CS5489 - Machine Learning

Lecture 7b - Unsupervised Learning - Clustering

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

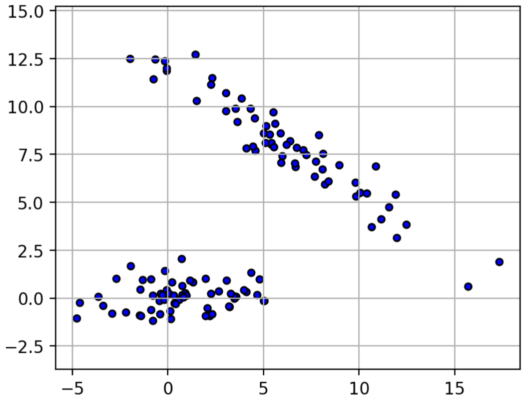
1. Unsupervised Learning
2. Parametric clustering
    A. K-means
    B. Gaussian mixture models (GMMs)
    C. Bayesian GMMs
3. **Non-parametric clustering and Mean-shift**
4. Spectral clustering

Non-parametric densities

- Suppose we have samples $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$
    - We want to estimate a probability density without assuming a parametric model (e.g., Gaussian)

In [12]:
```
fig
```

Out[12]:

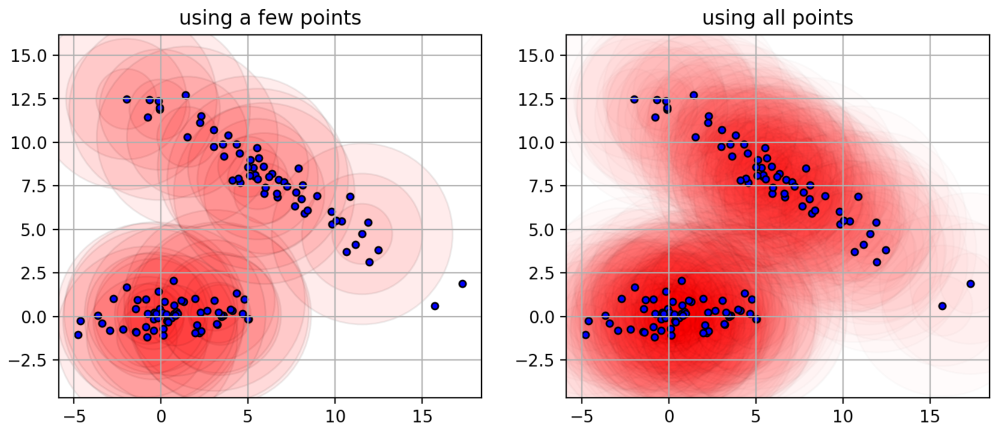


Non-parametric estimation

- Idea: put a small Gaussian at each data point, and sum it up.
    - each point contributes locally to the probability density.
    - $p(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^N \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \sigma^2\mathbf{I})$
        - σ is the bandwidth of the Gaussian.

In [14]:
```
fig
```

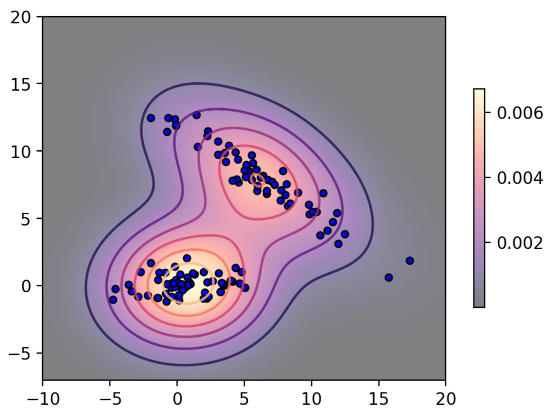
Out[14]:



Kernel density estimator

- This is called a *kernel density estimator*
  - the *kernel* is the small Gaussian.

In [15]:
```
kde = neighbors.KernelDensity(bandwidth=3.0).fit(X)
plot_scores(kde, axbox, 'magma')
plt.scatter(X[:,0], X[:,1], c='b', s=ssize, edgecolors='k');
```



Clustering using KDE

- The modes of the KDE can be considered the cluster centers.
  - A mode is a local maximum of the probability density.
  - The number of clusters is selected automatically according to the data.

- *How to find the cluster centers?*
  - mode: $\boldsymbol{\mu} = \mathrm{argmax}\, p(\mathbf{x})$
  - select a point, and run gradient ascent on $p(\mathbf{x})$.
    - $\hat{\boldsymbol{\mu}} \leftarrow \hat{\boldsymbol{\mu}} + \eta \frac{d}{d\mathbf{x}} p(\mathbf{x})$
  - using a clever choice of η, we get an algorithm that is guaranteed to converge called the "Mean-shift algorthm".

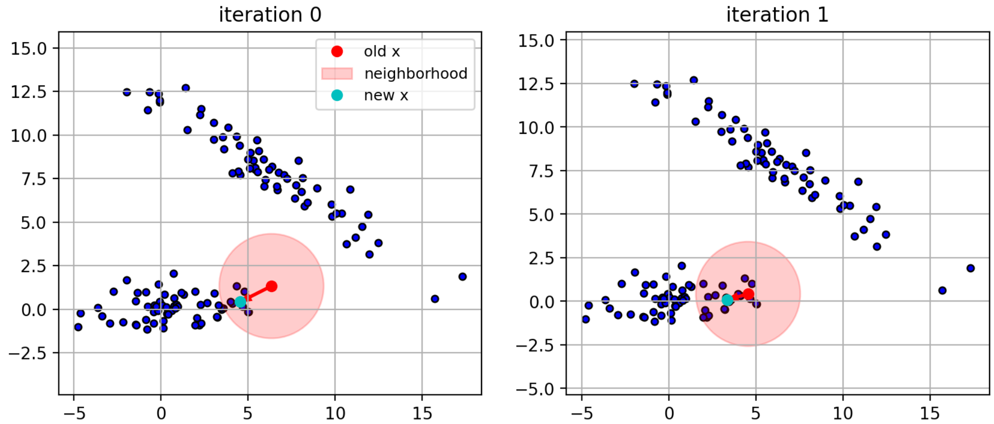
Mean-shift algorithm

- **Idea:** iteratively shift towards the largest concentration of points.
  - start from an initial point $\mathbf{x}$ (e.g., one of the data points).
  - repeat until $\mathbf{x}$ is unchanged:
    - 1. find the nearest neighbors to $\mathbf{x}$ within some radius (bandwidth)
      - according to the Gaussian kernels.

- 2. set $\mathbf{x}$ to be the mean of the neighbor points.

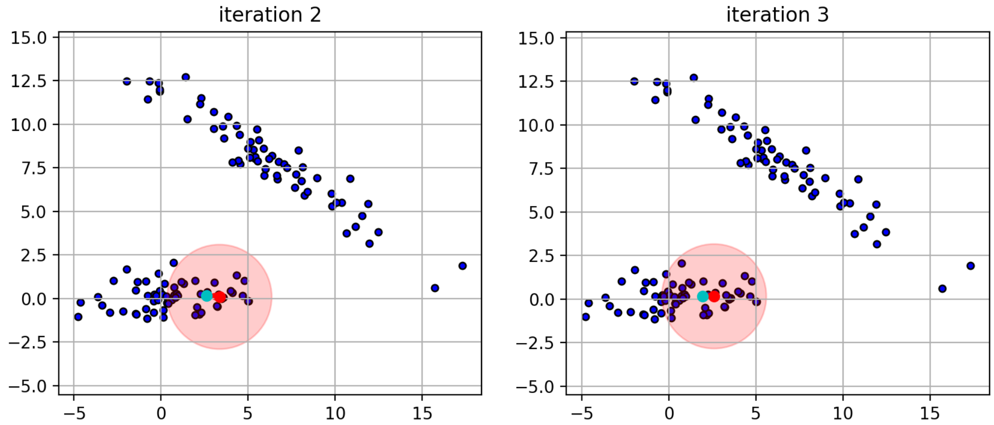
In [17]:
```
efigs[0]
```

Out[17]:



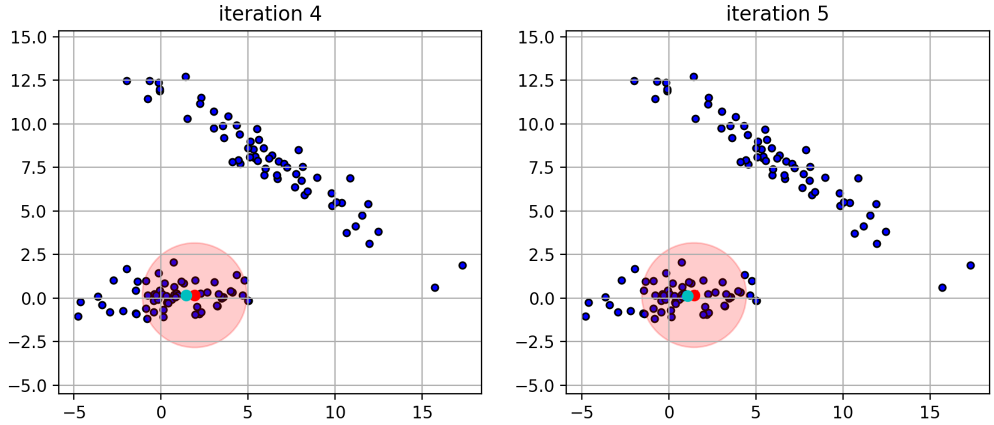
In [18]:
```
efigs[1]
```

Out[18]:



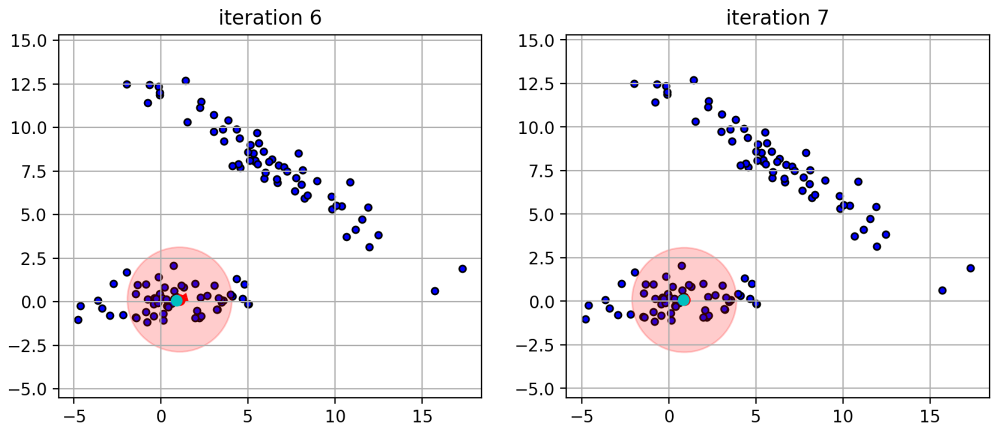
In [19]:
```
efigs[2]
```

Out[19]:



In [20]:
```
efigs[3]
```

Out[20]:



Getting the clusters

- Run the mean-shift algorithm for many initial points $\{\mathbf{x}_i\}$.
  - the set of converged points contain the cluster centers.
    - need to remove the duplicate centers.
  - data points that converge to the same center belong to the same cluster.
  - different initializations can run in parallel (`n_jobs`)

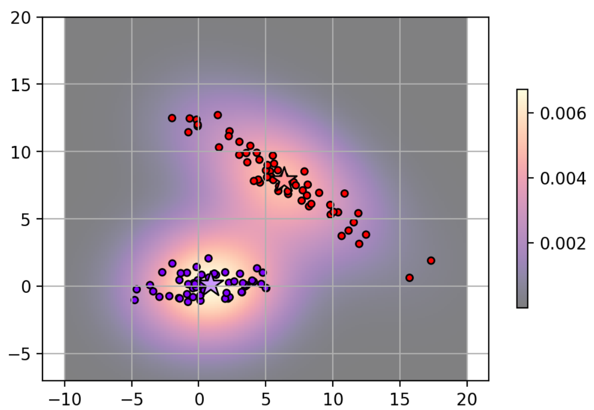
In [21]:

```
# bin_seeding=True -- coarsely uses data points as initial points
ms = cluster.MeanShift(bandwidth=5, bin_seeding=True, n_jobs=-1)
Y = ms.fit_predict(X)

cc = ms.cluster_centers_  # cluster centers

plot_scores(kde, axbox, 'magma', showcontour=False)
plot_clusters(ms, axbox, X, Y, rbow, rbow2)
```

Out[21]: (2,)

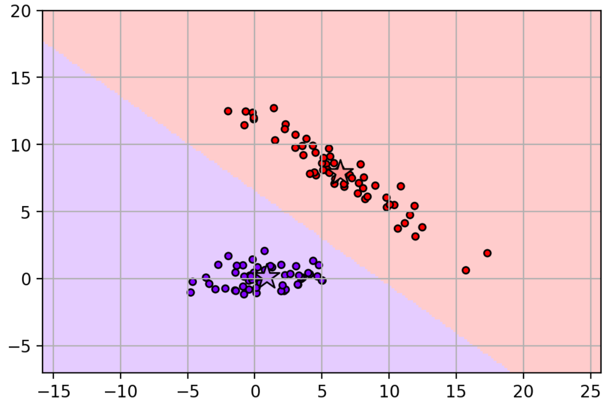


- Cluster partitions
  - assign point based on convergence to same cluster center

In [22]:

```
plot_clusters(ms, axbox, X, Y, rbow, rbow2, showregions=True)
```

Out[22]: (2,)

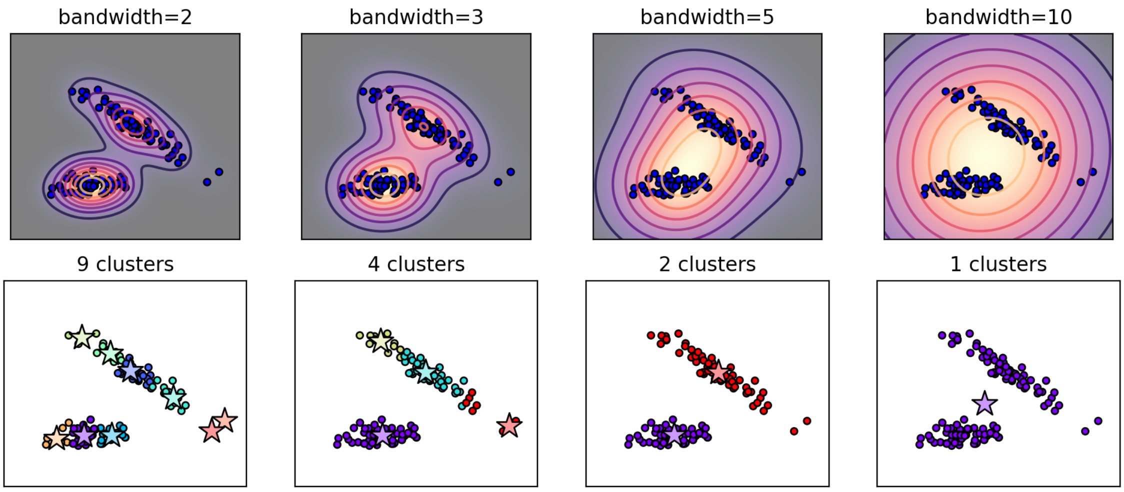


Number of clusters

- Number of clusters is implicitly controlled by the bandwidth (radius of the nearest-neighbors)
  - larger bandwidth creates less clusters
    - focuses on global large groups
  - smaller bandwidth creates more clusters
    - focuses on local groups.

In [24]:
```
msfig
```

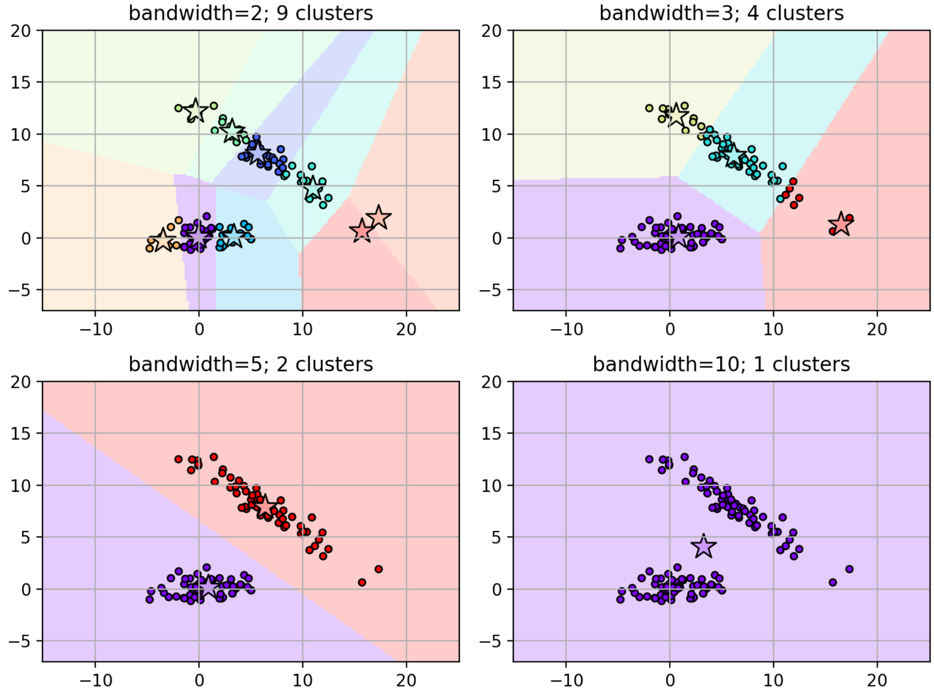
Out[24]:



- Cluster partitions: assign points based on convergence to same cluster center.

In [26]:
```
msfig
```

Out[26]:

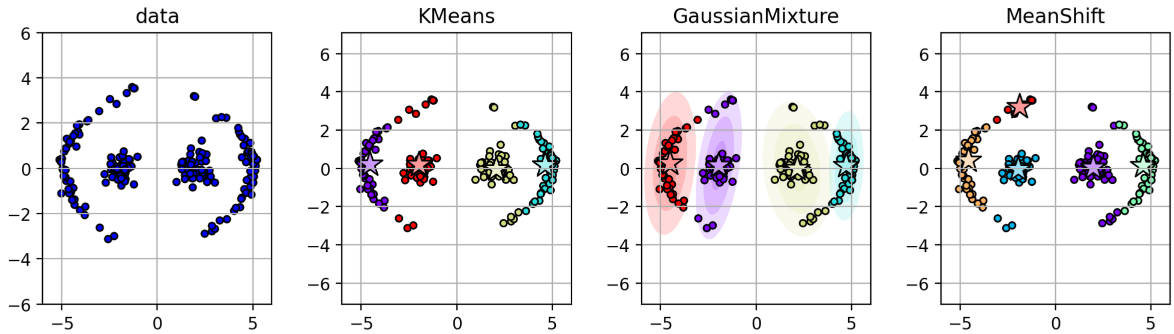


Non-compact clusters

- K-means, GMM, and Mean-Shift assume that all clusters are compact.
  - i.e., circles or ellipses
- What about clusters of other shapes?
  - e.g., clusters not defined by compact distance to a "center"

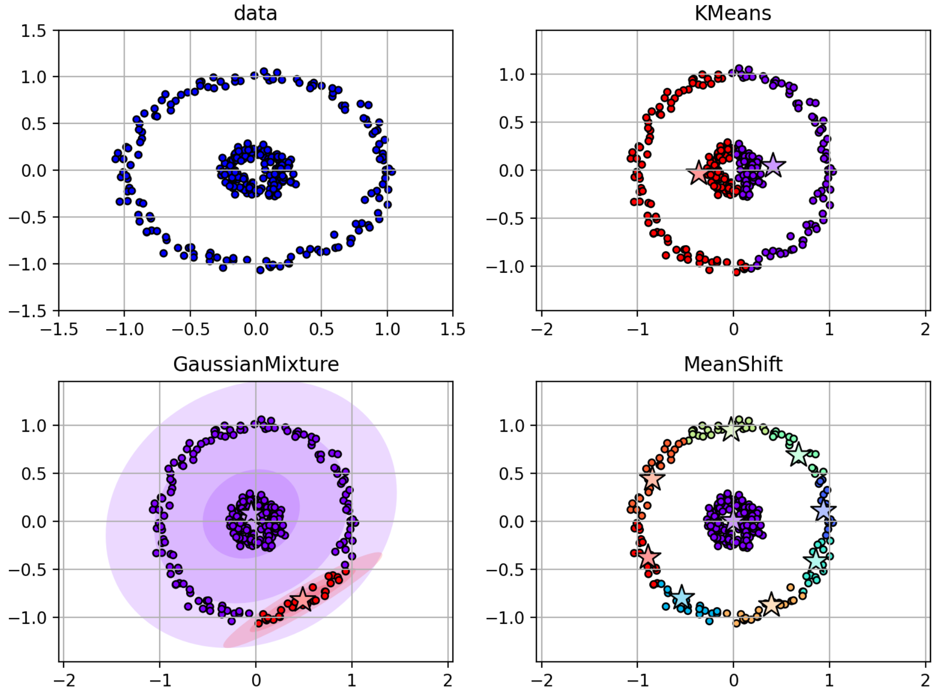
In [28]:
```
tiefig
```

Out[28]:



In [30]:
```
circfig
```

Out[30]:



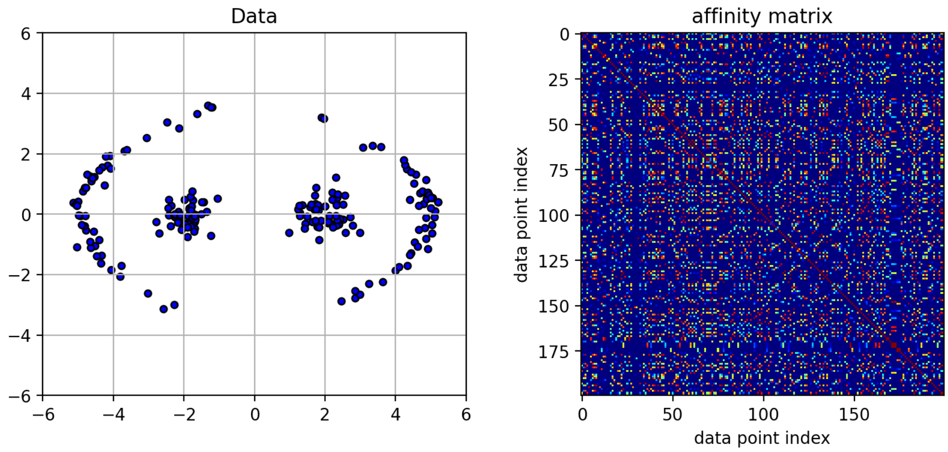
Spectral Clustering

- Estimate clusters using pair-wise affinity between points.
- Affinity (similarity) between points
  - kernel function: $k(\mathbf{x}_i, \mathbf{x}_j)$ -- RBF kernel
  - number of nearest neighbors within a radius (bandwidth)

In [32]:

```
afig
```

Out[32]:



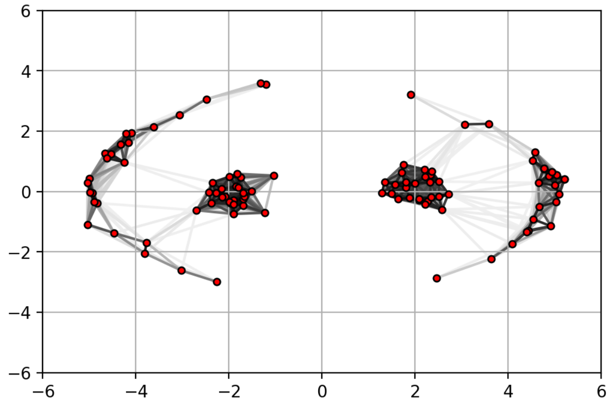
Spectral Clustering

- **Idea:** clustering with a graph formulation
  - each data point is a node in a graph
  - edge weight between two nodes is the affinity $k(\mathbf{x}_i, \mathbf{x}_j)$
    - (darker colors indicate stronger weights)

In [34]:

```
graphfig
```

Out[34]:



- **Goal:** cut the graph into clusters such that weights of cut edges is small compared to the total edge weight within each cluster.
  - find "blocks" of high affinity in the affinity matrix.

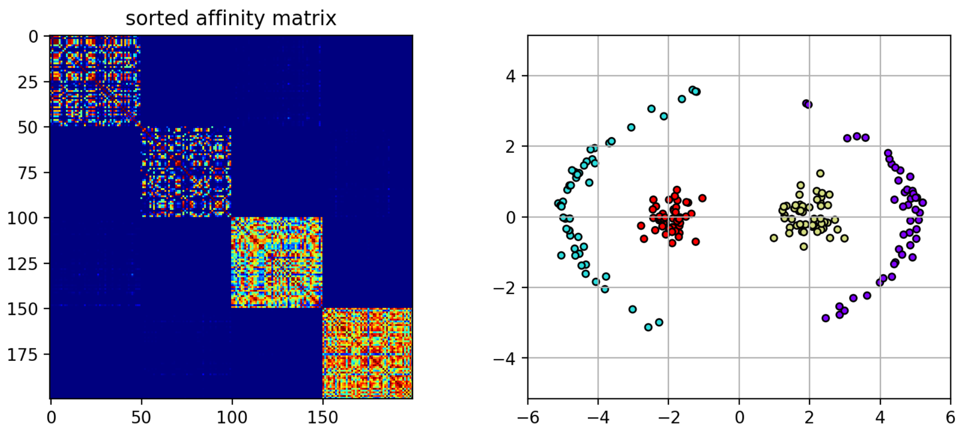
- Intuitively, consider a "mass-spring" system -- masses connected together with springs.
  - the graph nodes are masses, the edge weights are the spring stiffness.
  - if you hit the masses...
    - the masses that are tightly connected by stiff springs will move together in a low-frequency vibration mode.

- These low-frequency modes are found with the smallest non-zero eigenvectors of the graph Laplacian
- Graph Laplacian: $\mathbf{L} = \mathbf{D} - \mathbf{A}$
  - $\mathbf{A}$ is the adjacency (affinity) matrix.
  - $\mathbf{D}$ is the degree matrix.
    - diagonal matrix with entry $D_{ii} = \sum_j A_{i,j}$
- There are different ways to define the Laplacian, leading to different versions of Spectral Clustering
  - the one in sklearn is called "Normalized Cuts".

In [36]:
```
# spectral clustering
# rbf affinity
sc = cluster.SpectralClustering(n_clusters=4, affinity='rbf',
                                gamma=1.0, assign_labels='discretize', n_jobs=-1)
Y = sc.fit_predict(X)
```

In [38]:
```
scfig
```

Out[38]:

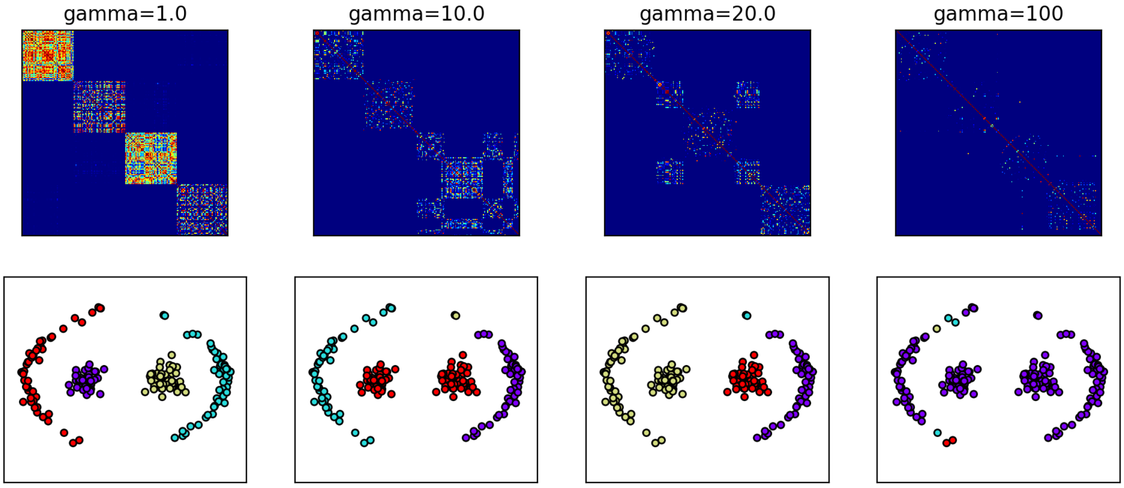


Sensitivity to gamma

- gamma controls which structures are important
  - small gamma - far away points are still considered similar

In [52]:
```
scfig2
```

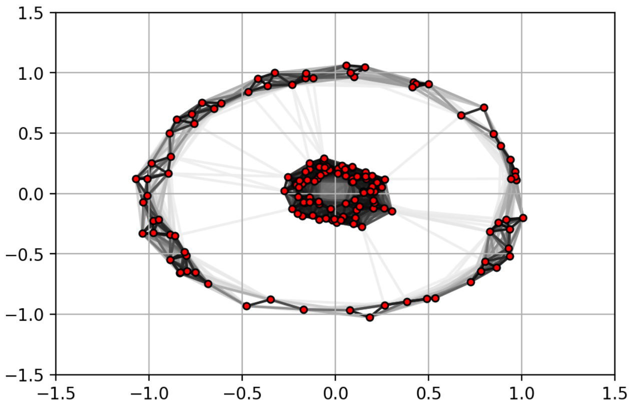
Out[52]:



Another Example

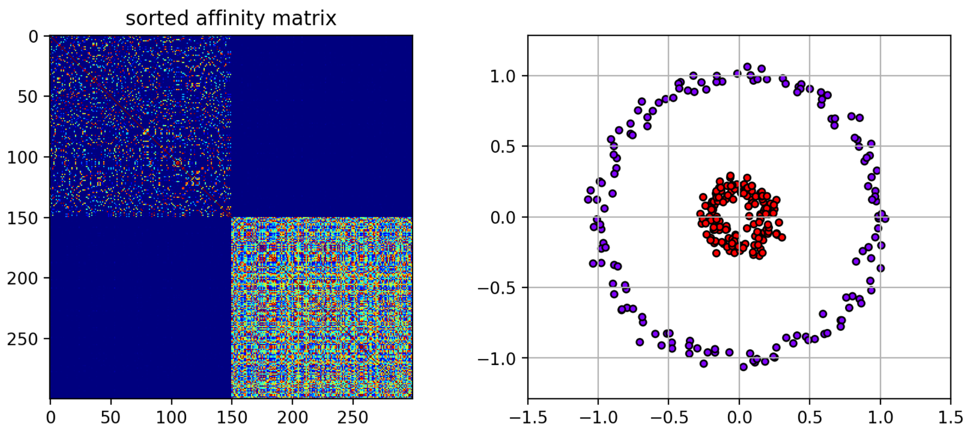
In [45]:
```
graphfig2
```

Out[45]:



In [47]:
```
scfig
```

Out[47]:



Clustering Summary

- **Goal:** given set of input vectors $\{\mathbf{x}_i\}_{i=1}^n$, with $\mathbf{x}_i \in \mathbb{R}^d$, group similar x_i together into clusters.
  - estimate a cluster center, which represents the data points in that cluster.
  - predict the cluster for a new data point.

Name	Cluster Shape	Principle	Advantages	Disadvantages
K-Means	circular	minimize distance to cluster center	- scalable (MiniBatchKMeans)	- sensitive to initialization; could get bad solutions due to local minima.

				- need to choose K.
Gaussian Mixture Model	elliptical	maximum likelihood	- elliptical cluster shapes.	- sensitive to initialization; could get bad solutions due to local minima. - need to choose K.
Bayesian GMM	elliptical	maximum marignal likelihood	- automatically selects K via concentration parameter.	- can be slow. - sensitive to initialization; could get bad solutions due to local minima.
Mean-Shift	concentrated compact	move towards local mean	- automatically selects K via bandwidth parameter.	- can be slow.
Spectral clustering	irregular shapes	graph-based	- can handle clusters of any shape, as long as connected.	- need to choose K. - cannot assign novel points to a cluster. - can be slow (kernel matrix)

Other Things

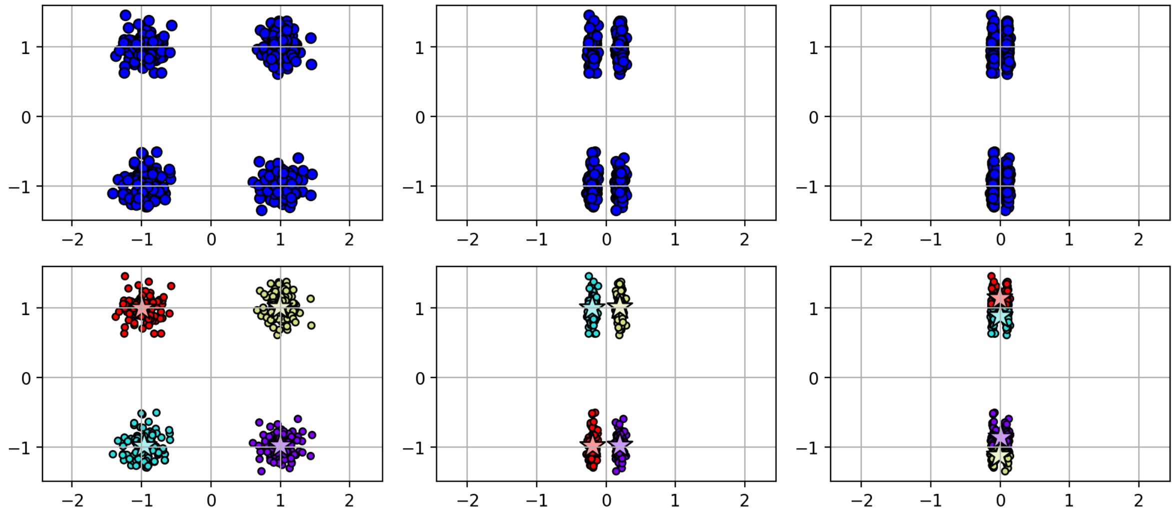
- *Feature normalization*
  - feature normalization is typically required clustering.
  - e.g., algorithms based on Euclidean distance (Kmeans, Mean-Shift, Spectral Clustering)

Example

- scaling down the x_1 feature makes its differences less important, compared to x_2.

In [49]:
```
efig
```

Out[49]:



In [ ]:
```
```