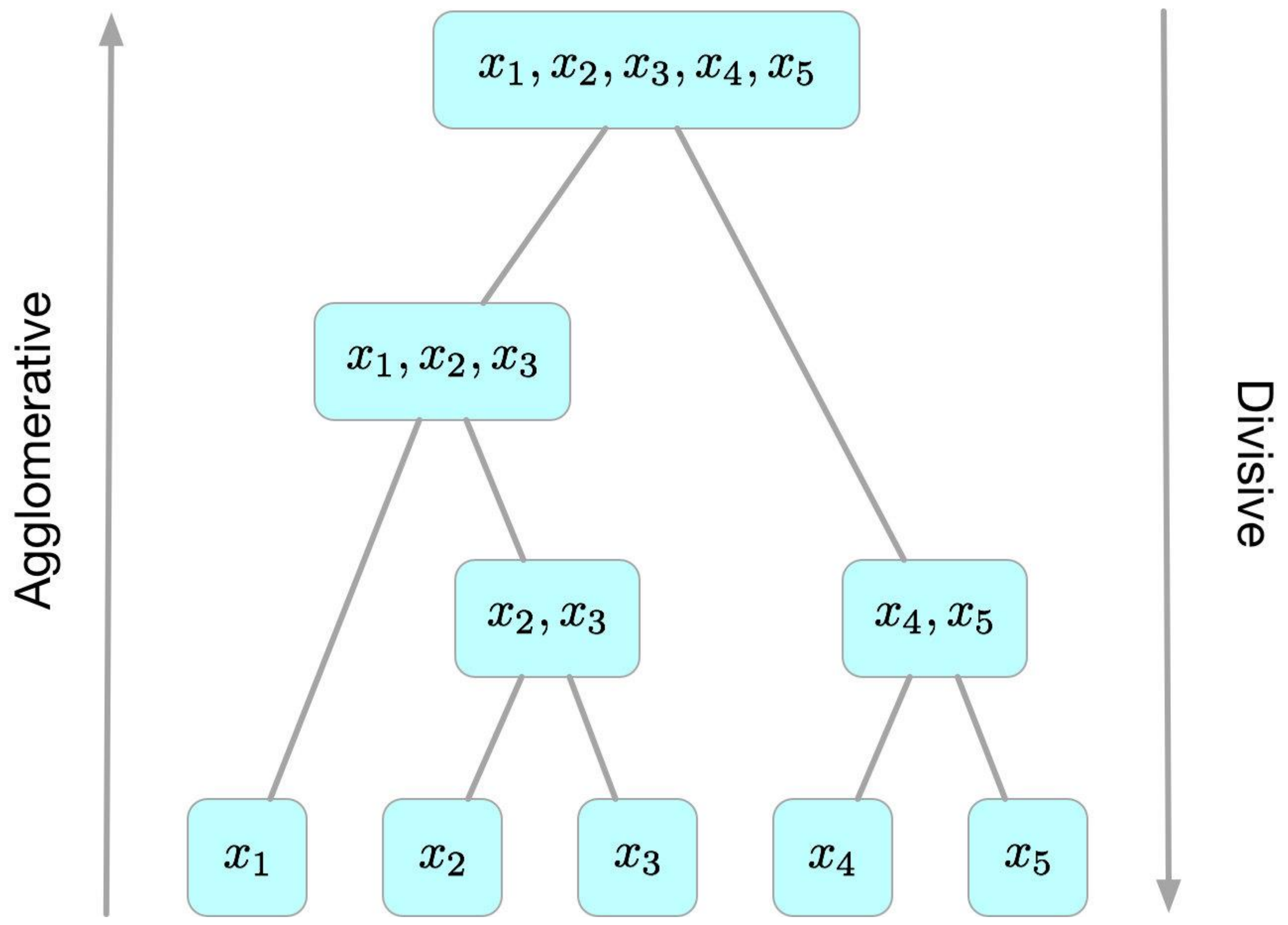




Hierarchical Clustering

- Hierarchical clustering algorithms recursively find nested clusters
- either in **agglomerative mode** (starting with each data point in its own cluster and merging the most similar pair of clusters successively to form a cluster hierarchy);
- or in **divisive (top-down) mode** (starting with all the data points in one cluster and recursively dividing each cluster into smaller clusters).

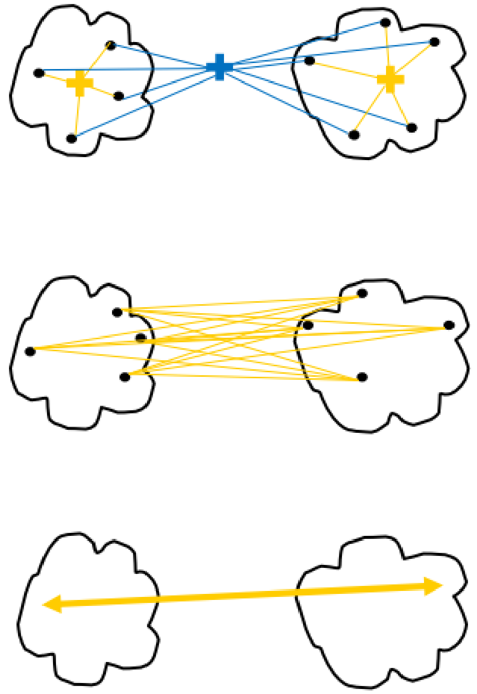
Example

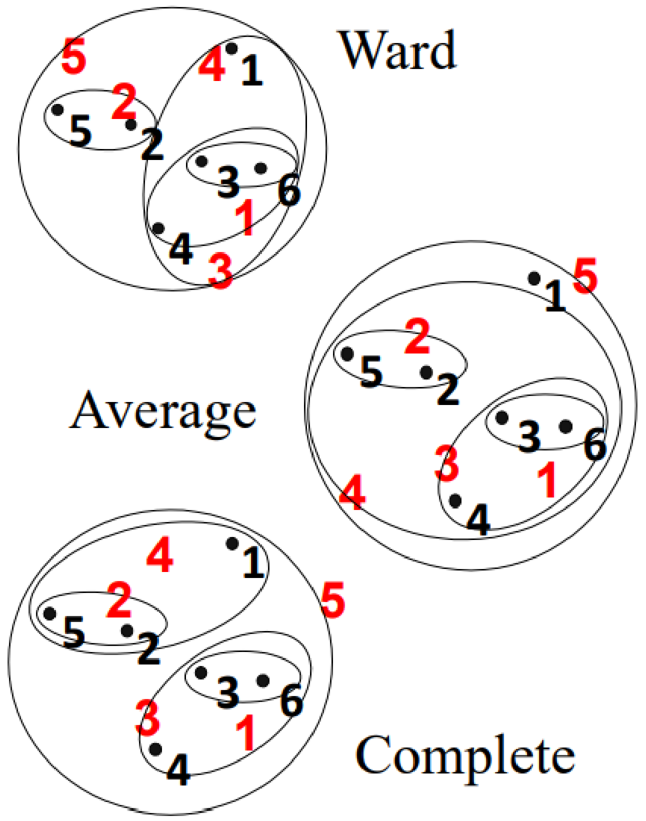

x_1, x_2, x_3, x_4, x_5
x_1, x_2, x_3
x_2, x_3
x_4, x_5
x_1
x_2
x_3
x_4
x_5
Agglomerative
Divisive

Agglomerative Clustering

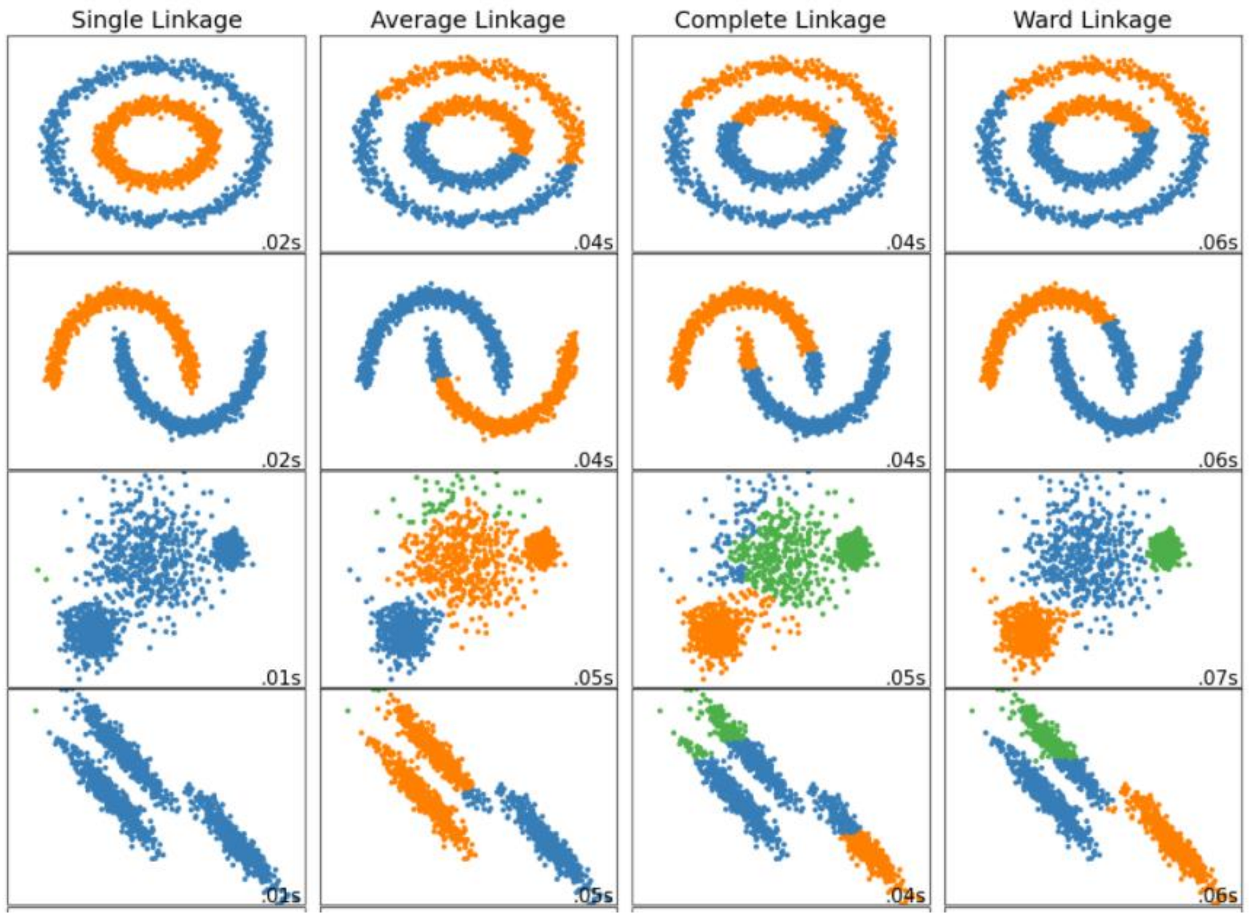
Linkage Criteria for Agglomerative Clustering

- **Ward's method**
  - *Least increase in total variance (around cluster centroids)*
- **Average linkage**
  - *Average distance between clusters*
- **Complete linkage**
  - *Max distance between clusters*





Agglomerative Clustering



Compare different clustering algorithms on a concrete problem

Compare different clustering algorithms on a concrete problem

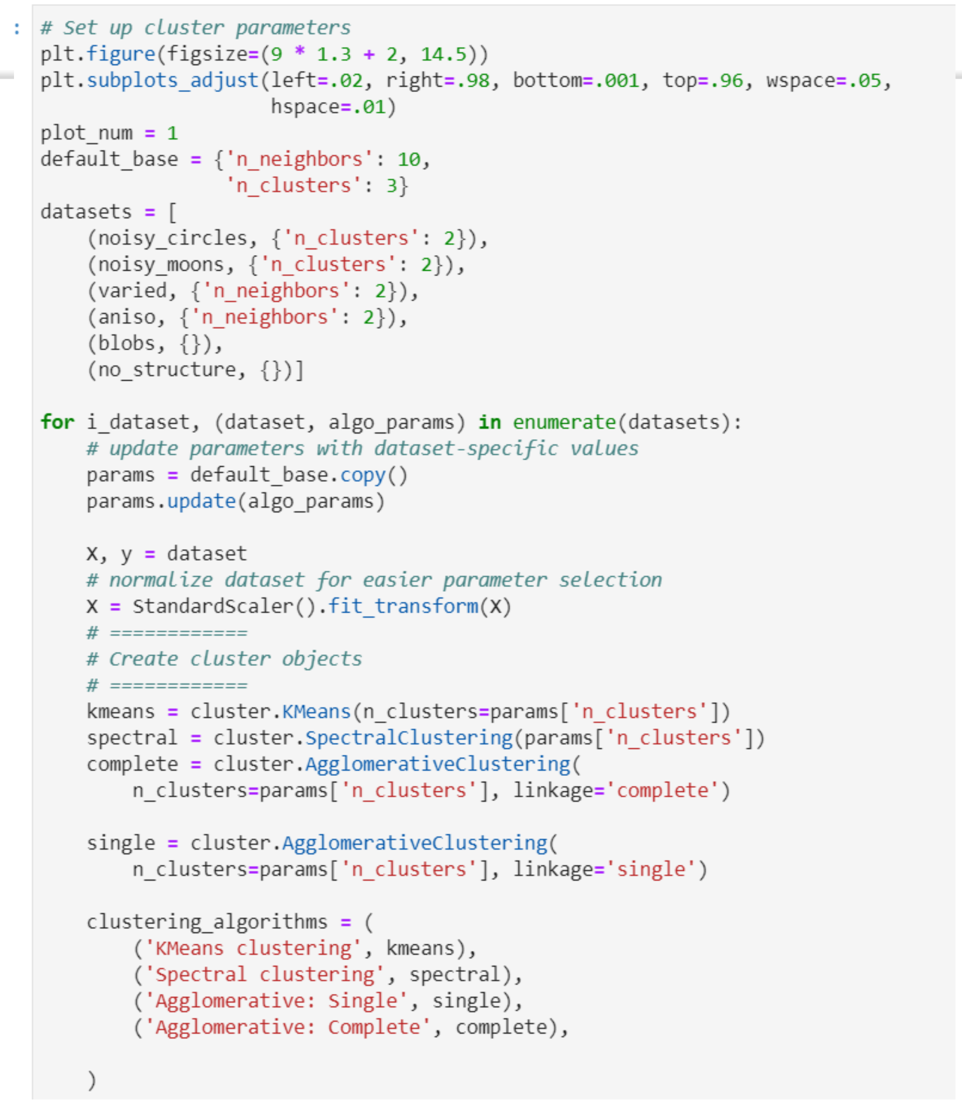
- Comparing different clustering methods on toy datasets：

1. *K*-means clustering
2. spectral clustering
3. Agglomerative Clustering

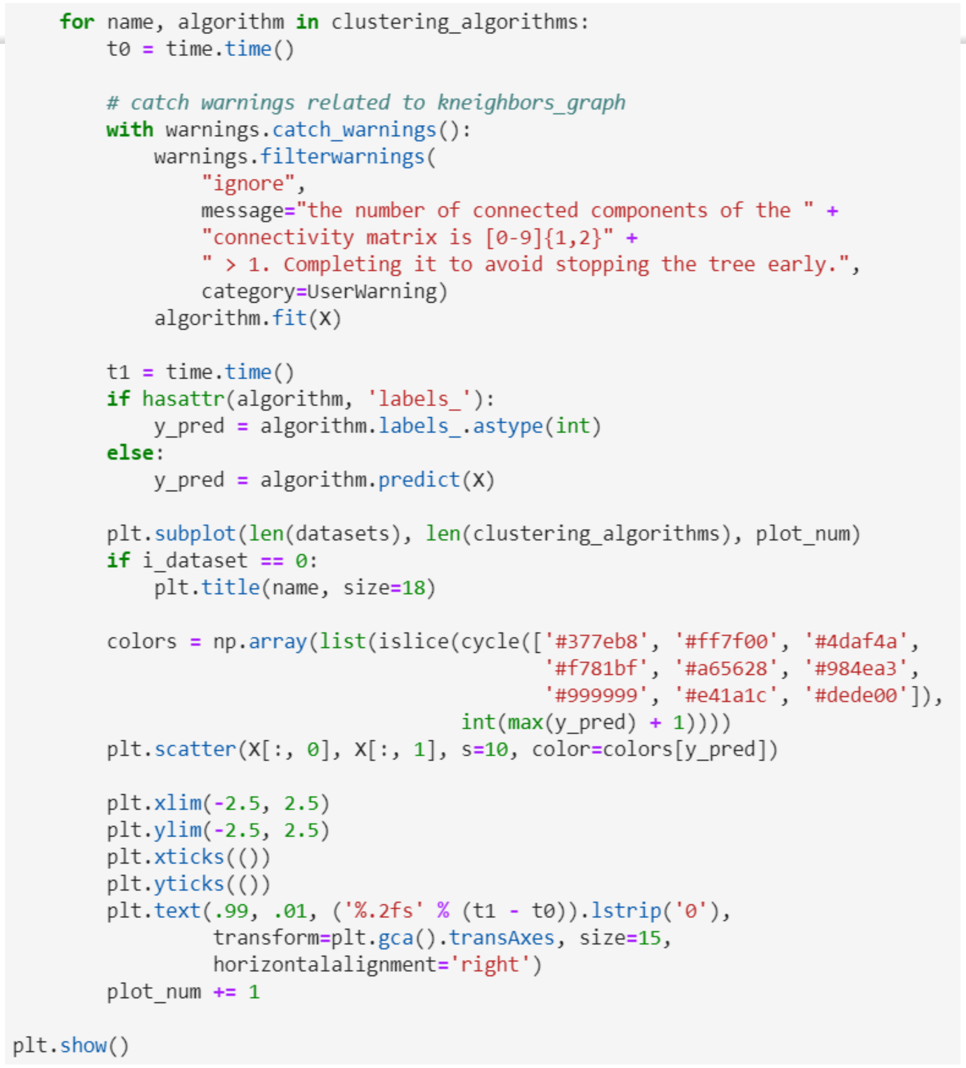
Dataset

Generate datasets. We choose the size big enough to see the scalability of the algorithms, but not too big to avoid too long running times

```
n_samples = 1500
noisy_circles = datasets.make_circles(n_samples=n_samples, factor=.5,
                                      noise=.05)
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=.05)
blobs = datasets.make_blobs(n_samples=n_samples, random_state=8)
no_structure = np.random.rand(n_samples, 2), None

# Anisotropicly distributed data
random_state = 170
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X, transformation)
aniso = (X_aniso, y)

# blobs with varied variances
varied = datasets.make_blobs(n_samples=n_samples,
                             cluster_std=[1.0, 2.5, 0.5],
                             random_state=random_state)
```

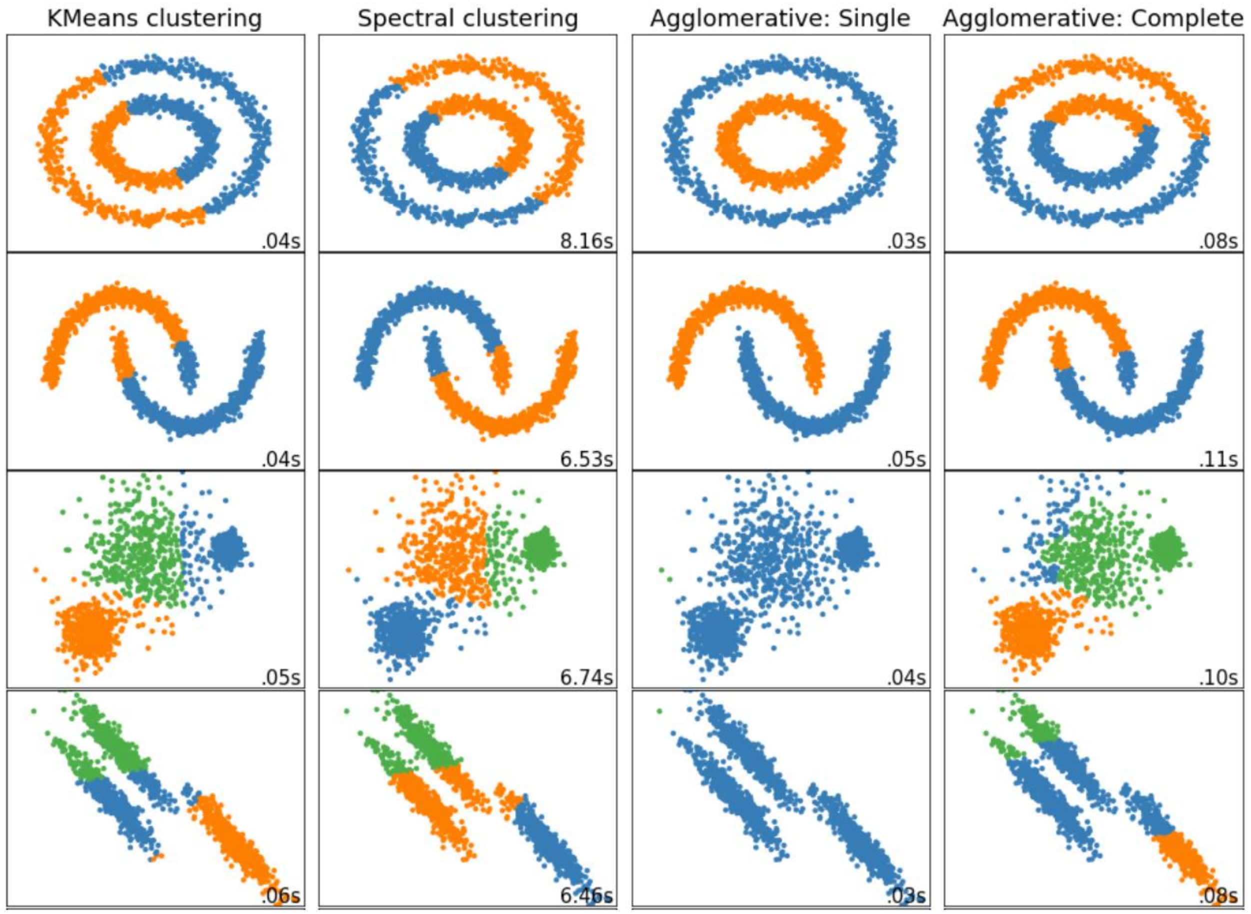


```python
# Set up cluster parameters
plt.figure(figsize=(9 * 1.3 + 2, 14.5))
plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
                    hspace=.01)
plot_num = 1
default_base = {'n_neighbors': 10,
                'n_clusters': 3}
datasets = [
    (noisy_circles, {'n_clusters': 2}),
    (noisy_moons, {'n_clusters': 2}),
    (varied, {'n_neighbors': 2}),
    (aniso, {'n_neighbors': 2}),
    (blobs, {}),
    (no_structure, {})]

for i_dataset, (dataset, algo_params) in enumerate(datasets):
    # update parameters with dataset-specific values
    params = default_base.copy()
    params.update(algo_params)

    X, y = dataset
    # normalize dataset for easier parameter selection
    X = StandardScaler().fit_transform(X)
    # ============
    # Create cluster objects
    # ============
    kmeans = cluster.KMeans(n_clusters=params['n_clusters'])
    spectral = cluster.SpectralClustering(params['n_clusters'])
    complete = cluster.AgglomerativeClustering(
        n_clusters=params['n_clusters'], linkage='complete')

    single = cluster.AgglomerativeClustering(
        n_clusters=params['n_clusters'], linkage='single')

    clustering_algorithms = (
        ('KMeans clustering', kmeans),
        ('Spectral clustering', spectral),
        ('Agglomerative: Single', single),
        ('Agglomerative: Complete', complete),

    )
```



```python
    for name, algorithm in clustering_algorithms:
        t0 = time.time()

        # catch warnings related to kneighbors_graph
        with warnings.catch_warnings():
            warnings.filterwarnings(
                "ignore",
                message="the number of connected components of the " +
                "connectivity matrix is [0-9]{1,2}" +
                " > 1. Completing it to avoid stopping the tree early.",
                category=UserWarning)
            algorithm.fit(X)

        t1 = time.time()
        if hasattr(algorithm, 'labels_'):
            y_pred = algorithm.labels_.astype(int)
        else:
            y_pred = algorithm.predict(X)

        plt.subplot(len(datasets), len(clustering_algorithms), plot_num)
        if i_dataset == 0:
            plt.title(name, size=18)

        colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',
                                             '#f781bf', '#a65628', '#984ea3',
                                             '#999999', '#e41a1c', '#dede00']),
                                      int(max(y_pred) + 1))))
        plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[y_pred])

        plt.xlim(-2.5, 2.5)
        plt.ylim(-2.5, 2.5)
        plt.xticks(())
        plt.yticks(())
        plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
                 transform=plt.gca().transAxes, size=15,
                 horizontalalignment='right')
        plot_num += 1

plt.show()
```

Results


KMeans clustering
Spectral clustering
Agglomerative: Single
Agglomerative: Complete
.04s
8.16s
.03s
.08s
.04s
6.53s
.05s
.11s
.05s
6.74s
.04s
.10s
.06s
6.46s
.03s
.08s

Results

- single linkage is fast, and can perform well on non-globular data, but it performs poorly in the presence of noise.
- Spectral clustering is slow
- *K*-means clustering performs well in the presence of noise but not in non-globular data.

Exercise

- To implement the instances of the Section Clustering in Sklearn
- https://scikit-learn.org/stable/modules/clustering.html#