CS5489 - Machine Learning

Lecture 7a - Unsupervised Learning - Clustering

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

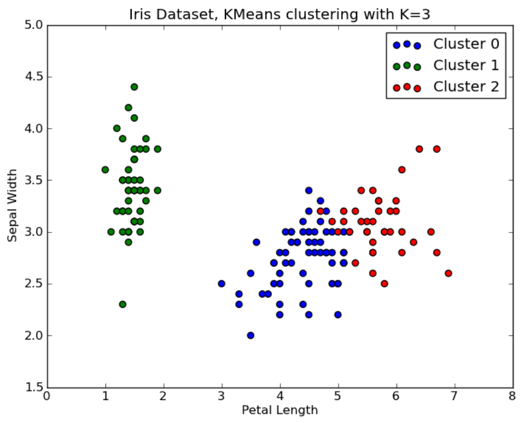
1. Unsupervised Learning
2. Parametric clustering
    A. K-means
    B. Gaussian mixture models (GMMs)
    C. Bayesian GMMs
3. Non-parametric clustering and Mean-shift
4. Spectral clustering

Unsupervised Learning

- Unsupervised learning only considers the input data $\mathbf{x}$.
    - There are no output values.
- **Goal:** Try to discover inherent properties in the data.
    - Clustering
    - Dimensionality Reduction
    - Manifold Embedding

Clustering

- Find clusters of similar items in the data.
    - Find a representative item that can represent all items in the cluster.
- **For example:** grouping iris flowers by their measurements.
    - Features are sepal width and petal length.

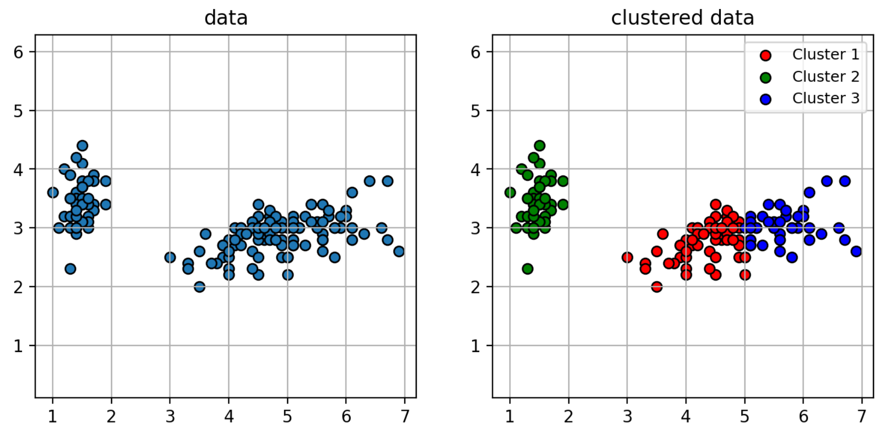


Clustering

- Data is set of vectors $\{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$
  - Each data point is a vector $\mathbf{x} \in \mathbb{R}^d$.
- **Goal:** group similar data together.
  - groups are also called clusters.
  - each data point is assigned with a cluster index ($y \in \{1, \cdots, K\}$)
    - K is the number of clusters

In [3]:
```
clusterfig
```

Out[3]:

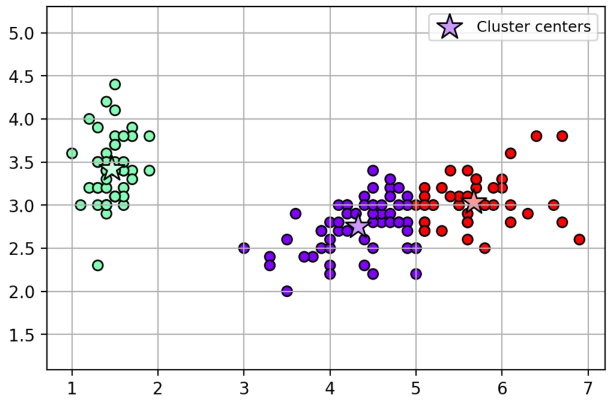


K-Means Clustering

- **Idea:**
  - there are K clusters.
  - each cluster is represented by a *cluster center*.
    - $\mathbf{c}_j \in \mathbb{R}^d, j \in \{1, \cdots, K\}$
  - assign each data point to the closest cluster center.
    - assignment variable $z_i \in \{1, \cdots, K\}$ indexes the cluster center of $\mathbf{x}_i$.

In [5]:
```
kmfig
```

Out[5]:



K-means Clustering Objective

- **Objective:** minimize the total sum-squared difference between points and their centers

$$\min_{\mathbf{c}_1,\cdots,\mathbf{c}_K,z_1,\cdots,z_n} \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{c}_{z_i}||^2$$

K-means Clustering Objective

- *How to pick the cluster centers?*

    - Assume the assignments z_i are known.
    - Pick the cluster centers that minimize the squared distance to all its cluster members.

$$\min_{\mathbf{c}_1,\cdots,\mathbf{c}_K} \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{c}_{z_i}||^2$$

- Solution:
    - if the assignments $\{z_i\}$ are known...
        - let C_j be the set of points assigned to cluster j
            - $C_j = \{\mathbf{x}_i | z_i = j\}$
        - For each cluster, we have
            - $\mathbf{c}_j = \text{argmin} \sum_{i \in C_j} ||\mathbf{x}_i - \mathbf{c}_j||^2$
        - Take the derivative and set to 0:

$$\frac{d}{d\mathbf{c}_j} \sum_{i \in C_j} ||\mathbf{x}_i - \mathbf{c}_j||^2 = \sum_{i \in C_j} 2(\mathbf{x}_i - \mathbf{c}_j)(-1) = 0$$
$$\Rightarrow \sum_{i \in C_j} \mathbf{x}_i - |C_j|\mathbf{c}_j = 0$$

        - Cluster center is the mean of the points in the cluster
            - $\mathbf{c}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i$

K-means Clustering Objective

- *How to pick the assignments?*
    - Assume the clusters $\{\mathbf{c}_1, \cdots, \mathbf{c}_K\}$ are *known*.
    - Pick the assignments that minimize the squared distance to the clusters.

$$\min_{z_1,\cdots,z_n} \sum_{i=1}^{n} ||\mathbf{x}_i - \mathbf{c}_{z_i}||^2$$

- For each data point,
    - $z_i = \text{argmin}_{j \in \{1,\cdots K\}} ||\mathbf{x}_i - \mathbf{c}_j||^2$
    - i.e., assign point $\mathbf{x}_i$ to its closest cluster.

Chicken and Egg Problem

- Cluster assignment of each point depends on the cluster centers.
- Location of cluster center depends on which points are assigned to it.

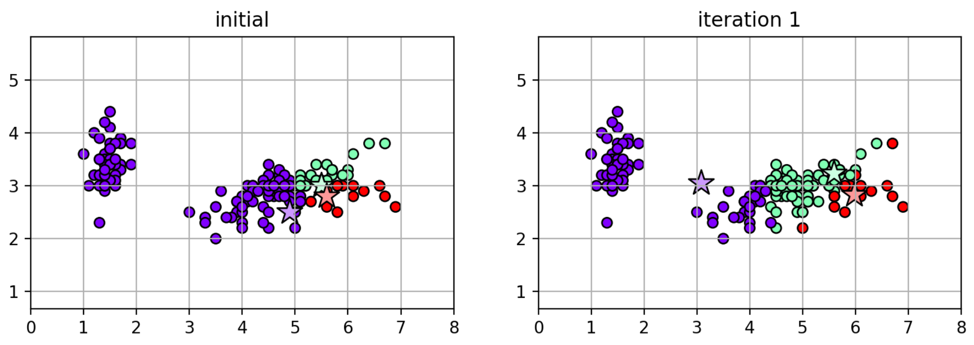
- **Solution**: just iterate between the two steps.
    - Note: in each step we are holding one set of variables fixed, while minimizing over the others.
    - Thus we are always minimizing the original objective!

K-means Algorithm

- Pick initial cluster centers
- Repeat:
    - 1. calculate assignment z_i for each point $\mathbf{x}_i$: closest cluster center using Euclidean distance.
    - 2. calculate cluster center $\mathbf{c}_j$ as average of points assigned to cluster j.
- This procedure will converge eventually.

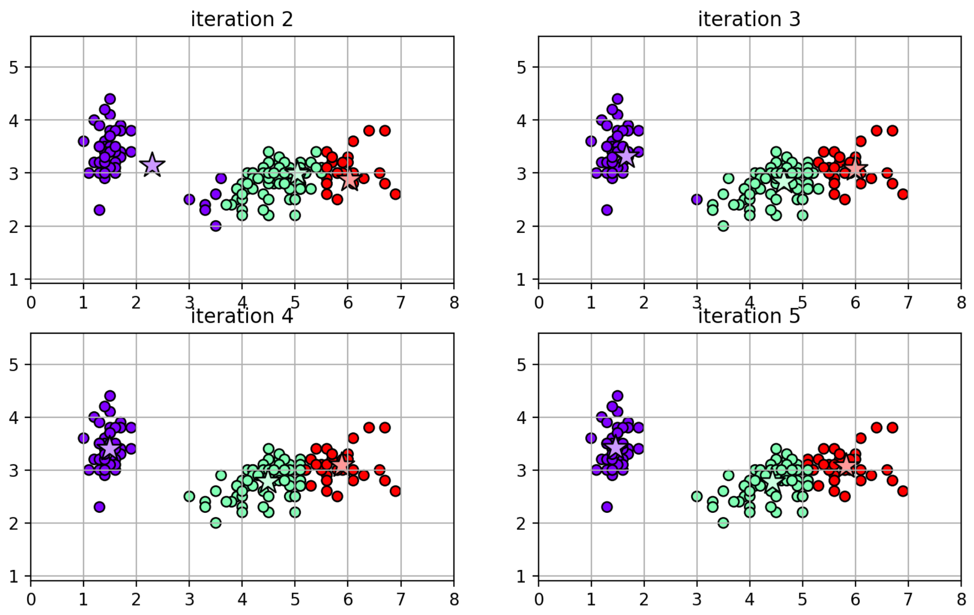
In [8]:
```
kmifig1
```

Out[8]:



In [9]:
```
kmifig2
```

Out[9]:

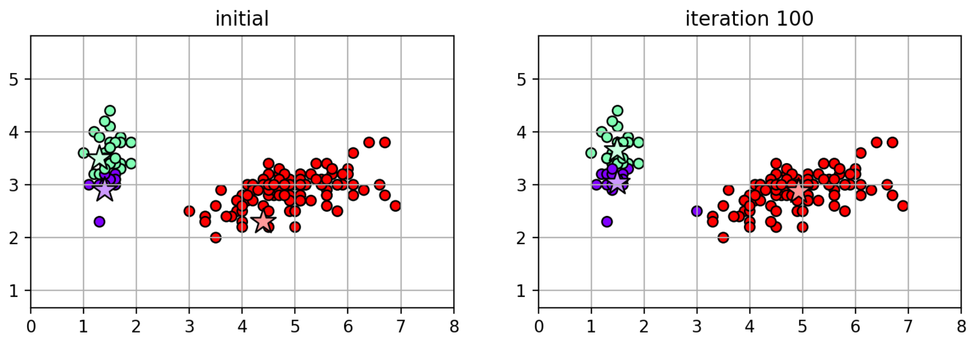


Important Note

- The final result depends on the initial cluster centers!
  - Some bad initializations will yield poor clustering results!
  - (Technically, there are multiple local minimums in the objective function)

In [11]:
```
kmbadfig
```

Out[11]:

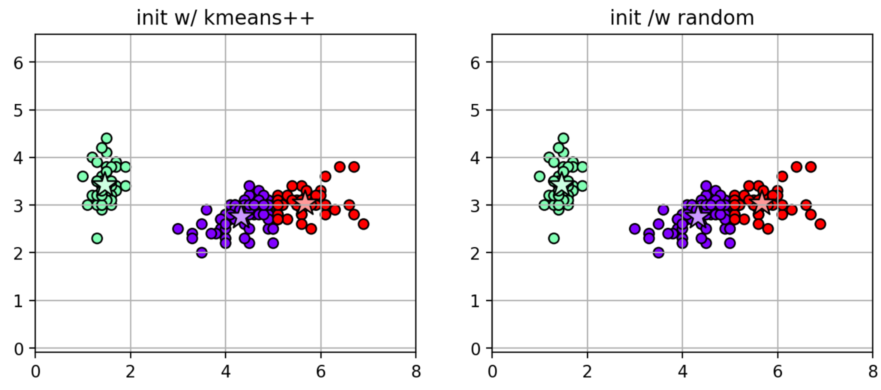


- **Solution:**
  - Try several times using different initializations.
  - Pick the answer with lowest objective score.
- In scikit-learn,
  - 1. use a smart initialization method called "k-means++", which speeds up convergence. (default)
  - 2. use multiple random initializations.

In [12]:
```
# K-Means with 3 clusters (w/ Kmeans++ init)
km = cluster.KMeans(n_clusters=3, random_state=4487)
Yp = km.fit_predict(X)    # cluster data, and return labels

cc = km.cluster_centers_    # the cluster centers
cl = km.labels_             # labels also stored here

# K-means w/ random initialization (automatically does 10 trials)
kmr = cluster.KMeans(n_clusters=3, random_state=4487, init='random')
Ypr = kmr.fit_predict(X)    # cluster data, and return labels
```

In [14]:
```
kmfig
```

Out[14]:

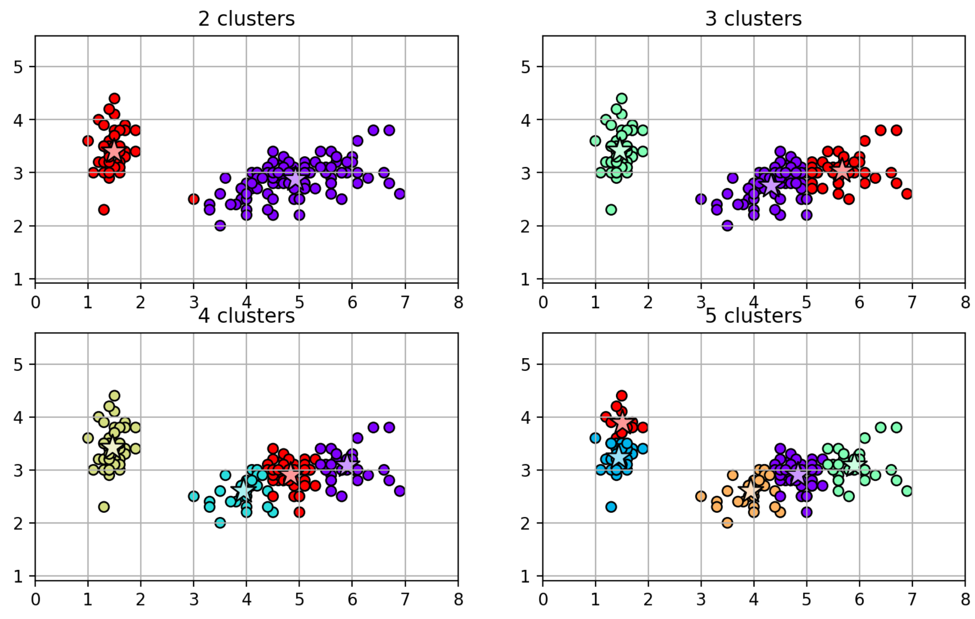


For different K

- We need to choose the appropriate K

In [16]:
```
ksfig
```

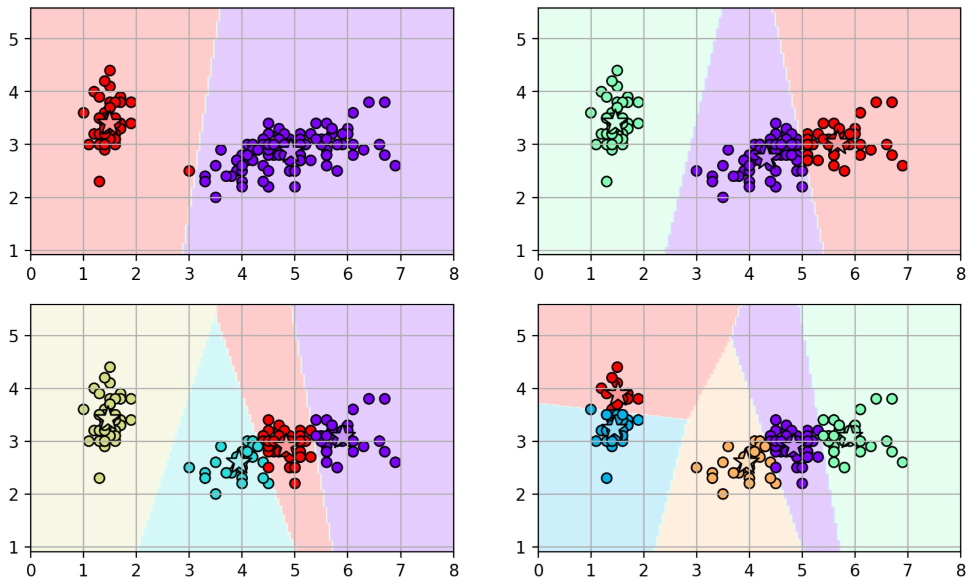
Out[16]:



- K-means partitions the input space into non-overlapping regions belonging to each cluster
  - assign each location in the space to the closest center.

In [18]:
```
krfig
```

Out[18]:



Bag-of-X Representation

- K-means can partition the input space into regions.
- Create a new quanitized representation for a set of samples.
  - 1. learn the partition space of the samples using K-means (i.e, the vocabulary)
  - 2. assign each sample to its closest center (i.e., the word)
  - 3. count the number of assignments for each center and form a histogram (i.e., the bag-of-words)
- Called a "bag-of-X" representation.
  - "X" is whatever modality you are using.

Example: Bag-of-visual-words

- Procedure:
  - 1. extract small patches from images
  - 2. learn the visual words using K-means
  - 3. assign patches to visual words, and formm a histogram for each image.
- Use the bag-of-words model as the new feature vector.

In [19]:

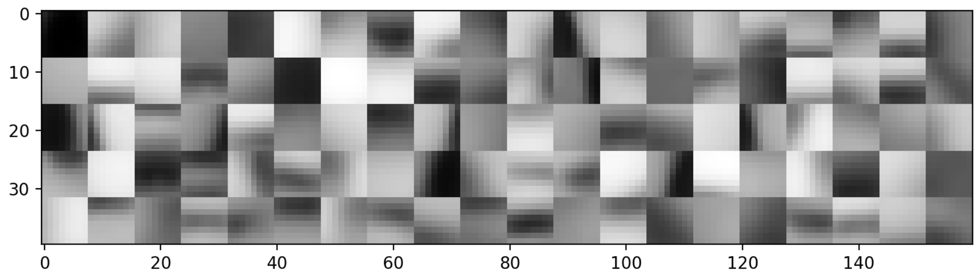
```
# load images
oli = datasets.fetch_olivetti_faces(data_home="./")
img = oli.images
print(img.shape)

# extract 8x8 window patches with step size of 4x4
patches = skimage.util.view_as_windows(img, (1,8,8), step=(1,4,4))
print(patches.shape)

# reshape patches into 64-dim vectors
patchesall = patches.reshape((prod(patches.shape[0:3]),8*8))
print(patchesall.shape)

# run k-means (mini-batch version can handle large datasets)
# fit k-means, and predict the cluster index for each sample
K = 100
bows = cluster.MiniBatchKMeans(n_clusters=K, random_state=5489, n_init=10)
wordsall = bows.fit_predict(patchesall)
```

```
(400, 64, 64)
(400, 15, 15, 1, 8, 8)
(90000, 64)
```

In [21]:

```
# get the visual words, and show them
visualwords = bows.cluster_centers_.reshape((K,8,8))
```

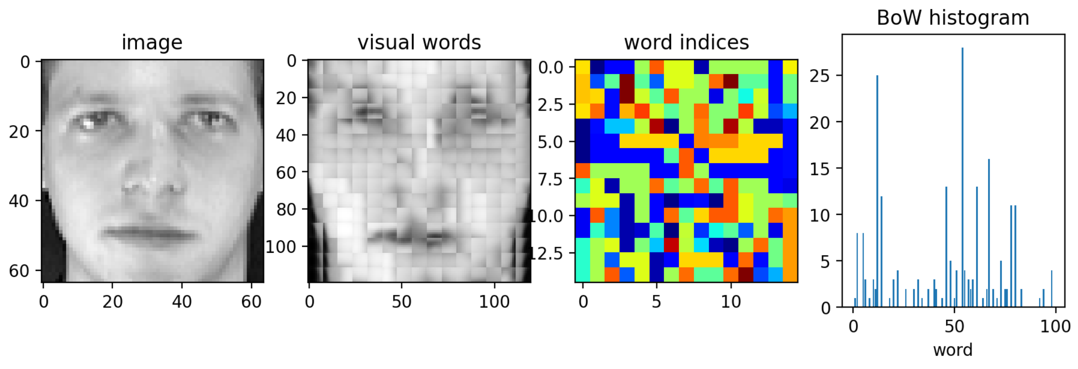
```
plt.figure(figsize=(10,4))
plt.imshow(image_montage(visualwords, maxw=20), cmap='gray', interpolation='nearest');
```

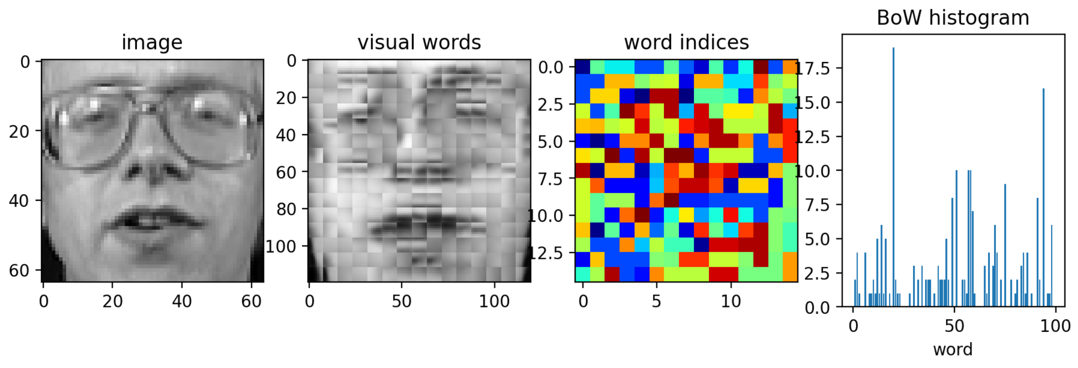


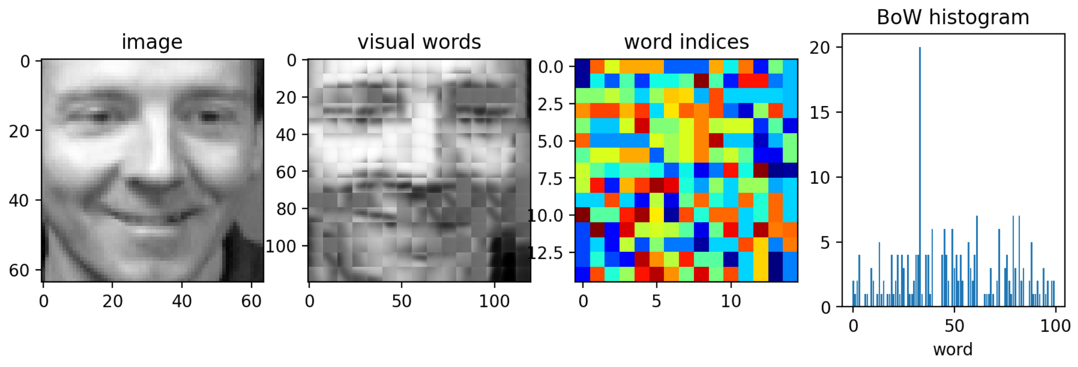
In [22]:
```
# reshape the predicted words into small images
print(wordsall.shape)
patches_words = wordsall.reshape(patches.shape[0:3])
print(patches_words.shape)

# build the BoW histogram for each image
bowhist = [bincount(wds.ravel(), minlength=K) for wds in patches_words]
```

```
(90000,)
(400, 15, 15)
```

In [24]:
```
# view some reconstructions, words, and histograms
for t in [0,10,20]:
    plt.figure(figsize=(11,3))
    show_bow_recon(visualwords, img[t], patches_words[t], bowhist[t])
    plt.show()
```





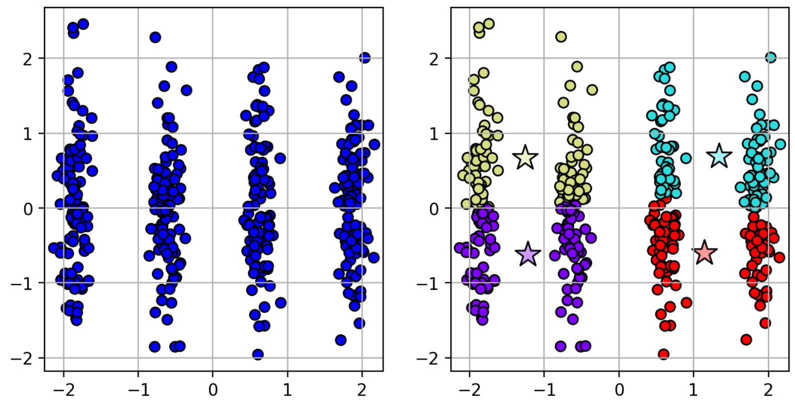


Circular clusters

- One problem with K-means is that it assumes that each cluster has a circular shape.
  - based on Euclidean distance to each center
  - Kmeans cannot handle skewed (elliptical) clusters.

```
In [26]: efig
```

Out[26]:



Gaussian mixture model (GMM)

- A multivariate Gaussian can model a cluster with an elliptical shape.
  - the ellipse shape is controlled by the covariance matrix of the Gaussian
  - the location of the cluster is controlled by the mean.
- Gaussian mixture model is a weighted sum of Gaussians

$$p(\mathbf{x}) = \sum_{j=1}^{K} \pi_j N(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

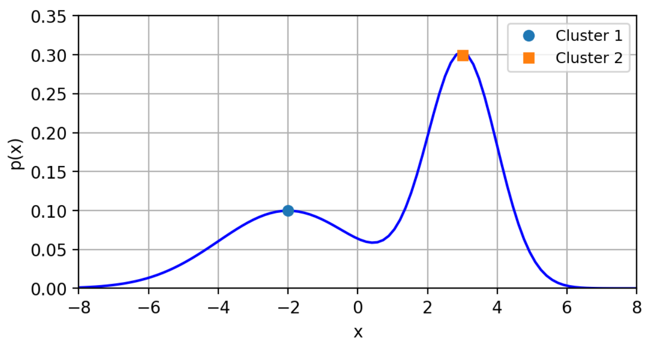
- Each Gaussian represents one elliptical cluster
  - $\boldsymbol{\mu}_j$ is the mean of the j-th Gaussian. (the location)
  - $\boldsymbol{\Sigma}_j$ is the covariance matrix of the j-th Gaussian. (the ellipse shape)
  - π_j is the prior weight of the j-th Gaussian. (how likely is this cluster)

1-D example of GMM

- $p(x) = 0.25 * N(x|-2, 4) + 0.75 * N(x|3, 1)$
- each Gaussian is a "mountain"

```
In [28]: eggmm
```

Out[28]:

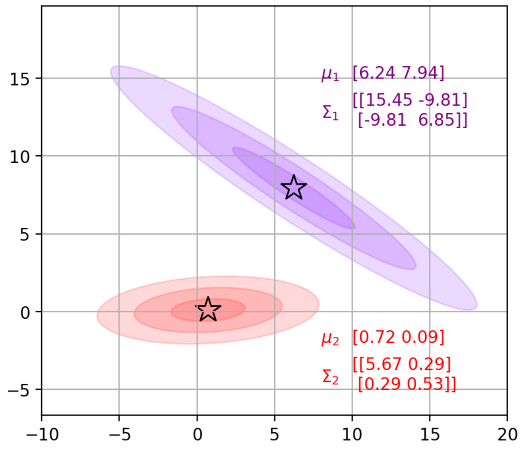


2D example of GMM

- Each Gaussian defines a "mountain"
  - contours are ellipses

In [30]:
```
gmm2fig
```

Out[30]:



Clustering with GMMs

- Using the data, learn a GMM using maximum likelihood estimation:

$$\max_{\pi,\boldsymbol{\mu},\boldsymbol{\Sigma}} \sum_{i=1}^{N} \log \sum_{j=1}^{K} \pi_j N(\mathbf{x}_i | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

  - The learned $\{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}$ are the cluster center and ellipse shape.
  - The learned π_j is the cluster population (percentage of points).
- This is a difficult to optimize because the "sum" is inside the "log".

Expectation Maximization (EM) Algorithm

- An algorithm for finding the MLE solution when there are hidden (unseen) variables z.
  - Goal: $\hat{\theta} = \text{argmax}_\theta\, p_\theta(\mathbf{x})$
    - where $p_\theta(\mathbf{x}) = \sum_z p(\mathbf{x}|z)p(z)$
    - θ are the parameters.
- Solution:
  - iterate between estimating the hidden variables z and maximizing w.r.t the parameters θ.

- For GMMs:
  - z is the assignment of $\mathbf{x}$ to one of the Gaussian components.
  - prior probability: $p(z = j) = \pi_j$
  - Gaussian component: $p(\mathbf{x}|z = j) = \mathcal{N}(\mathbf{x}|\mu_j, \Sigma_j)$
  - likelihood: $p(\mathbf{x}) = \sum_z p(\mathbf{x}|z)p(z) = \sum_j \pi_j \mathcal{N}(\mathbf{x}|\mu_j, \Sigma_j)$

EM Algorithm

- 1. Select an initial model $\hat{\theta}$
- 2. **E-step**: estimate the hidden variables as their expected value.
  - $\mathcal{Q}(\theta) = E_{z|\mathbf{x},\hat{\theta}}[\log p_\theta(\mathbf{x}, z)]$
  - the expectation uses current model parameters $\hat{\theta}$.
- 3. **M-step**: maximize w.r.t θ
  - $\hat{\theta} = \text{argmax}_\theta\, \mathcal{Q}(\theta)$

- 4. repeat E- and M-steps until convergence.
  - It's gauranteed to converge.

Joint likelihood for GMMs

- Joint likelihood of $(\mathbf{x}_i, z_i)$

$$p(\mathbf{x}_i, z_i) = p(\mathbf{x}_i|z_i)p(z_i) = \pi_{z_i}\mathcal{N}(\mathbf{x}_i|\mu_{z_i}, \Sigma_{z_i})$$

- annoying because z_i is indexing the parameters.

Indicator Variable Trick

- Define z_{ij} as the indicator variable that $z_i = j$
  - $z_{ij} = \begin{cases} 1, & z_i = j \\ 0, & \text{otherwise} \end{cases}$
- Joint likelihood of $(\mathbf{x}_i, z_i)$
  - z_{ij} selects the correct Gaussian component.

$$p(\mathbf{x}_i, z_i) = \prod_{j=1}^{K} (p(\mathbf{x}_i|z_i = j)p(z_i = j))^{z_{ij}} = \prod_{j=1}^{K} \left(\pi_j\mathcal{N}(\mathbf{x}_i|\mu_j, \Sigma_j)\right)^{z_{ij}}$$

$$\Rightarrow \log p(\mathbf{x}_i, z_i) = \sum_{j=1}^{K} z_{ij} \log(\pi_j\mathcal{N}(\mathbf{x}_i|\mu_j, \Sigma_j))$$

EM algorithm for GMMs

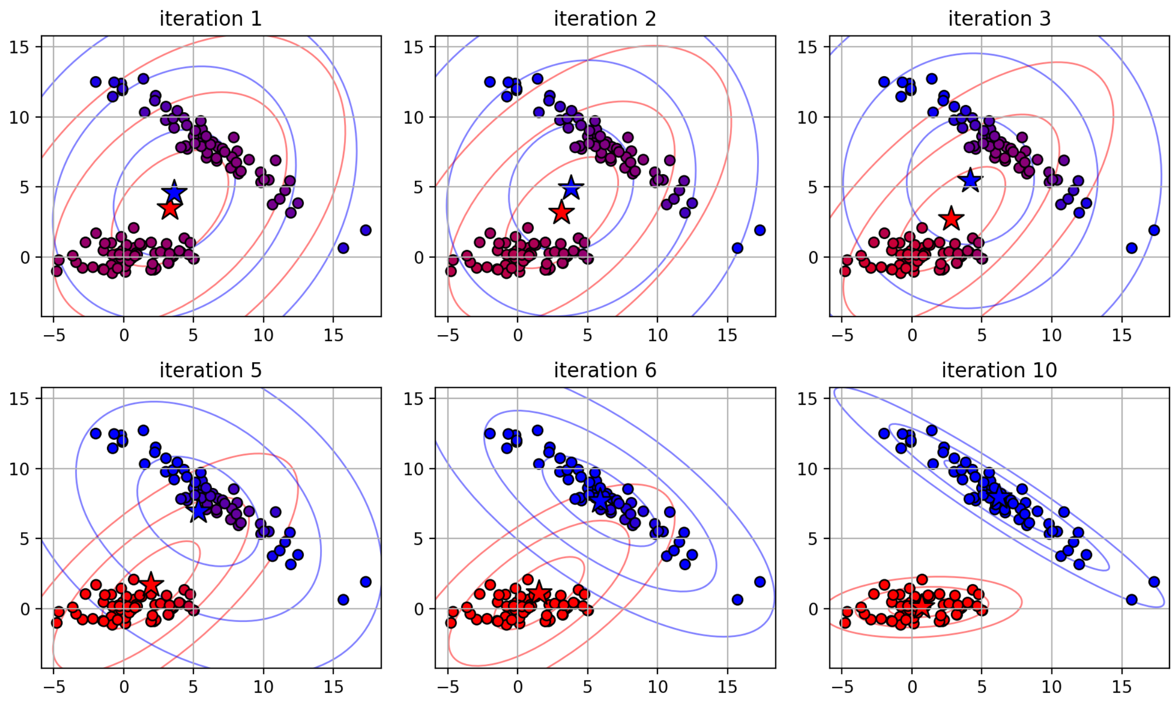
- **E-step:** Calculate cluster membership
  - assignment of point i to cluster j
    - $\hat{z}_{ij} = p(z_i = j|\mathbf{x}_i) = \frac{\pi_j\mathcal{N}(\mathbf{x}_i|\mu_j,\Sigma_j)}{\sum_k \pi_k\mathcal{N}(\mathbf{x}_i|\mu_k,\Sigma_k)}$
  - uses "soft" assignment - a data point can have a fractional assignment to different clusters.

- **M-step:** Update each Gaussian cluster (mean, covariance, and weight)
  - uses "soft" weighting
    - "soft" count of points in cluster j: $N_j = \sum_{i=1}^{N} \hat{z}_{ij}$
    - weight: $\pi_j = N_j/N$
    - mean: $\boldsymbol{\mu}_j = \frac{1}{N_j}\sum_{i=1}^{N} \hat{z}_{ij}\mathbf{x}_i$
    - variance: $\Sigma_j = \frac{1}{N_j}\sum_{i=1}^{N} \hat{z}_{ij}(\mathbf{x}_i - \boldsymbol{\mu}_j)^2$
- Similar to K-means, except uses "soft" assignments, rather than "hard" assignments.

- EM for GMM Example
  - red points - assigned to cluster 1
  - blue points - assigned to cluster 2
  - purple points - fractionally assigned between clusters 1 and 2

```
In [32]: efig
```

Out[32]:



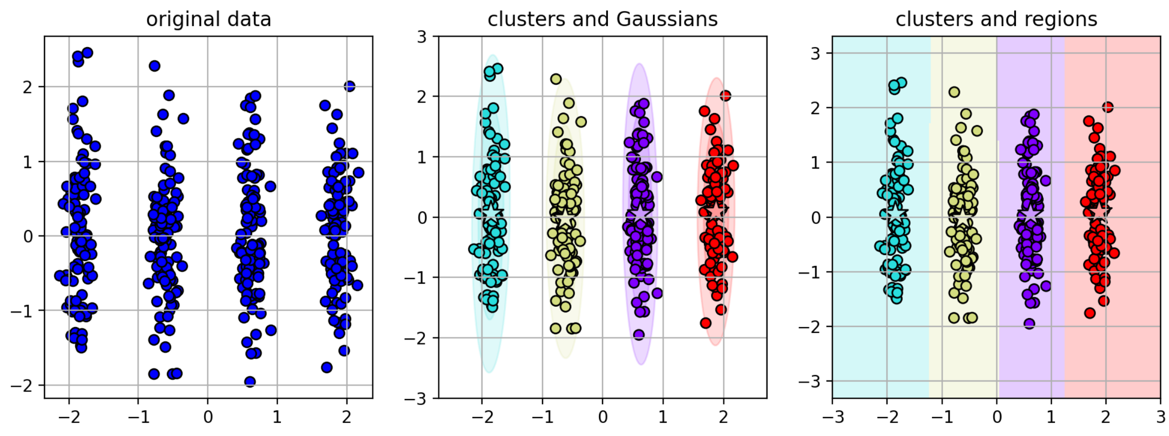
In [33]:

```
# fit a GMM
gmm = mixture.GaussianMixture(n_components=4, random_state=4487, n_init=10)

gmm.fit(X)
Y = gmm.predict(X)

cc = gmm.means_       # the cluster centers
```

In [35]:

```
efig
```

Out[35]:



Covariance matrix

- The covariance matrix is a $d \times d$ matrix.

$$\Sigma_j = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- For high-dimensional data, it can be very large.
  - requires a lot of data to learn effectively.

- Solution:
  - use *diagonal* covariance matrices (d parameters):

$$\Sigma_j = \mathrm{diag}(\mathbf{a}) = \begin{bmatrix} a_1 & 0 & 0 \\ 0 & a_2 & 0 \\ 0 & 0 & a_3 \end{bmatrix}$$

- Axes of ellipses will be aligned with the axes.
- use *spherical* covariance matrices (1 parameter)

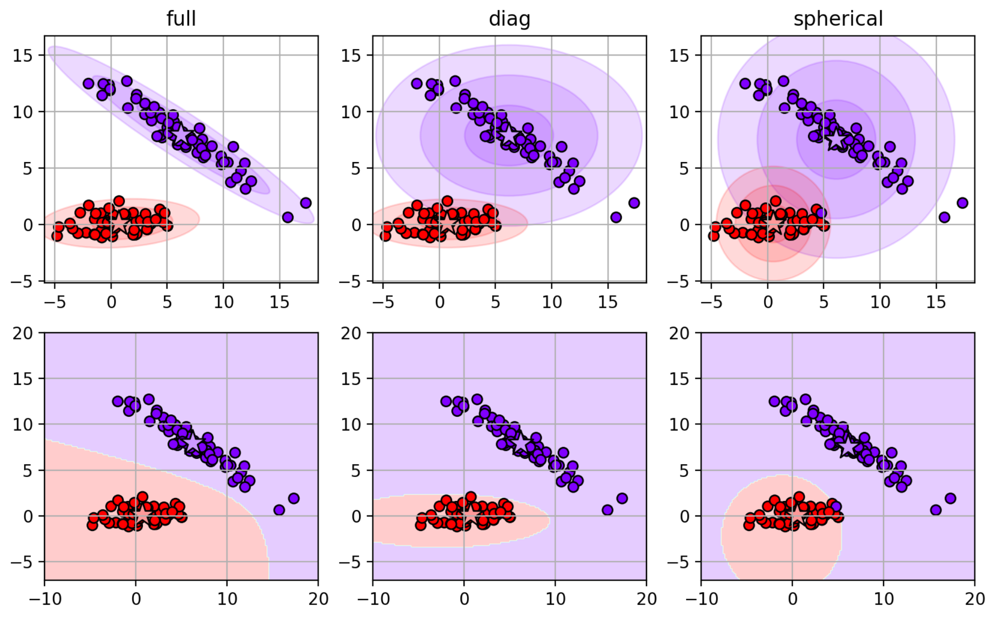
$$\Sigma_j = a\mathbf{I} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{bmatrix}$$

In [37]:
```
# full covariance (d*d parameters)
gmmf = mixture.GaussianMixture(n_components=2,
                               covariance_type='full',
                               random_state=4487, n_init=10)
gmmf.fit(X)

# diagonal convariance (d parameters)
gmmd = mixture.GaussianMixture(n_components=2,
                               covariance_type='diag',
                               random_state=4487, n_init=10)
gmmd.fit(X)

# spherical covariance (1 parameter)
gmms = mixture.GaussianMixture(n_components=2,
                               covariance_type='spherical',
                               random_state=4487, n_init=10)
gmms.fit(X)
```

Out[37]:
```
▾                    GaussianMixture
GaussianMixture(covariance_type='spherical', n_components=2, n_init=10,
                random_state=4487)
```

In [39]:
```
efig
```

Out[39]:



How to select K?

- Clustering results depends on the number of clusters used.
- We don't typically know this information beforehand.

Bayesian GMM

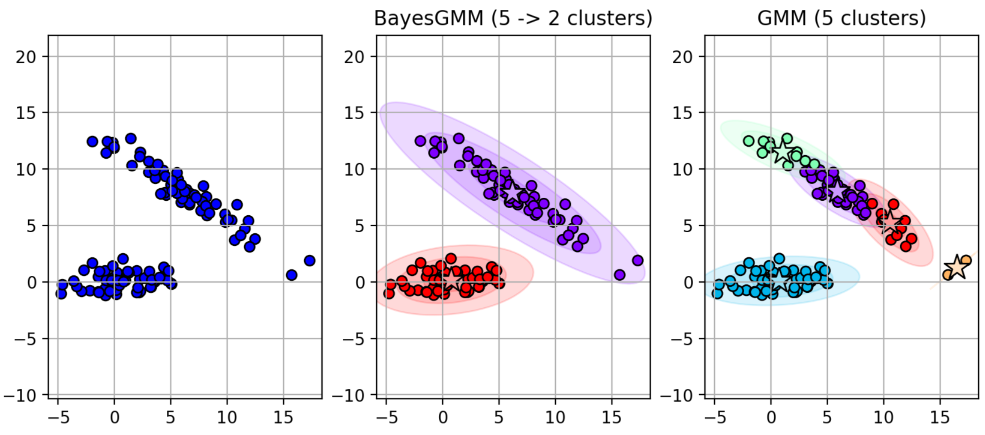
- A Bayesian version of GMM that automatically selects the value of K
  - use a Dirichlet distribution to model the prior probabilities, $p(\pi)$.
- *concentration* parameter α controls the range of π values that are preferred
  - higher values encourage more clusters (π_j are uniform)
  - lower values encourage less clusters (more $\pi_j = 0$)
  - Mathematically, α adds to the number of samples in the cluster, $\pi_j = \frac{N_j+\alpha}{N+K\alpha}$
    - similar to Laplace smoothing.
- (more details in the lecture on graphical models)

In [40]:
```
# alpha = concentration parameter
# n_components = the max number of components to consider
dpgmm = mixture.BayesianGaussianMixture(covariance_type='full',
                   weight_concentration_prior=1,
                   n_components=5, max_iter=100, random_state=4487)
dpgmm.fit(X)
Y = dpgmm.predict(X)
cl = unique(Y)          # find active clusters
newK = len(cl)          # number of clusters
cc = dpgmm.means_[cl]   # get means
```

- Bayesian GMM automatically selects 2 components from 5
  - for comparison, GMM with 5 clusters looks messy

In [42]:
```
dfig
```

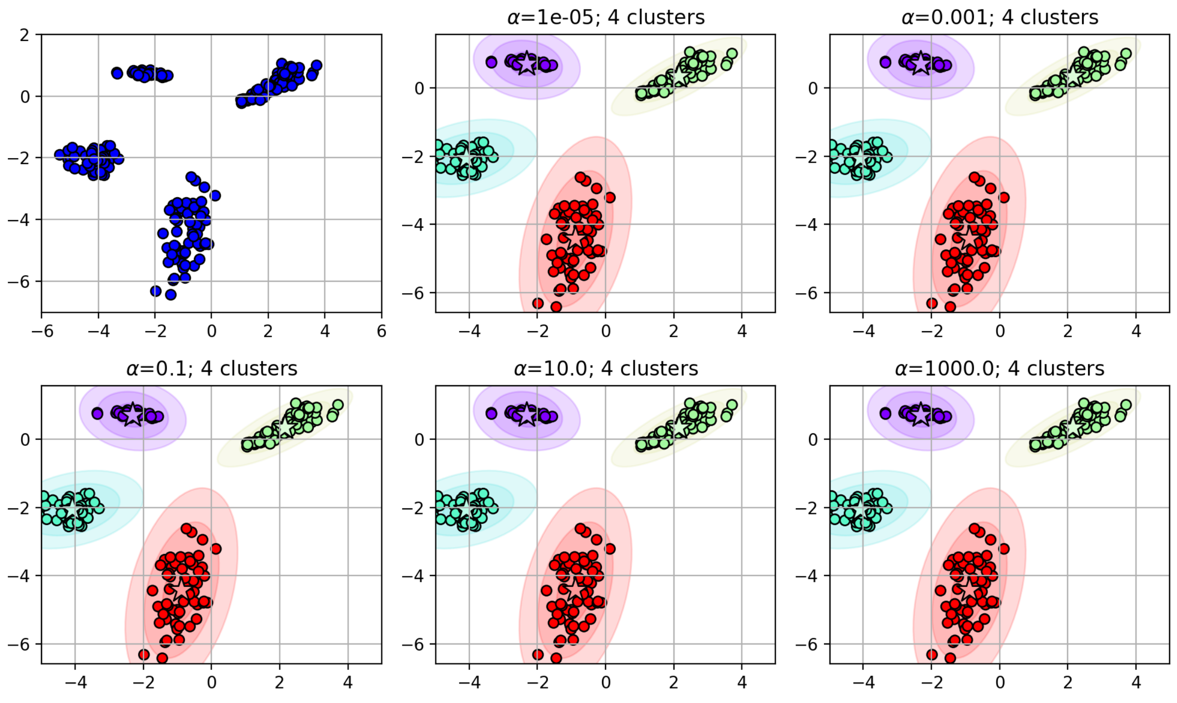
Out[42]:



- For different concentration parameter α
  - larger α may yield more clusters

In [45]:
```
dpfig
```

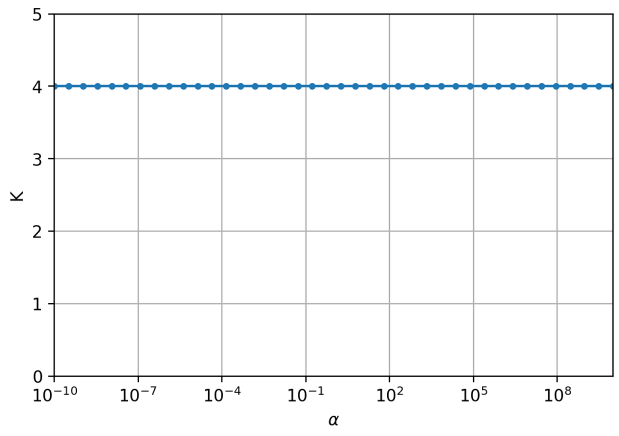
Out[45]:



- Choice of α is not that critical
  - same number of clusters for large ranges of α

In [47]:
```
dpgmmafig
```

Out[47]:



In [ ]: