

Data Science and Advanced Programming — Lecture 11

Unsupervised Machine Learning

Simon Scheidegger
Department of Economics, University of Lausanne, Switzerland

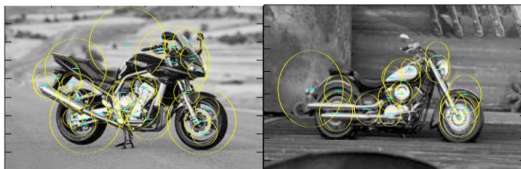
November 24th, 2025 | 12:30 - 16:00 | Internef 263

Today's Roadmap

1. k-Means
2. Evaluating Clusterings ([read at home - here for completeness](#))
3. Gaussian Mixture Models
4. Principal Component Analysis (PCA)
5. Expectation Maximization ([read at home - here for completeness](#))
6. Hierarchical Clustering
7. Density-based Clustering (cont'd)

Learning Parameters of Probability Distributions

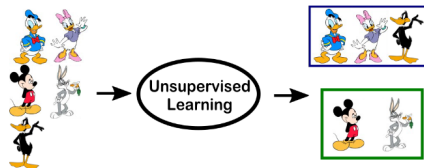
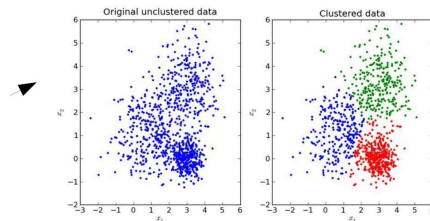
- ▶ In many settings **not all variables are observed (labeled)** in the training data $\mathbf{x}_i = (\mathbf{x}_i, \mathbf{h}_i)$
- ▶ e.g. Speech recognition: have speech signals, but not phoneme labels.
- ▶ e.g. **object recognition**: have object labels (car, bike), but **not part labels** (wheel, door, seat).
- ▶ Unobserved variables are called **LATENT VARIABLES**.



Recall — Unsupervised Learning

Learning “what normally happens”.

- ▶ No output.
- ▶ Clustering: Grouping similar instances.
- ▶ Example applications:
 - ▶ Customer segmentation.
 - ▶ Image compression: Color quantization.
 - ▶ Bioinformatics: Learning motifs.

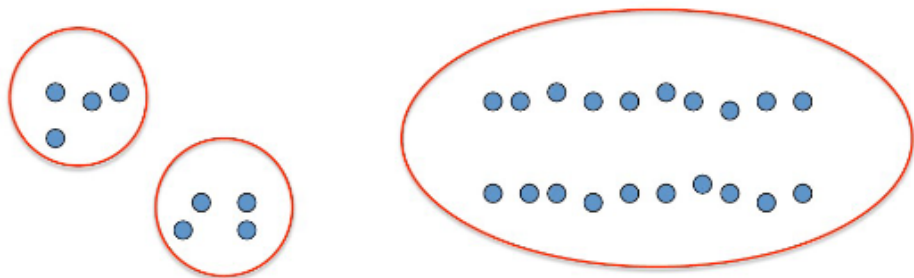


Motivation — Clustering

- ▶ Clustering, as a kind of **unsupervised learning**, aims at grouping data points into clusters.
- ▶ **Intuition:** Data points within
 - ▶ the same cluster should be close to each other
 - ▶ different clusters should be far apart from each other
- ▶ **Applications:**
 - ▶ segmentation of customers (e.g., for marketing campaigns)
 - ▶ organization/exploration of data (e.g., search results)
 - ▶ detection of outliers data points

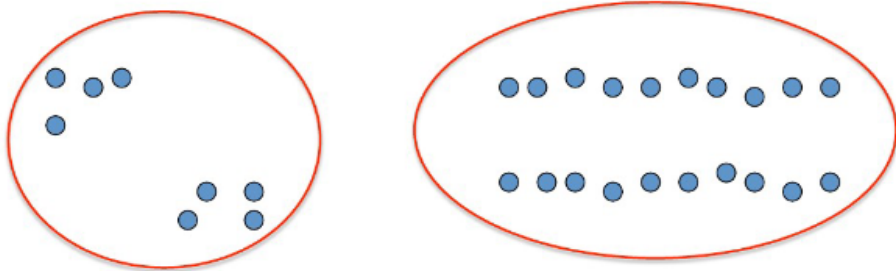
Clustering: Basic idea

- ▶ Basic idea: group together similar instances
- ▶ Example: 2D point patterns



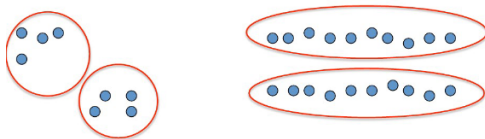
Clustering: Basic idea

- ▶ Basic idea: group together similar instances
- ▶ Example: 2D point patterns



Clustering: Basic idea

- ▶ Basic idea: group together similar instances
- ▶ Example: 2D point patterns

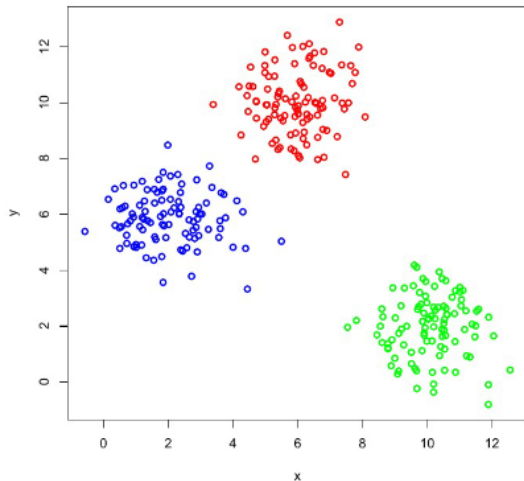
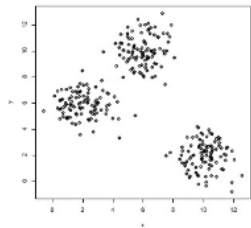


- ▶ **What could similar mean?**
- ▶ One option: small Euclidean distance (squared)

$$\text{dist}(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_2^2$$

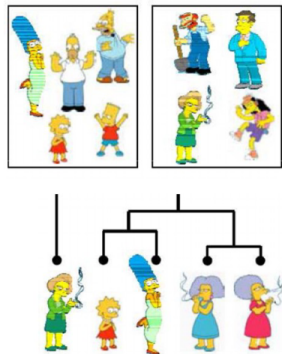
- ▶ **Clustering results are crucially dependent on the measure of similarity (or distance) between “points” to be clustered**

Clustering: Basic idea in color



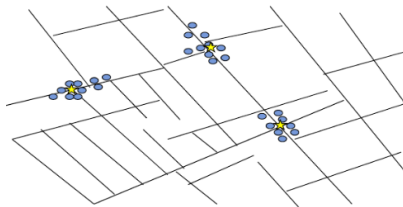
Clustering Algorithms

- ▶ Partition algorithms (Flat)
 - ▶ K-Means
 - ▶ Mixture of Gaussians
 - ▶ ...
- ▶ Hierarchical algorithms
 - ▶ Bottom-up - agglomerative
 - ▶ Top down - divisive



First (?) Application of Clustering

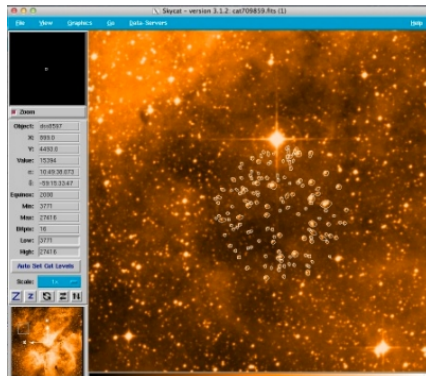
- ▶ John Snow, a London physician plotted the location of **cholera deaths** on a map during an outbreak in the 1850s.
- ▶ The locations indicated that cases were **clustered around certain intersections where there were polluted wells** — thus exposing both the problem and the solution.



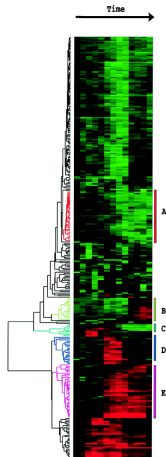
From: Nina Mishra HP Labs

Clustering Example: Astronomy

SkyCat (<http://www.eso.org/sci/observing/tools/skycat.html>): Clustered 2×10^9 sky objects into stars, galaxies, quasars, etc. based on radiation emitted in different spectrum bands



Another Clustering Example: Genetics

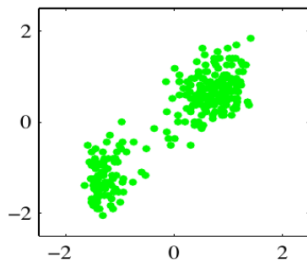


Eisen et al, PNAS 1998

1. k-Means – Unsupervised ML

Bishop, Chapter 9

- ▶ We will start with an unsupervised learning (clustering) problem:
- ▶ Given a dataset $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ each $\mathbf{x}_i \in \mathbb{R}^D$ partition the dataset into **K clusters** (e.g. healthy / sick patients).
- ▶ Intuitively, a cluster is a group of points, which are close together and far from others.



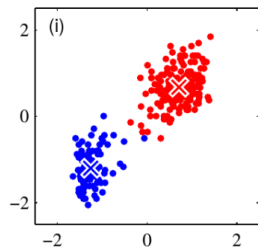
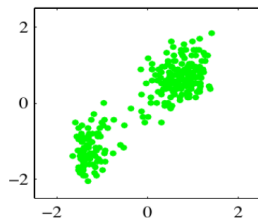
Distortion Measure

- Formally, introduce prototypes (or cluster centers) $\mu_k \in \mathbb{R}^D$
- Use binary r_{nk} , 1 if point n is in cluster k , 0 otherwise (1-of- K coding scheme again)
- Find $\{\mu_k\}, \{r_{nk}\}$ to **minimize distortion measure**:

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \mu_k\|^2$$

e.g. two clusters $k = 1, 2$:

$$J = \sum_{\mathbf{x}_n \in C_1} \|\mathbf{x}_n - \mu_1\|^2 + \sum_{\mathbf{x}_n \in C_2} \|\mathbf{x}_n - \mu_2\|^2$$



Minimizing Distortion Measure

- ▶ Minimizing J directly is hard

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- ▶ However, two things are easy
 - ▶ If we know $\boldsymbol{\mu}_k$, minimizing J wrt r_{nk}
 - ▶ If we know r_{nk} , minimizing J wrt $\boldsymbol{\mu}_k$
- ▶ This suggests an **iterative procedure**
 - ▶ Start with **initial guess for** $\boldsymbol{\mu}_k$
 - ▶ Iteration of two steps:
 - ▶ Minimize J wrt r_{nk}
 - ▶ Minimize J wrt $\boldsymbol{\mu}_k$
 - ▶ Rinse and repeat until convergence

Determining Membership Variables

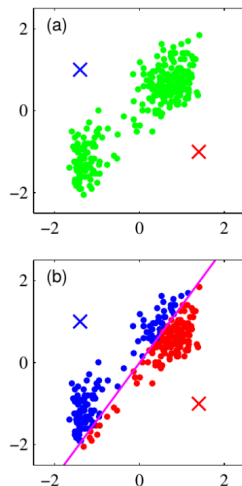
- Step 1 in an iteration of K-means is to minimize distortion measure J wrt. cluster membership variables r_{nk}

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- Terms for different data points $\mathbf{x}_n$ are independent, for each data point set r_{nk} to minimize

$$\sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- Simply set $r_{nk} = 1$ for the cluster center $\boldsymbol{\mu}_k$ with smallest distance.



Determining Cluster Centers

- Step 2: fix r_{nk} , minimize J wrt the cluster centers μ_k

$$J = \sum_{k=1}^K \sum_{n=1}^N r_{nk} \|\mathbf{x}_n - \mu_k\|^2 \quad \text{switch order of sums}$$

- So we can minimize wrt each μ_k separately
- Take derivative, set to zero:

$$\begin{aligned} 2 \sum_{n=1}^N r_{nk} (\mathbf{x}_n - \mu_k) &= 0 \\ \Leftrightarrow \mu_k &= \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}} \end{aligned}$$

i.e. mean of datapoints $\mathbf{x}_n$ assigned to cluster k ($\rightarrow$ “k-Means”)

k-Means Algorithm

- ▶ Start with an initial guess for μ_k
- ▶ Iteration of two steps:
 1. Minimize J wrt r_{nk}
 - ▶ Assign points to nearest cluster center
 2. Minimize J wrt μ_k
 - ▶ Set cluster center as average of points in cluster
- ▶ Rinse and repeat until convergence

Old Faithful Dataset

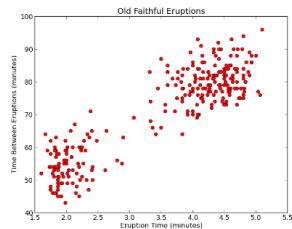
<https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat>

Description: (From R manual):

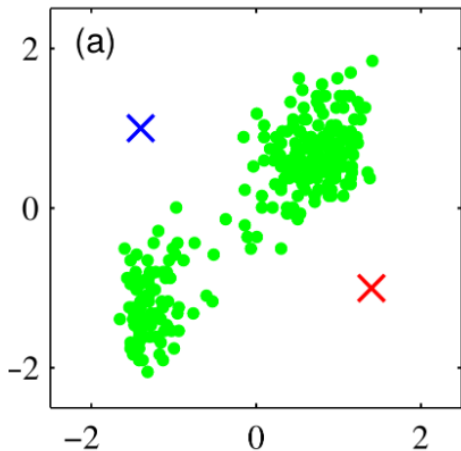
Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

A data frame with 272 observations on 2 variables.

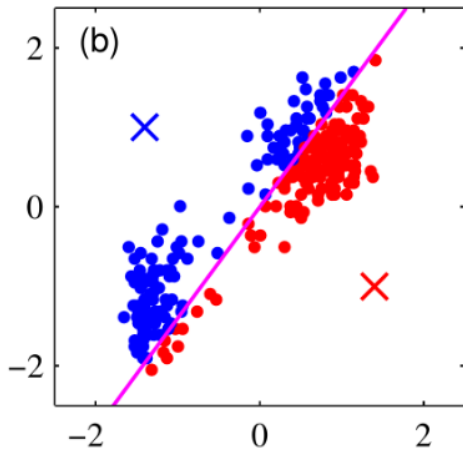
```
eruptions  numeric  Eruption time in mins
waiting    numeric  Waiting time to next eruption
```



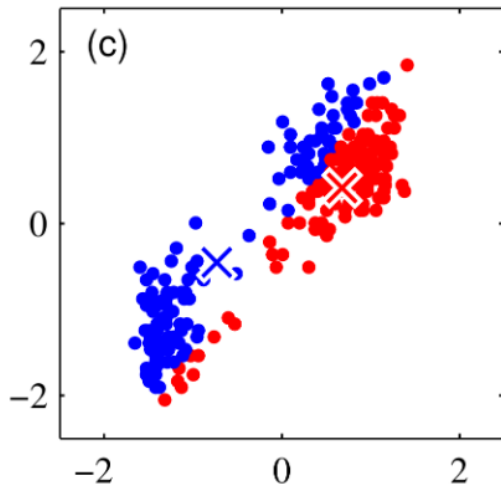
k-means example



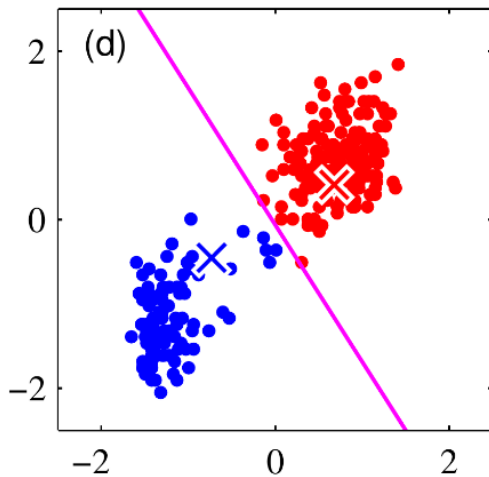
k-means example



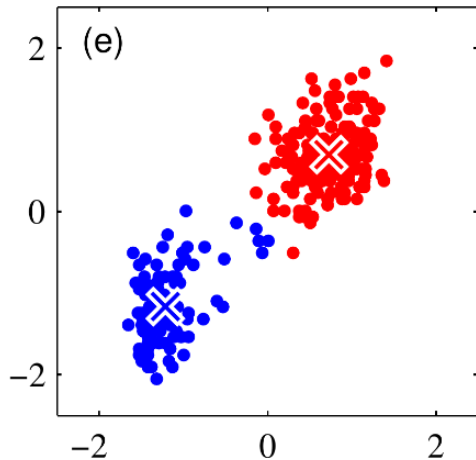
k-means example



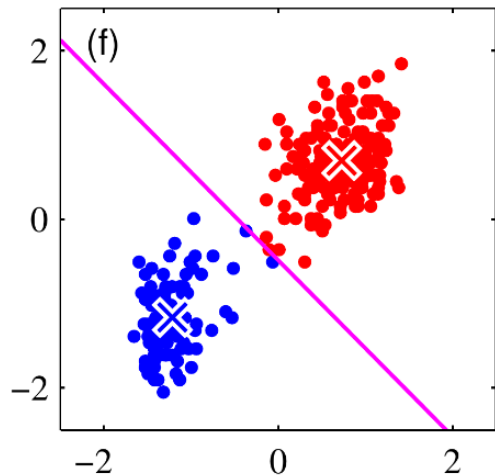
k-means example



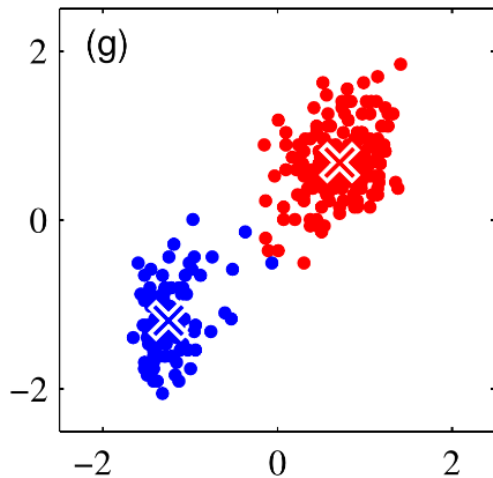
k-means example



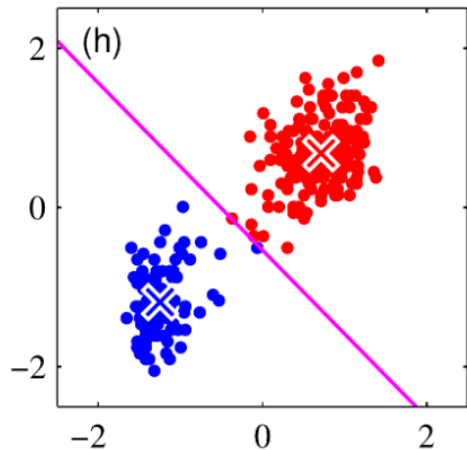
k-means example



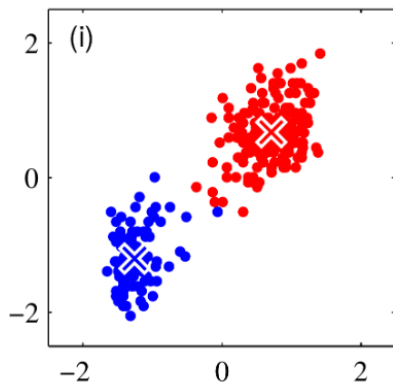
k-means example



k-means example

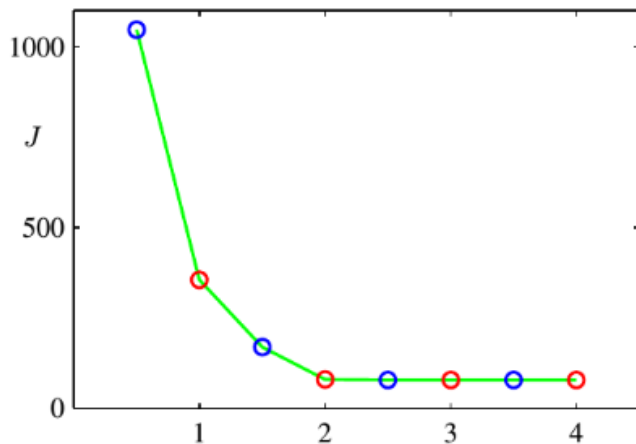


k-means example



Next step doesn't change membership — stop

Cost function J



k-means Convergence

- ▶ **Repeat steps until no change in cluster assignments.**
- ▶ For each step, value of J either goes down, or we stop.
- ▶ Finite number of possible assignments of data points to clusters, so we are guaranteed to converge eventually.
- ▶ Note it may be a local maximum rather than a global maximum to which we converge.

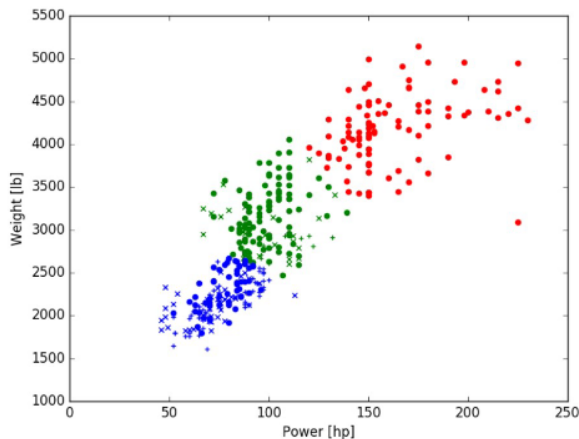
Clustering Cars based on Power and Weight

demo/k_means_car.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')
# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values
# extract origin as target value y
y = cars.iloc[:, 7].values
# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X) # determine min and max
X_normalized = min_max_scaler.transform(X)
# apply k-Means
km = KMeans(n_clusters=3, random_state=0).fit(X_normalized)
# plot cars
# U.S. : o / Europe: x / Japan : +
m = ['o' if o==1 else 'x' if o==2 else '+' for o in y]
# Cluster 1 : red / Cluster 2 : blue / Cluster 3 : green
c = ['red' if l==0 else 'blue' if l==1 else 'green' for l in km.labels_]
for i in range(0, len(X)):
    plt.scatter(X[i,0], X[i,1], color=c[i], marker=m[i])
plt.xlabel('Power [hp]')
plt.ylabel('Weight [lb]')
plt.show()
```


Clustering Cars based on Power and Weight: Plot



2. Evaluating Clustering: Some Notation

Read at home — here for completeness

- ▶ Consider a **set of data points** $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ $\mathbf{x}_i \in \mathbb{R}^m$
- ▶ Objective: Determine clustering (also: grouping, partitioning)

$$\mathcal{C} = \{C_1, \dots, C_k\} \quad \text{with} \quad C_i \subseteq \mathcal{D}$$

such that

- ▶ **clusters are disjoint** $\forall i \neq j: C_i \cap C_j = \emptyset$
- ▶ **each data point is assigned to a cluster**

$$\bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{D}$$

Evaluating Clustering

- ▶ **How can we evaluate the quality of a clustering computed?**
- ▶ **External measures** assume that **ideal clustering** is known (e.g., class labels assigned to data points)

$$\mathcal{I} = \{I_1, \dots, I_{|\mathcal{I}|}\} \quad \text{with} \quad I_i \subseteq \mathcal{D}$$

- ▶ **Internal measures** assume no knowledge of ideal clustering (i.e., we only know the data points and the clustering)

Purity

- Purity of a cluster is the fraction of data points therein that belongs to the dominant cluster from the ideal clustering

$$\text{purity}(C_i) = \frac{1}{|C_i|} \max_{I_j \in \mathcal{I}} |C_i \cap I_j|$$

- Purity of a clustering is then the weighted average of the purity values of its clusters

$$\text{purity}(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} \frac{|C_i|}{n} \text{purity}(C_i)$$

Purity



C_1

$$\text{purity}(C_1) = \frac{2}{3}$$



C_2

$$\text{purity}(C_2) = \frac{2}{4}$$



C_3

$$\text{purity}(C_3) = \frac{1}{2}$$

$$\text{purity}(\mathcal{C}) = \frac{3}{9} \cdot \frac{2}{3} + \frac{4}{9} \cdot \frac{2}{4} + \frac{2}{9} \cdot \frac{1}{2} \approx 0.56$$

- **BetaCV**, as an **internal measure**, considers the ratio of average distances between pairs of points within the same or different clusters

$$\text{BetaCV}(\mathcal{C}) = \frac{W_{\text{in}} / N_{\text{in}}}{W_{\text{out}} / N_{\text{out}}}$$

- with N_{in} and N_{out} as pairs of data points within the same or within different clusters

$$N_{\text{in}} = \frac{1}{2} \sum_{C_i \in \mathcal{C}} |C_i| (|C_i| - 1) \quad N_{\text{out}} = \frac{1}{2} \sum_{C_i, C_j \in \mathcal{C}, C_i \neq C_j} |C_i| |C_j|$$

- **BetaCV**, as an **internal measure**, considers the ratio of average distances between pairs of points within the same or different clusters

$$\text{BetaCV}(\mathcal{C}) = \frac{W_{\text{in}} / N_{\text{in}}}{W_{\text{out}} / N_{\text{out}}}$$

- and W_{in} and W_{out} as the total distance of pairs of data points within the same or within different clusters

$$W_{\text{in}} = \frac{1}{2} \sum_{C_i \in \mathcal{C}} \sum_{\mathbf{x}, \mathbf{y} \in C_i} d(\mathbf{x}, \mathbf{y}) \quad W_{\text{out}} = \frac{1}{2} \sum_{C_i, C_j \in \mathcal{C}} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y})$$

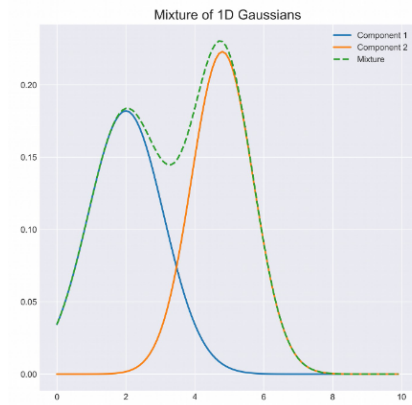
Dunn Index

- Dunn Index, as another **internal measure**, compares the minimal distance between any pair of data points from different clusters against the maximal distance between any pair of data points from the same cluster.

$$\text{DunnIndex}(\mathcal{C}) = \frac{\min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j, C_i \neq C_j} d(\mathbf{x}, \mathbf{y})}{\max_{\mathbf{x} \in C_i, \mathbf{y} \in C_i} d(\mathbf{x}, \mathbf{y})}$$

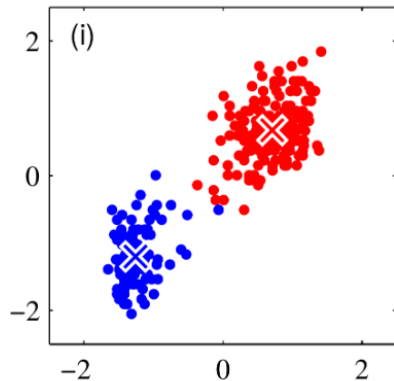
3. Gaussian Mixture Models

See Bishop (2006), Chapter 9; Murphy (2012), Chapter 11

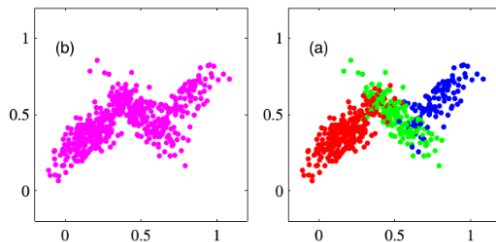


Hard Assignment versus Soft Assignment

- ▶ In the K-means algorithm, a **hard assignment** of points to clusters is made.
- ▶ However, for points near the decision boundary, this may not be such a good idea.
- ▶ Instead, we could think about making a **soft assignment** of points to clusters.



Gaussian Mixture Models

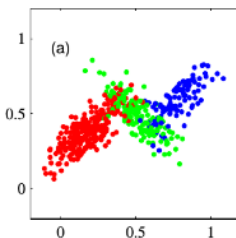


- ▶ The Gaussian mixture model (**or Mixture of Gaussians MoG**) models the data as a combination of Gaussians.
- ▶ Above shows a dataset generated by drawing samples from three different Gaussians.

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad p(z_k = 1) = \pi_k$$

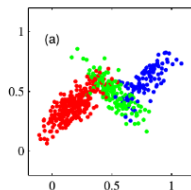
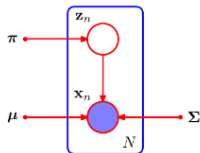
A Generative Model

- ▶ The mixture of Gaussians is a generative model.
- ▶ To generate a data point x_n , we first generate a value for a - discrete variable $z_n \in \{1, \dots, K\}$
- ▶ We then generate a value $\mathbf{x}_n \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ for the corresponding Gaussian



A Graphical Model

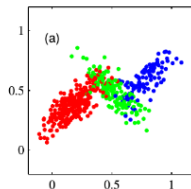
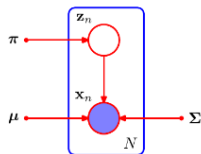
- Note z_n is a latent variable, unobserved.
- Need to give conditional distributions $p(z_n)$ and $p(x_n | z_n)$
- The one-of-K representation is helpful here: $z_{nK} \in \{0, 1\}$, $z_n = (z_{n1}, \dots, z_{nK})$



Graphical Model — Latent Component Variable

- ▶ Use a Bernoulli distribution for $p(z_n)$
 - ▶ i.e. $p(z_{nk} = 1) = \pi_k$
 - ▶ Parameters to this distribution $\{\pi_K\}$
- ▶ Must have $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^K \pi_k = 1$
- ▶ $p(\mathbf{z}_n) = \prod_{k=1}^K \pi_k^{z_{nk}}$

Graphical Model – Observed Variable

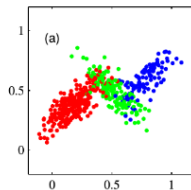
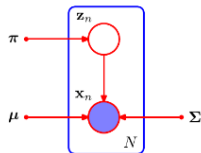


- Use a Gaussian distribution for $p(\mathbf{x}_n | z_n)$
- Parameters to this distribution $\{\mu_k, \Sigma_k\}$

$$p(\mathbf{x}_n | z_{nk} = 1) = \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)$$

$$p(\mathbf{x}_n | \mathbf{z}_n) = \prod_{k=1}^K \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k)^{z_{nk}}$$

A Graphical Model – Joint Distribution



- The full **joint distribution** is given by:

$$\begin{aligned} p(\mathbf{x}, \mathbf{z}) &= \prod_{n=1}^N p(\mathbf{z}_n) p(\mathbf{x}_n | \mathbf{z}_n) \\ &= \prod_{n=1}^N \prod_{k=1}^K \pi_k^{z_{nk}} \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}} \end{aligned}$$

Marginal over Observed (MoG) Variables

- ▶ The marginal distribution $p(\mathbf{x}_n)$ for this model is:

$$\begin{aligned} p(\mathbf{x}_n) &= \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n) p(\mathbf{x}_n | \mathbf{z}_n) \\ &= \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \end{aligned}$$

- ▶ A mixture of Gaussians

MoG Conditional over Latent Variable

- ▶ The conditional $p(z_{nk} = 1 | \mathbf{x}_n)$ will play an important role for learning
- ▶ It is denoted by $\gamma(z_{nk})$ can be computed as:

$$\begin{aligned}\gamma(z_{nk}) \equiv p(z_{nk} = 1 | \mathbf{x}_n) &= \frac{p(z_{nk} = 1) p(\mathbf{x}_n | z_{nk} = 1)}{\sum_{j=1}^K p(z_{nj} = 1) p(\mathbf{x}_n | z_{nj} = 1)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}\end{aligned}$$

- ▶ $\gamma(z_{nk})$ is the responsibility of component k for datapoint n

MoG Learning

- ▶ Given a set of observations $\{x_1, \dots, x_N\}$, without the latent
- ▶ variables z_n , **how can we learn the parameters?**
- ▶ Model parameters are $\theta = \{\pi_k, \mu_k, \Sigma_k\}$
- ▶ **Answer will be similar to k-means:**
 - ▶ If we know the latent variables z_n , fitting the Gaussians is easy
 - ▶ If we know the Gaussians μ_k, Σ_k , finding the latent variables is easy
- ▶ Rather than latent variables, **we will use responsibilities $\gamma(z_{nk})$**

MoG Maximum Likelihood Learning

- ▶ Given a set of observations $\{x_1, \dots, x_N\}$, **without the latent variables** z_n , how can we learn the parameters?
- ▶ Model parameters are $\theta = \{\pi_k, \mu_k, \Sigma_k\}$
- ▶ We can use the maximum likelihood criterion:

$$\begin{aligned}\theta_{ML} &= \arg \max_{\theta} \prod_{n=1}^N \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_k, \Sigma_k) \\ &= \arg \max_{\theta} \sum_{n=1}^N \log \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_k, \Sigma_k) \right\}\end{aligned}$$

- ▶ Unfortunately, closed-form solution not possible this time - log of sum rather than log of product

MoG Maximum Likelihood Learning - Problem

- ▶ Maximum likelihood criterion, 1-D:

$$\theta_{ML} = \arg \max_{\theta} \sum_{n=1}^N \log \left\{ \sum_{k=1}^K \pi_k \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ - (x_n - \mu_k)^2 / (2\sigma^2) \right\} \right\}$$

- ▶ Suppose we set $\mu_k = x_n$ for some k and n , then we have one term in the sum:

$$\begin{aligned} & \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} \exp \left\{ - (x_n - \mu_k)^2 / (2\sigma_k^2) \right\} \\ &= \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} \exp \left\{ - (0)^2 / (2\sigma_k^2) \right\} \end{aligned}$$

- ▶ In the limit as $\sigma_k \rightarrow 0$, this goes to ∞
- ▶ So ML solution is to set some $\mu_k = x_n$, and $\sigma_k = 0$!

ML for Mixture of Gaussians

- ▶ Keeping this problem in mind, we will develop an algorithm for ML estimation of the parameters for a MoG model
- ▶ Search for a local optimum.
- ▶ Consider the log-likelihood function

$$\ell(\boldsymbol{\theta}) = \sum_{n=1}^N \log \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- ▶ We can try taking derivatives and setting to zero, even though no closed form solution exists.

Maximizing Log-Likelihood - Means

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{n=1}^N \log \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \\ \frac{\partial}{\partial \boldsymbol{\mu}_k} \ell(\boldsymbol{\theta}) &= \sum_{n=1}^N \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) \\ &= \sum_{n=1}^N \gamma(z_{nk}) \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k)\end{aligned}$$

- Setting derivative to 0 , and multiply by $\boldsymbol{\Sigma}_k$

$$\begin{aligned}\sum_{n=1}^N \gamma(z_{nk}) \boldsymbol{\mu}_k &= \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \\ \Leftrightarrow \boldsymbol{\mu}_k &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \text{ where } N_k = \sum_{n=1}^N \gamma(z_{nk})\end{aligned}$$

Maximizing Log-Likelihood: Means and Covariances

- Note that the mean μ_k is a weighted combination of points $\mathbf{x}_n$, using the responsibilities $\gamma(z_{nk})$ for the cluster k

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n$$

- $N_k = \sum_{n=1}^N \gamma(z_{nk})$ is the effective number of points in the cluster
- A similar result comes from taking derivatives wrt. the covariance matrices Σ_k :

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \mu_k) (\mathbf{x}_n - \mu_k)^T$$

Maximizing Log-Likelihood: Mixing Coefficients

- ▶ We can also maximize wrt. the mixing coefficients π_k
- ▶ Note there is a constraint that $\sum_k \pi_k = 1$
- ▶ Use Lagrange multipliers
- ▶ End up with: $\pi_k = \frac{N_k}{N}$ average responsibility that component k takes.

Three Parameter Types and Three Equations

- ▶ These three equations a solution does not make

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N}$$

- ▶ **All depend on $\gamma(z_{nk})$, which depends on all 3 !**
- ▶ **But an iterative scheme can be used**

EM for Mixtures of Gaussians

- Initialize parameters, then iterate:

- **E step**: Calculate responsibilities using current parameters

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

- **M step**: Re-estimate parameters using these $y(z_{nk})$

$$\begin{aligned}\boldsymbol{\mu}_k &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) \mathbf{x}_n \\ \boldsymbol{\Sigma}_k &= \frac{1}{N_k} \sum_{n=1}^N \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \\ \pi_k &= \frac{N_k}{N}\end{aligned}$$

- This algorithm is known as the [expectation-maximization algorithm \(EM\)](#)
 - Next we describe its general form, why it works, and why it's called EM (but first an example)

The Likelihood

- ▶ The form of the Gaussian mixture distribution is governed by the parameters $\boldsymbol{\pi}, \boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, where we have used the notation $\boldsymbol{\pi} \equiv \{\pi_1, \dots, \pi_k\}, \boldsymbol{\mu} \equiv \{\mu_1, \dots, \mu_k\}, \boldsymbol{\Sigma} \equiv \{\Sigma_1, \dots, \Sigma_k\}$.
- ▶ One way to set the values of these parameters is to use **maximum likelihood**.
- ▶ The log of the likelihood function is given by

$$\ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

where $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.

Problems with optimizing the likelihood

- ▶ The situation is now much more complex than with a single Gaussian, due to the presence of the summation over k inside the logarithm.
- ▶ As a result, the maximum likelihood solution for the parameters no longer has a closed-form analytical solution.
- ▶ One approach to maximizing the likelihood function is to use iterative numerical optimization techniques.
- ▶ Gradient methods could be used but are painful to implement. $\implies$ Non-convex optimization problem! (multiple optima possible)

Example in one dimension

- Observations $x_1 \dots x_n$
- $K = 2$ Gaussians with unknown μ, σ^2
- Estimation trivial if we know the source of each observation

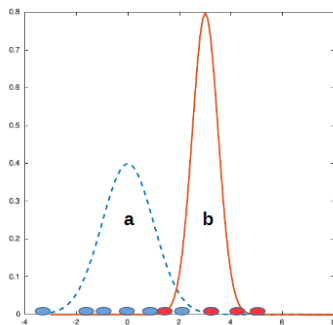
$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$
$$\sigma_b^2 = \frac{(x_1 - \mu_1)^2 + \dots + (x_n - \mu_n)^2}{n_b}$$



Example in one dimension

- Observations $x_1 \dots x_n$
- $K = 2$ Gaussians with unknown μ, σ^2
- Estimation trivial if we know the source of each observation

$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$
$$\sigma_b^2 = \frac{(x_1 - \mu_1)^2 + \dots + (x_n - \mu_n)^2}{n_b}$$



Example: Expectation Maximization in 1d (II)

- ▶ What if we don't know the source?
- ▶ If we knew parameters of the Gaussians (μ, σ^2)



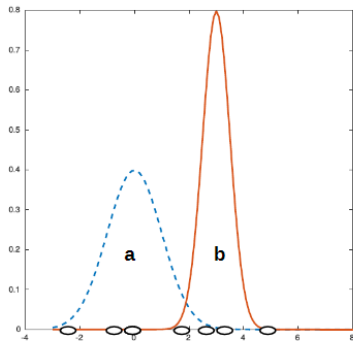
Example: Expectation Maximization in 1d (II)

- What if we don't know the source?
- If we knew parameters of the Gaussians (μ, σ^2)

→ can guess whether point is more likely to be *a* or *b*.

$$P(b | x_i) = \frac{P(x_i | b) P(b)}{P(x_i | b) P(b) + P(x_i | a) P(a)}$$

$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$



EM Algorithm (in 1d)

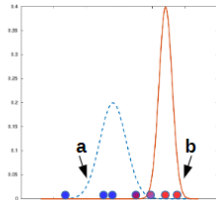
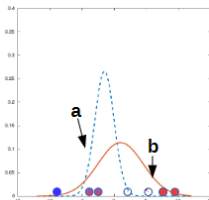
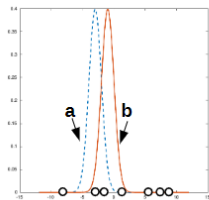
A fundamental problem:

- ▶ we need (μ_a, σ_a^2) and (μ_b, σ_b^2) to guess the source of the points.
- ▶ we need to know the source to estimate (μ_a, σ_a^2) and (μ_b, σ_b^2) .

EM algorithm:

1. **Start** with two randomly placed Gaussians (μ_a, σ_a^2) and (μ_b, σ_b^2) .
2. **E(xpectation) step:**
 - ▶ for each point: $P(b | x_i)$ = does it look like it came from b ?
3. **M(aximization)-step:**
 - ▶ adjust (μ_a, σ_a^2) and (μ_b, σ_b^2) to fit points assigned to them.
4. **Iterate until convergence.**

EM in 1d



$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$

$$b_i = P(b | x_i) = \frac{P(x_i | b) P(b)}{P(x_i | b) P(b) + P(x_i | a) P(a)}$$

$$a_i = P(a | x_i) = 1 - b_i$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \dots + b_n x_{n_h}}{b_1 + b_2 + \dots + b_n}$$

$$\sigma_b^2 = \frac{b_1 (x_1 - \mu_1)^2 + \dots + b_n (x_n - \mu_n)^2}{b_1 + b_2 + \dots + b_n}$$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \dots + a_n x_{n_n}}{a_1 + a_2 + \dots + a_n}$$

$$\sigma_a^2 = \frac{a_1 (x_1 - \mu_1)^2 + \dots + a_n (x_n - \mu_n)^2}{a_1 + a_2 + \dots + a_n}$$

→ We could also estimate priors:

$$P(b) = (b_1 + b_2 + \dots + b_n) / n$$

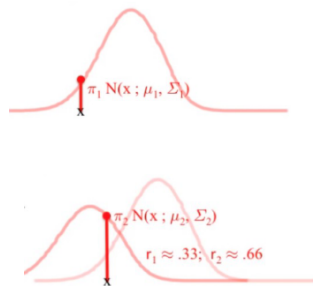
$$P(a) = 1 - P(b)$$

EM in the multidimensional case

- ▶ Start with parameters describing each cluster
- ▶ Mean μ_c , Covariance Σ_c , "size" π_c
- ▶ **E-step ("Expectation"):**
 - ▶ For **each observation/point** x_i
 - ▶ Compute " r_{ic} ", the probability that it belongs to cluster c .
 - ▶ Compute its probability under model c .
 - ▶ Normalize to sum to one (over clusters c).

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i; \mu_{c'}, \Sigma_{c'})}$$

- ▶ If x_i is very likely under the c -th Gaussian, it gets high weight.
- ▶ Denominator just makes r 's sum to one.



EM in the multidimensional case

► M-step ("Maximization step"):

- For each cluster (Gaussian) $z = c$
- Update its parameters using the (weighted) data points

$$N_c = \sum_i r_{ic} \quad \text{Total responsibility allocated to cluster } c$$

$$\pi_c = \frac{N_c}{N} \quad \text{Fraction of total assigned to cluster } c$$

$$\mu_c = \frac{1}{N_c} \sum_i r_{ic} x_i \quad \Sigma_c = \frac{1}{N_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c)$$

Weighted mean of assigned data weighted covariance of assigned data
(use new weighted means here)

Expectation-Maximization: Summary

- Likelihood of the data

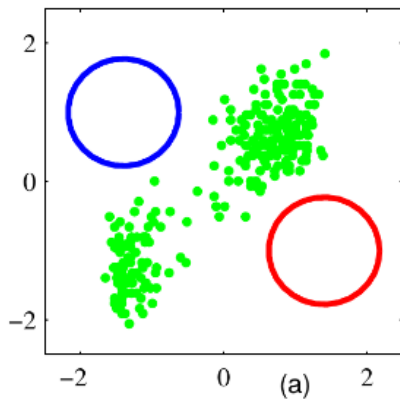
$$P(x_1, \dots, x_N) = \prod_{i=1}^N \sum_{k=1}^K P(x_i | k) P(k)$$

- Each step increases the log-likelihood of our model

$$\ln p(\mathbf{X} | \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

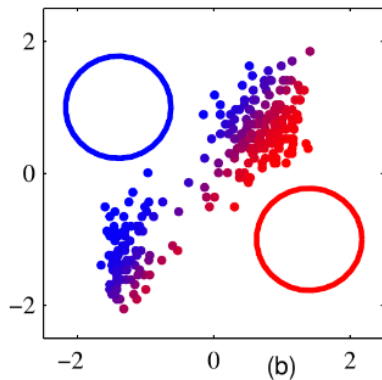
- Iterate until convergence
 - Convergence guaranteed — another ascent method.
- Cannot discover k .

MoG EM — Example



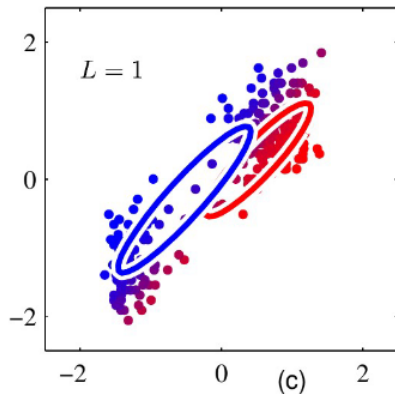
- ▶ Same initialization as with K-means before
- ▶ Often, K-means is actually used to initialize EM

MoG EM — Example



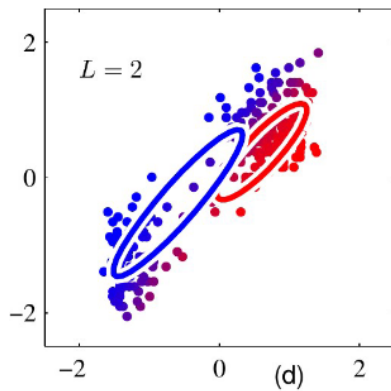
Calculate responsibilities $\gamma(z_{nk})$

MoG EM — Example



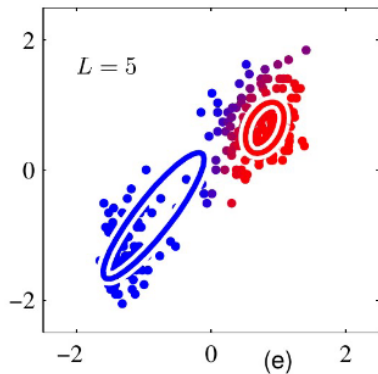
Calculate model parameters $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$ using these responsibilities

MoG EM — Example



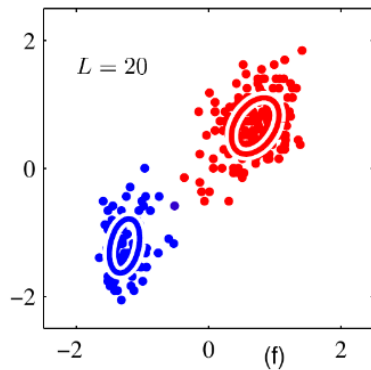
Iteration 2

MoG EM — Example



Iteration 5

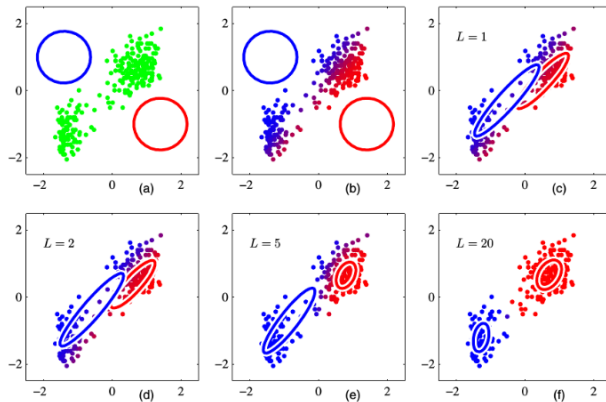
MoG EM — Example



Iteration 20 — converged

Gaussian mixture models: $d > 1$

See Bishop (2006) for details



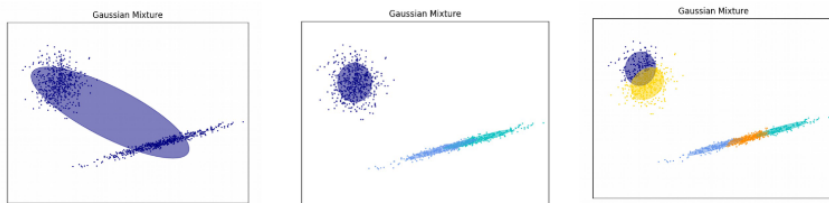
Bayesian Information Criterion (BIC)

- ▶ How to pick k ?
- ▶ Probabilistic model:
$$L = \ln p(\mathbf{X} \mid \pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_k, \Sigma_k) \right\}$$
 - ▶ Tries to “fit” the data (maximize likelihood)
- ▶ Choose K that makes L as large as possible?
 - ▶ $K = n$: each data point has its own “source”
 - ▶ may not work well for new data points
- ▶ Split points into training set $\mathbf{T}$ and validation set $\mathbf{V}$
 - ▶ for each k : fit parameters of $\mathbf{T}$
 - ▶ measure likelihood of $\mathbf{V}$
 - ▶ sometimes still best when $k = n$
- ▶ “Occam’s razor”:
 - ▶ Pick the “simplest” of all models that fits the data.
 - ▶ Assess, e.g., via Bayes Information Criterion (BIC): $\max_p \{L - 1/2p * \log(n)\}$
 - ▶ L : Likelihood; p : # Parameters in the model - how simple is the model.

Hands-on example

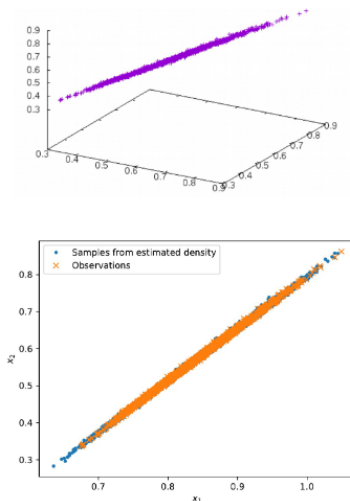
<https://scikit-learn.org/stable/modules/mixture.html>
`demo/GMM_scikit_example.py`

- ▶ Plot the confidence ellipsoids of a mixture of two Gaussians obtained with Expectation Maximization (GaussianMixture class)
- ▶ The model has access to 1, 3, and 5 components with which to fit the data. Note that the Expectation Maximization model will necessarily use ALL components
- ▶ In the 5-component example, we can see that the Expectation Maximization model splits some components arbitrarily, because it is trying to fit too many components.



Hands-on example 2

- ▶ We simulate a bunch of data (e.g., an ergodic set). — it is in a text file (ergodic_data.txt - 3 dimensions)
- ▶ We apply GMM ([build_density.py](#))
- ▶ We can sample data from the fitted GMM model ([sample.py](#))



GMM — Cars based on Power and Weight

```
import itertools
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn import mixture
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold', 'darkorange'])

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\\s+')
# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values
# extract origin as target value y
y = cars.iloc[:, 7].values

# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X) # determine min and max
X_normalized = min_max_scaler.transform(X)
## Fit a Gaussian mixture with EM using five components
gmm = mixture.GaussianMixture(n_components=5, covariance_type='full').fit(X_normalized)
plot_results(X_normalized, gmm.predict(X_normalized), gmm.means_, gmm.covariances_, 0,
              'Gaussian Mixture')
plt.show()
```

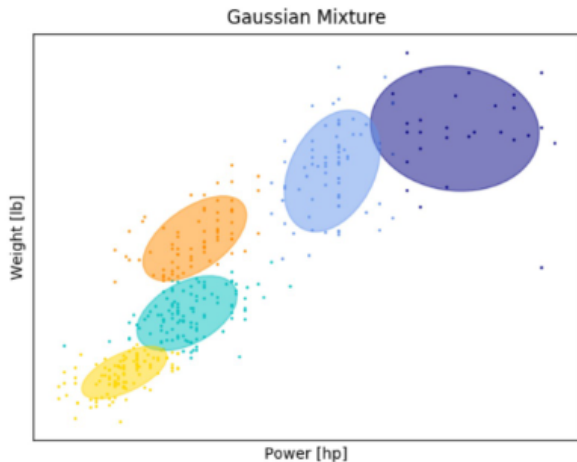
Cont.

```
def plot_results(X, Y_, means, covariances, index, title):
    splot = plt.subplot(1, 1, 1 + index)
    for i, (mean, covar, color) in enumerate(zip(
        means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
        plt.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

        # Plot an ellipse to show the Gaussian component
        angle = np.arctan(u[1] / u[0])
        angle = 180. * angle / np.pi # convert to degrees
        ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
        ell.set_clip_box(splot.bbox)
        ell.set_alpha(0.5)
        splot.add_artist(ell)

plt.xticks(())
plt.yticks(())
plt.xlabel('Power [hp]')
plt.ylabel('Weight [lb]')
plt.title(title)
```

GMM — Cars based on Power and Weight



4. Expectation Maximization: A General Version of EM

- ▶ In general, we are interested in maximizing the likelihood

$$p(\mathbf{X} \mid \theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} \mid \theta)$$

where $\mathbf{X}$ denotes all observed variables, and $\mathbf{Z}$ denotes all latent (hidden, unobserved) variables

- ▶ Assume that maximizing $p(\mathbf{X} \mid \theta)$ is difficult (e.g. mixture of Gaussians)
- ▶ But maximizing $p(\mathbf{X}, \mathbf{Z} \mid \theta)$ is tractable (everything observed)
 - ▶ $p(\mathbf{X}, \mathbf{Z} \mid \theta)$ is referred to as the **complete-data likelihood function**, which we don't have

A Lower Bound

- ▶ The strategy for optimization will be to introduce a lower bound on the likelihood
 - ▶ This lower bound will be based on the complete-data likelihood, which is easy to optimize
- ▶ Iteratively increase this lower bound
- ▶ Make sure we're increasing the likelihood while doing so

A Decomposition Trick

- To obtain the lower bound, we use a decomposition:

$$\ln p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\theta}) = \ln p(\mathbf{X} \mid \boldsymbol{\theta}) + \ln p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta}) \text{ product rule}$$

$$\ln p(\mathbf{X} \mid \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + KL(q \parallel p)$$

$$\mathcal{L}(q, \boldsymbol{\theta}) \equiv \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X}, \mathbf{Z} \mid \boldsymbol{\theta})}{q(\mathbf{Z})} \right\}$$

$$KL(q \parallel p) \equiv - \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z} \mid \mathbf{X}, \boldsymbol{\theta})}{q(\mathbf{Z})} \right\}$$

- $KL(q \parallel p)$ is known as the Kullback-Leibler divergence (KL-divergence), and is ≥ 0
(next slide) $\longrightarrow$ Hence $\ln p(\mathbf{X} \mid \boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta})$

TODO colors

Kullback-Leibler Divergence

- ▶ $KL(p(x)||q(x))$ is a measure of the difference between distributions $p(x)$ and $q(x)$:

$$KL(p(x)||q(x)) = - \sum_x p(x) \log \frac{q(x)}{p(x)}$$

- ▶ Motivation: average additional amount of information required to encode x using code assuming distribution $q(x)$ when x actually comes from $p(x)$
- ▶ Note it is not symmetric: $KL(q(x)||p(x)) \neq KL(p(x)||q(x))$ in general
- ▶ It is non-negative:
 - ▶ Jensen's inequality: $-\ln(\sum_x x p(x)) \leq -\sum_x p(x) \ln x$
 - ▶ Apply to KL:

$$KL(p||q) = - \sum_x p(x) \log \frac{q(x)}{p(x)} \geq -\ln \left(\sum_x \frac{q(x)}{p(x)} p(x) \right) = -\ln \sum_x q(x) = 0$$

Increasing the Lower Bound — E-step

- EM is an iterative optimization technique which tries to maximize this lower bound: $\ln p(\mathbf{X} | \boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta})$

E step: Fix $\boldsymbol{\theta}^{old}$, maximize $\mathcal{L}(q, \boldsymbol{\theta}^{old})$ wrt q

i.e. Choose distribution q to maximize $\mathcal{L}$

Reordering bound:

$$\mathcal{L}(q, \boldsymbol{\theta}^{old}) = \ln p(\mathbf{X} | \boldsymbol{\theta}^{old}) - KL(q || p)$$

$\ln p(\mathbf{X} | \boldsymbol{\theta}^{old})$ does not depend on q

Maximum is obtained when $KL(q || p)$ is as small as possible

Occurs when $q = p$, i.e. $q(\mathbf{Z}) = p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta})$

This is the posterior over $\mathbf{Z}$, recall these are the **responsibilities** from MoG model

Increasing the Lower Bound — M-step

M step: Fix q , maximize $\mathcal{L}(q, \theta)$ wrt θ

The maximization problem is on

$$\begin{aligned}\mathcal{L}(q, \theta) &= \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln p(\mathbf{X}, \mathbf{Z} | \theta) - \sum_{\mathbf{Z}} q(\mathbf{Z}) \ln q(\mathbf{Z}) \\ &= \sum_{\mathbf{Z}} p(\mathbf{Z} | \mathbf{X}, \theta^{\text{old}}) \ln p(\mathbf{X}, \mathbf{Z} | \theta) - \sum_{\mathbf{Z}} p(\mathbf{Z} | \mathbf{X}, \theta^{\text{old}}) \ln p(\mathbf{Z} | \mathbf{X}, \theta^{\text{old}})\end{aligned}$$

Second term is constant with respect to θ

First term is $\ln$ of complete data likelihood, which is assumed easy to optimize

Expected complete log likelihood - what we think complete data likelihood will be

Why does EM work?

- ▶ In the M-step we changed from θ^{old} to θ^{new}
- ▶ This increased the lower bound L , unless we were at a maximum (so we would have stopped)
- ▶ This must have caused the log likelihood to increase
- ▶ The E-step set q to make the KL-divergence 0:

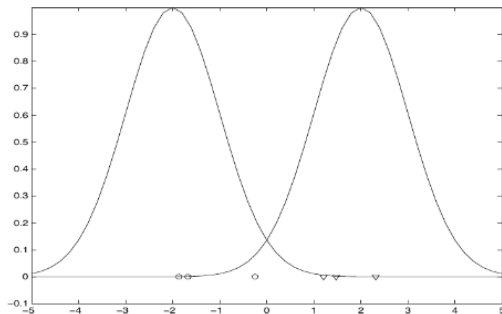
$$\ln p(\mathbf{X} | \theta^{\text{old}}) = \mathcal{L}(q, \theta^{\text{old}}) + KL(q||p) = \mathcal{L}(q, \theta^{\text{old}})$$

- ▶ Since the lower bound L increased when we moved from θ^{old} to θ^{new}

$$\begin{aligned}\ln p(\mathbf{X} | \theta^{\text{old}}) &= \mathcal{L}(q, \theta^{\text{old}}) < \mathcal{L}(q, \theta^{\text{new}}) \\ &= \ln p(\mathbf{X} | \theta^{\text{new}}) - KL(q||p^{\text{new}})\end{aligned}$$

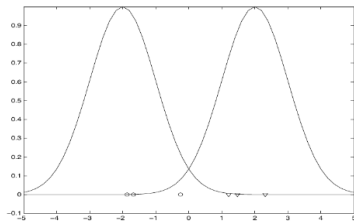
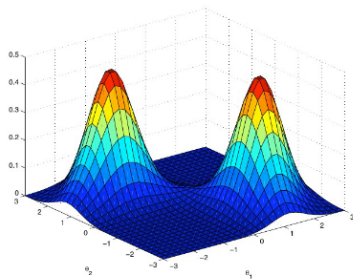
- ▶ So the log-likelihood has increased going from θ^{old} to θ^{new}

Bounding Example



Consider 2 component 1-D MoG with known variances.

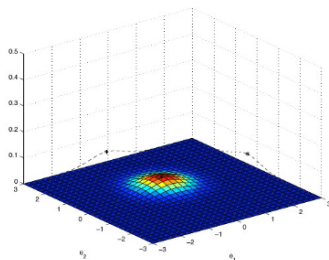
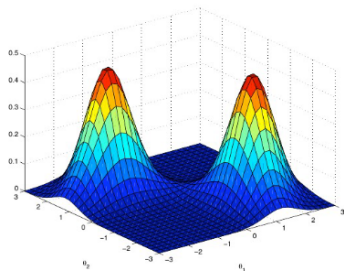
Bounding Example



True likelihood function

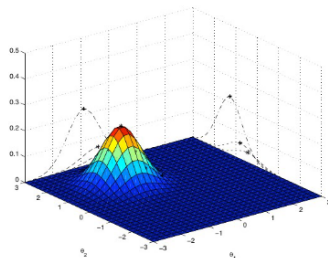
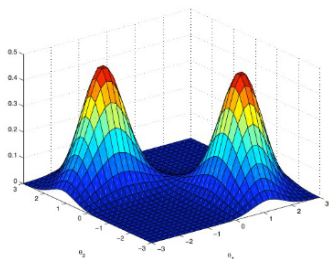
Recall we're fitting means θ_1, θ_2

Bounding Peaks



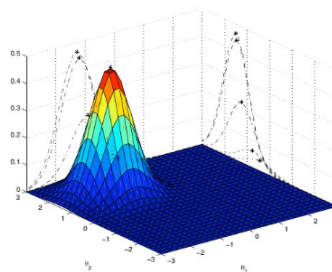
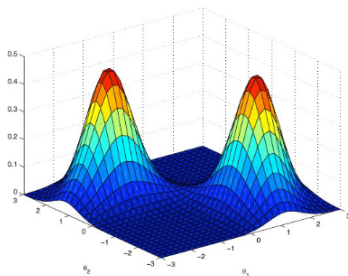
- ▶ Lower bound the likelihood function using averaging distribution $q(\mathbf{Z})$
 - ▶ $\ln p(\mathbf{X} | \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + KL(q(\mathbf{Z}) || p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta}))$
 - ▶ Since $q(\mathbf{Z}) = p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta}^{\text{old}})$, bound is tight (equal to actual likelihood) at $\boldsymbol{\theta} = \boldsymbol{\theta}^{\text{old}}$

Bounding Peaks



- Lower bound the likelihood function using averaging distribution $q(\mathbf{Z})$
 - $\ln p(\mathbf{X} | \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + KL(q(\mathbf{Z}) || p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta}))$
 - Since $q(\mathbf{Z}) = p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta}^{\text{old}})$, bound is tight (equal to actual likelihood) at $\boldsymbol{\theta} = \boldsymbol{\theta}^{\text{old}}$

Bounding Peaks



- Lower bound the likelihood function using averaging distribution $q(\mathbf{Z})$
 - $\ln p(\mathbf{X} | \boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + KL(q(\mathbf{Z}) || p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta}))$
 - Since $q(\mathbf{Z}) = p(\mathbf{Z} | \mathbf{X}, \boldsymbol{\theta}^{\text{old}})$, bound is tight (equal to actual likelihood) at $\boldsymbol{\theta} = \boldsymbol{\theta}^{\text{old}}$

Recall About the EM Algorithm

Some good things about EM:

- ▶ no learning rate (step-size) parameter.
- ▶ automatically enforces parameter constraints.
- ▶ very fast for low dimensions.
- ▶ each iteration guaranteed to improve likelihood.

Some bad things about EM:

- ▶ can get stuck in local minima.
- ▶ can be slower than conjugate gradient (especially near convergence).
- ▶ requires expensive inference step.
- ▶ is a maximum likelihood/MAP (maximum a posterior) method.

EM — Summary

- ▶ EM finds local maximum to likelihood

$$p(\mathbf{X} \mid \theta) = \sum_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z} \mid \theta)$$

- ▶ Iterates two steps:
 - ▶ E step “fills” in the missing variables $\mathbf{Z}$ (calculates their distribution)
 - ▶ M step maximizes expected complete log likelihood (expectation wrt E step distribution)
- ▶ This works because these two steps are performing a coordinatewise hill-climbing on a lower bound on the likelihood $p(\mathbf{X} \mid \theta)$

5. Hierarchical Clustering

- ▶ k-Means determines a flat clustering of data points; there is no relationship between the clusters
- ▶ Hierarchical clustering determines a sequence of increasingly fine-grained clusterings

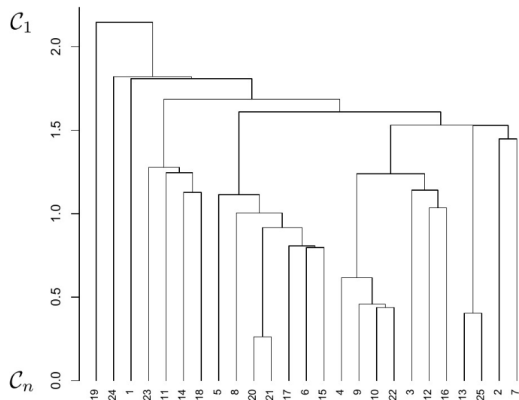
$$\mathcal{C}_1, \dots, \mathcal{C}_n$$

- ▶ $\mathcal{C}_1 = \{\mathcal{D}\}$ contains all data points in a single cluster
- ▶ $\mathcal{C}_n = \{\{\mathbf{x}_i\} : \mathbf{x}_i \in \mathcal{D}\}$ contains one cluster per data point
- ▶ Clustering $\mathcal{C}_i$ is contained in clustering $\mathcal{C}_{i-1}$

$$\forall \mathcal{C}_j \in \mathcal{C}_i : \exists \mathcal{C}_l \in \mathcal{C}_{i-1} : \mathcal{C}_j \subseteq \mathcal{C}_l$$

Dendrogram

Sequence of clusterings can be visualized in a dendrogram



Hierarchical Agglomerative vs. Divisive Clustering

- ▶ Hierarchical Agglomerative Clustering (HAC)
 - ▶ starts with the most fine-grained clustering $\mathcal{C}_n$
 - ▶ proceeds bottom-up and merges the two closest clusters in $\mathcal{C}_i$ to obtain the more coarse-grained clustering $\mathcal{C}_{i-1}$
- ▶ Hierarchical Divisive Clustering (HDC)
 - ▶ starts with the most coarse-grained clustering $\mathcal{C}_1$
 - ▶ proceeds top-down and splits one of the clusters in $\mathcal{C}_{i-1}$ to obtain the more fine-grained clustering $\mathcal{C}_i$

Hierarchical Agglomerative vs. Divisive Clustering

- ▶ Hierarchical Agglomerative Clustering (HAC)
 - ▶ starts with the most fine-grained clustering $\mathcal{C}_n$
 - ▶ proceeds bottom-up and merges the two closest clusters in $\mathcal{C}_i$ to obtain the more coarse-grained clustering $\mathcal{C}_{i-1}$
- ▶ So far, we can only measure distance between data points, but we need a measure of distance between clusters

Linkage Criteria

Linkage criteria measure distance between two clusters based on the distance between data points therein

Single-Link

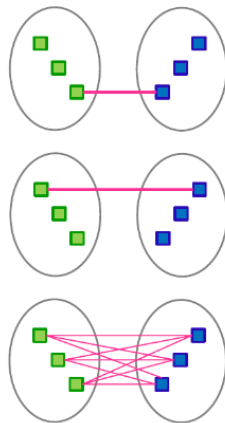
$$\delta(C_i, C_j) = \min \{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$$

Complete-Link

$$\delta(C_i, C_j) = \max \{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}$$

Average-Link

$$\delta(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\mathbf{x} \in C_i} \sum_{\mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y})$$



Pseudocode: Hierarchical Agglomerative Clustering

```
// Start with each data point in a separate cluster
 $\mathcal{C}_n = \{\{\mathbf{x}_i\} : \mathbf{x}_i \in \mathcal{D}\};$ 

for(int t = n; t > 1; t--) {
    // Determine the two clusters closest to each other
     $C_i^*, C_j^* = \arg \min_{C_i, C_j \in \mathcal{C}_t : C_i \neq C_j} \delta(C_i, C_j);$ 

    // Merge the two clusters
     $\mathcal{C}_{t-1} = (\mathcal{C}_t \setminus \{C_i^*, C_j^*\}) \cup \{C_i^* \cup C_j^*\};$ 
}
```

HAC Example

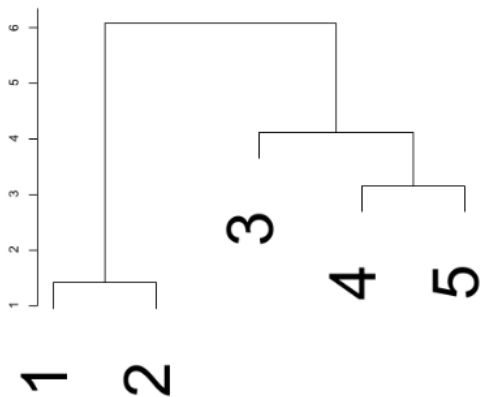
- Consider the following data points in $\mathbb{R}^2$

$$\begin{array}{l} \mathbf{x}_1 = (1, 0) \\ \mathbf{x}_2 = (2, 1) \\ \mathbf{x}_3 = (8, 0) \\ \mathbf{x}_4 = (12, 1) \\ \mathbf{x}_5 = (15, 1) \end{array} \quad \mathbf{d} = \begin{bmatrix} 0.00 & 1.41 & 7.00 & 11.05 & 14.04 \\ & 0.00 & 6.08 & 10.00 & 13.04 \\ & & 0.00 & 4.12 & 7.07 \\ & & & 0.00 & 3.00 \\ & & & & 0.00 \end{bmatrix}$$

- With distance matrix d

HAC with Single-Link Example

HAC with single-link based on distance matrix d



$$\mathcal{C}_1 = \{\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}\}$$

$$\mathcal{C}_2 = \{\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}\}$$

$$\mathcal{C}_3 = \{\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3\}, \{\mathbf{x}_4, \mathbf{x}_5\}\}$$

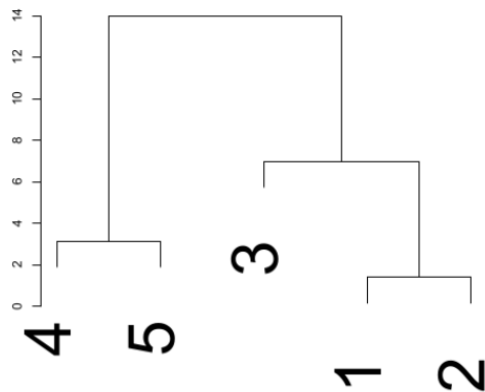
$$\mathcal{C}_4 = \{\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3\}, \{\mathbf{x}_4\}, \{\mathbf{x}_5\}\}$$

$$\mathcal{C}_5 = \{\{\mathbf{x}_1\}, \{\mathbf{x}_2\}, \{\mathbf{x}_3\}, \{\mathbf{x}_4\}, \{\mathbf{x}_5\}\}$$

$$\mathbf{d} = \begin{bmatrix} 0.00 & 1.41 & 7.00 & 11.05 & 14.04 \\ & 0.00 & 6.08 & 10.00 & 13.04 \\ & & 0.00 & 4.12 & 7.07 \\ & & & 0.00 & 3.00 \\ & & & & 0.00 \end{bmatrix}$$

HAC with Single-Link Example

HAC with complete-link based on distance matrix d



$$\mathcal{C}_1 = \{\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5\}\}$$

$$\mathcal{C}_2 = \{\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}, \{\mathbf{x}_4, \mathbf{x}_5\}\}$$

$$\mathcal{C}_3 = \{\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3\}, \{\mathbf{x}_4, \mathbf{x}_5\}\}$$

$$\mathcal{C}_4 = \{\{\mathbf{x}_1, \mathbf{x}_2\}, \{\mathbf{x}_3\}, \{\mathbf{x}_4\}, \{\mathbf{x}_5\}\}$$

$$\mathcal{C}_5 = \{\{\mathbf{x}_1\}, \{\mathbf{x}_2\}, \{\mathbf{x}_3\}, \{\mathbf{x}_4\}, \{\mathbf{x}_5\}\}$$

$$\mathbf{d} = \begin{bmatrix} 0.00 & 1.41 & 7.00 & 11.05 & 14.04 \\ & 0.00 & 6.08 & 10.00 & 13.04 \\ & & 0.00 & 4.12 & 7.07 \\ & & & 0.00 & 3.00 \\ & & & & 0.00 \end{bmatrix}$$

Clustering Cars based on Power and Weight

demo/HAC_example.py

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')

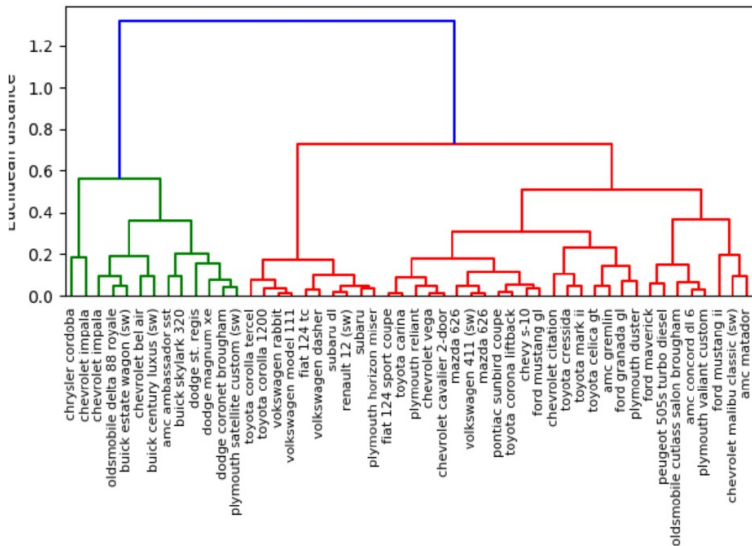
# keep a sample of 50 cars
cars = cars.sample(50, random_state=0)
# extract labels
labels = cars.iloc[:, 8].values
# extract power and weight as data matrix X
X = cars.iloc[:, [3, 4]].values

# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X) # determine min and max
X_normalized = min_max_scaler.transform(X)

# perform hierarchical agglomerative clustering using complete linkage
clusters = linkage(X_normalized, method='complete', metric='euclidean')

# plot dendrogram
dendrogram = dendrogram(clusters, labels=labels)
plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show()
```

Clustering Cars based on Power and Weight



6. Density-based Clustering

- ▶ k-Means as a representative-based clustering method can only find convex clusters and must assign every data point to a cluster.
- ▶ Density-based clustering methods determine clusters as regions having consistently high density and label isolated data points as noise
- ▶ Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Density-Based Clustering

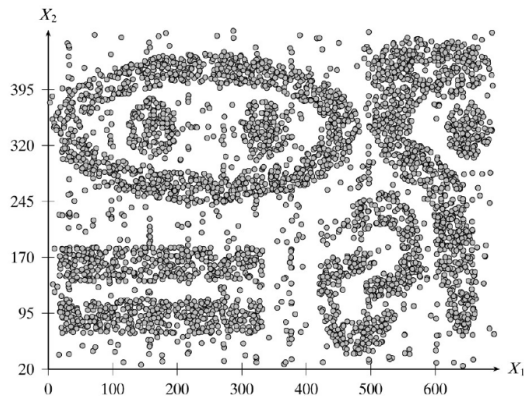


Fig. From Zaki and Meira (2014)

DBSCAN — the idea

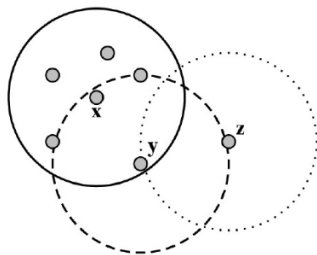
- ▶ Epsilon Neighborhood of a data point x

$$N_{\epsilon}(\mathbf{x}) = \{\mathbf{y} \mid d(\mathbf{x}, \mathbf{y}) \leq \epsilon\}$$

contains all points having distance less than or equal to ϵ

- ▶ Data point x is called a core to ϵ its epsilon neighborhood contains at least `minpts` data points (including x)
- ▶ Data point x is called a border point, if it is not a core, but belongs to the epsilon neighborhood of a core
- ▶ All other data points are considered noise

Core, Border, and Noise



$$\text{minpts} = 6$$

- ▶ Data point x is a core
- ▶ Data point y is a border point
- ▶ Data point z is noise

Reachability

- ▶ Data point x is directly reachable from data point y , if y is a core and x belongs to the epsilon neighborhood of y , i.e.

$$\mathbf{x} \in N_\epsilon(\mathbf{y})$$

- ▶ Data point x is (density) reachable from data point y ,
- ▶ if there is a chain of data points $x_0, \dots, x_l$, so that

$$x_0 = x \wedge x_l = y$$

$\forall 1 \leq i \leq l: x_i$ is directly reachable from x_{i-1}

- ▶ Reachability is not symmetric, since the data point y could be a core, but the data point x is not

Connectedness and Density-Based Clusters

- ▶ Two data points x and y are called connected, if there is a core z , so that both x and y are reachable from z
- ▶ Density-based cluster is a maximal subset of connected data points, i.e., there are no data points that could be added

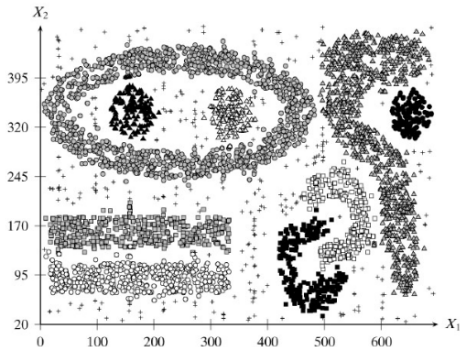
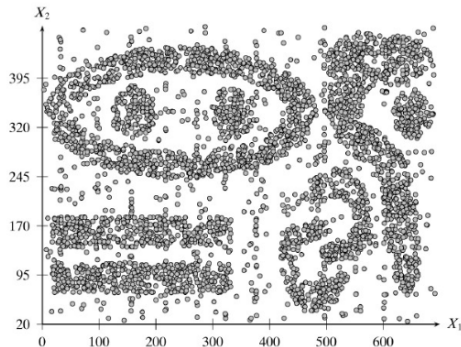
- ▶ Intuition:
 - ▶ Compute epsilon neighborhoods for all data points
 - ▶ Determine all cores
 - ▶ Determine noise
 - ▶ Grow a new density-based cluster from each data point that does not yet belong to an already-determined cluster
- ▶ Note that DBSCAN is not deterministic, since the assignment of data point to clusters depends on the order in which data points are considered

Pseudo-code DBSCAN

```
dbScan( $\mathcal{D}$ ,  $\epsilon$ ,  $minpts$ ) {  
  // Cores  
   $Cores = \emptyset$ ;  
  
  for ( $\mathbf{x} \in \mathcal{D}$ ) {  
    // Compute epsilon neighborhoods  
     $N_\epsilon(\mathbf{x}) = \text{computeNeighborhood}(\mathbf{x}, \epsilon)$ ;  
  
    // Initialize cluster id  
     $id(\mathbf{x}) = \emptyset$ ;  
  
    // Check whether data point is a core  
    if ( $|N_\epsilon(\mathbf{x})| \geq minpts$ )  $Cores = Cores \cup \{\mathbf{x}\}$ ;  
  }  
  
  // Grow density-based cluster from each core  
   $k = 0$ ;  
  for ( $\mathbf{x} \in Cores$ ) {  
    if ( $id(\mathbf{x}) == \emptyset$ ) {  
       $k++$ ;  
       $id(\mathbf{x}) = k$ ;  
      densityConnected( $\mathbf{x}, k$ );  
    }  
  }  
  
  // Determine clustering, border points, and noise  
   $\mathcal{C} = \emptyset$ ;  
  for ( $i = 1 \dots k$ )  $\mathcal{C} = \mathcal{C} \cup \{\{\mathbf{x} \in \mathcal{D} : id(\mathbf{x}) = i\}\}$ ;  
  
   $Noise = \{\mathbf{x} \in \mathcal{D} : id(\mathbf{x}) = \emptyset\}$ ;  
   $Border = \mathcal{D} \setminus \{Cores \cup Noise\}$ ;  
}
```

```
densityConnected( $\mathbf{x}, k$ ) {  
  for ( $\mathbf{y} \in N_\epsilon(\mathbf{x})$ ) {  
     $id(\mathbf{y}) = k$ ;  
    if ( $\mathbf{y} \in Cores$ ) densityConnected( $\mathbf{y}, k$ );  
  }  
}
```

DBSCAN in Action



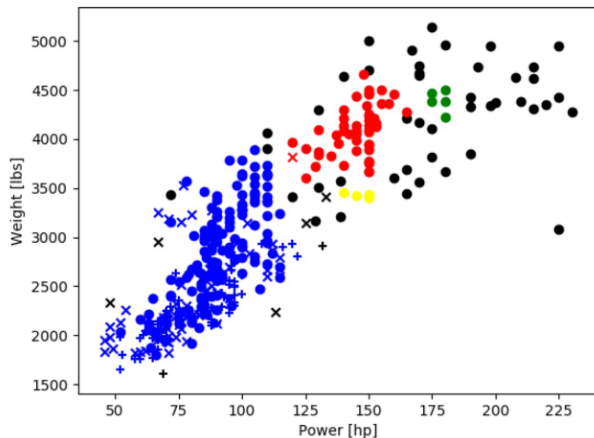
Clustering Cars based on Power and Weight

demo/DBSCAN_example.py

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import DBSCAN
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt

# load data
cars = pd.read_csv('auto-mpg.data.txt', header=None, sep='\s+')
# extract power and weight as data matrix X
X = cars.iloc[:, [3,4]].values
# extract origin as target value y
y = cars.iloc[:, 7].values
# normalize data
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X) # determine min and max
X_normalized = min_max_scaler.transform(X)
# DBSCAN
db = DBSCAN(eps=0.05, min_samples=5, metric='euclidean')
db.fit_predict(X_normalized)
# plot cars
# U.S. : o / Europe: x / Japan : +
m = ['o' if o==1 else 'x' if o==2 else '+' for o in y]
# Noise : black / Cluster 1 : red / Cluster 2 : blue /
# Cluster 3 : green / Cluster 4 : yellow
c = ['black' if l==-1 else 'red' if l==0 else 'blue' if l==1
     else 'green' if l==2 else 'yellow' for l in db.labels_]
for i in range(0, len(X)):
    plt.scatter(X[i,0], X[i,1], color=c[i], marker=m[i])
plt.xlabel('Power [hp]')
plt.ylabel('Weight [lbs]')
plt.show()
```

Clustering Cars based on Power and Weight



Summary

- ▶ Hierarchical clustering determines a sequence of clusterings that can be visualized in a dendrogram
- ▶ DBSCAN as a density-based clustering method can find non-convex clusters and is able to label data points as noise
- ▶ DBSCAN comes with two hyper parameters ε and minpts that need to be carefully tuned based on the data

