Machine Learning

FIB, Master in Innovation and Research in Informatics

Marta Arias, Computer Science @ UPC

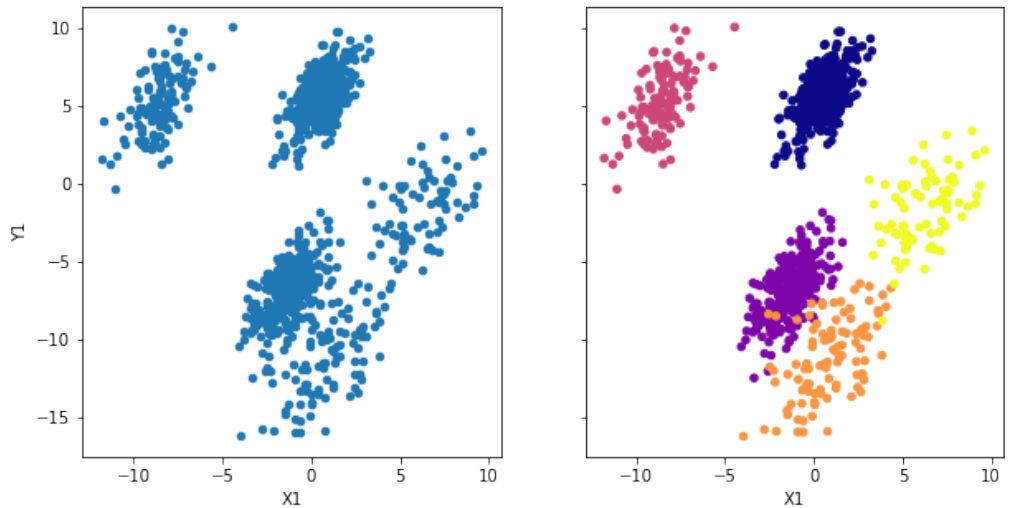
Lecture 3: Probabilistic clustering

Outline

1. What is clustering
2. K-means clustering
3. Mixture of Gaussians (MoG)
4. Expectation Maximization for learning MoG

What is clustering

The goal of **clustering** is to partition a data sample into groups ("clusters") in such a way that observations in the same cluster tend to be more similar than observations in different clusters



Input data is embedded in a d-dimensional space with a similarity/dissimilarity function defined among elements in the space, which should capture relatedness among elements in this space :

- elements are more related to *closer* elements than to *far away* elements
- a "cluster" is a compact group that is separated from other groups or elements outside the cluster

What is clustering

There is no consensus[1] on how to measure in a concise way (mathematically) these ideas, and different algorithms capture them in their own way. Thus, there is a large variety of clustering algorithms:

- **Hierarchical bottom-up/top-down**: single or average linkage, Ward, ...
- **Probabilistic** use MoGs (e.g. k-means and E-M)
- **Possibilistic** use fuzzy set memberships (e.g. fuzzy c-means)
- **Algorithmic** greedy/hill-climbing (swapping elements between clusters, e.g. PAM)
- **Spectral** use the spectrum (eigenvalues/vectors) of the data similarity matrix to perform dimensionality reduction before clustering in fewer dimensions
- **Density-based** finds connected dense regions in the data space (e.g. DBSCAN)

[1] Kleinberg's paper An Impossibility Theorem for Clustering argues that no consensus can exist.

Clustering is hard

- The number of ways one can partition a set of N elements into K groups is astronomical, it is given by the Stirling number of the second kind:

$$S(N,K) = \frac{1}{K!}\sum_{i=1}^{K}(-1)^i\binom{K}{i}(K-i)^N$$

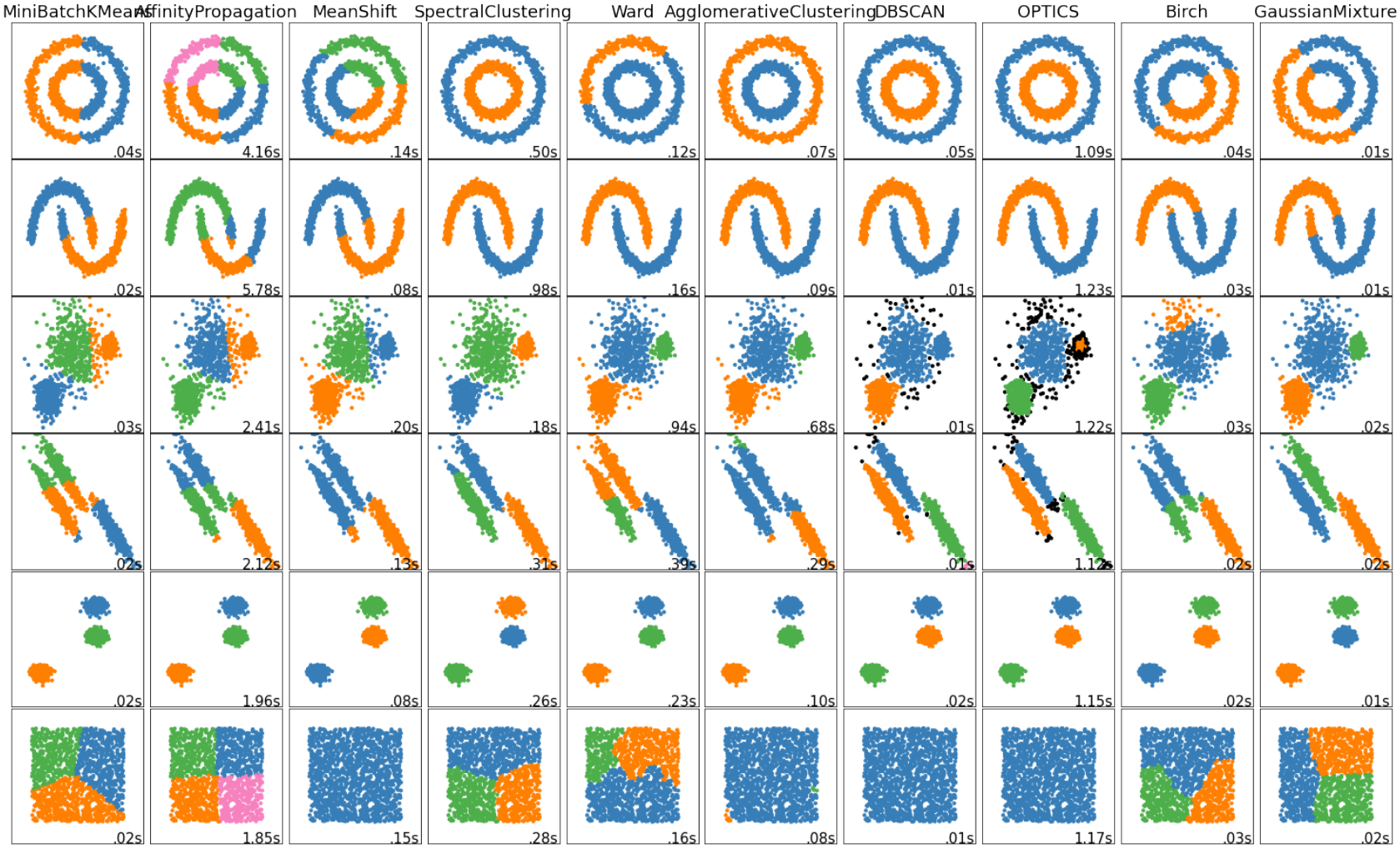
- Adding over all possible $K = 1,\ldots,N$, we get that the number of ways to partition a set of N elements is given by the N-th Bell number:

$$B(N) = \sum_{K=1}^{N} S(N,K)$$

To get a sense of these numbers:

$$S(10,4) = 35.105 \qquad S(19,4) \approx 10^{10} \qquad B(71) \approx 4\times 10^{71}$$

Many algorithms for many situations..



In this lecture, we'll cover

- K-means clustering
- Expectation-Maximization for Gaussian mixture models

Clustering with k-means

K-means

Input:

- a dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with $\mathbf{x}_i \in \mathbb{R}^d$, and
- an integer $K > 1$ (a hyper-parameter) denoting the number of desired clusters

Main intuition:

- each cluster represented by a **cluster center** $\mu_k \in \mathbb{R}^d$ for $k = 1, \ldots, K$
- each point should be closest to its assigned cluster center

In a "good cluster" all of its points should be close to its assigned cluster center, and so we would expect

$$\sum_{\substack{i:\mathbf{x}_i \text{ is assigned} \\ \text{to cluster } k}} \|\mathbf{x}_i - \mu_k\|^2$$

to be small, where $\|\mathbf{x}_i - \mu_k\|^2$ is the Euclidean distance between $\mathbf{x}_i$ and cluster center μ_k.

K-means

Cost function to optimize

We introduce a set of indicator variables:

$$r_{ik} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ is assigned to cluster } k \\ 0 & \text{otherwise} \end{cases}$$

And an objective (cost) function:

$$\mathcal{J}(\mu, \mathbf{r}) = \sum_{k=1}^{K} \sum_{i=1}^{n} r_{ik} \|\mathbf{x}_i - \mu_k\|^2$$

K-means

Optimizing the cost function $\mathcal{J}$

The goal is to find cluter centers $\{\mu_k\}_k$ and assignments $\{r_{ik}\}_{ik}$ that minimize

$$\mathcal{J}(\mu, \mathbf{r}) = \sum_{k=1}^{K} \sum_{i=1}^{n} r_{ik} \|\mathbf{x}_i - \mu_k\|^2.$$

Unfortunately, this is an **NP-hard problem**. So we will use an alternative procedure to try to minimize it. It is only guaranteed to find **local minima**.

It is based on the fact that:

a) For fixed cluster centers μ_k, it is easy to optimize cluster assignments r_{ik}
b) For fixed cluster assignments r_{ik}, it is easy to optimize cluster centers μ_k

K-means

Optimizing the cost function $\mathcal{J}$ for fixed μ_k

Assume fixed cluster centers μ_k for $k = 1, \ldots, K$.

The optimal way to assign points to clusters is by assigning each point to its nearest cluster center:

$$r_{ik} := \begin{cases} 1 & \text{if } k = \arg\min_{k'} \|\mathbf{x}_i - \mu_{k'}\|^2 \\ 0 & \text{otherwise} \end{cases}$$

To see this, it is enough to observe that if a point $\mathbf{x}_i$ is assigned to a cluster center μ_j instead of being assigned to its closest μ_k, then the cost function can be improved because its contribution to the cost is given by $\|\mathbf{x}_i - \mu_j\|^2 > \|\mathbf{x}_i - \mu_k\|^2$

K-means

Optimizing the cost function $\mathcal{J}$ for fixed r_{ik}

Assume fixed r_{ik} for $i = 1, \ldots, n$ and $k = 1, \ldots, K$.

Following usual procedure, we differentiate and equate to 0 in order to find the minimum.

$$
\begin{aligned}
\frac{\partial}{\partial \mu_j} \mathcal{J}(\mu_1, \ldots, \mu_K) &= \sum_{k=1}^{K} \sum_{i=1}^{n} \frac{\partial}{\partial \mu_j} r_{ik} \|\mathbf{x}_i - \mu_k\|^2 \\
&= \sum_{i=1}^{n} r_{ij} \frac{\partial}{\partial \mu_j} (\mathbf{x}_i - \mu_j)^T (\mathbf{x}_i - \mu_j) \\
&= \sum_{i=1}^{n} r_{ij} \frac{\partial}{\partial \mu_j} \left\{ \mathbf{x}_i^T \mathbf{x}_i - 2\mu_j^T \mathbf{x}_i + \mu_j^T \mu_j \right\} \\
&= \sum_{i=1}^{n} r_{ij} \left\{ -2\mathbf{x}_i + 2\mu_j \right\} \\
&= -2 \sum_{i=1}^{n} r_{ij} \mathbf{x}_i + 2\mu_j \sum_{i=1}^{n} r_{ij}
\end{aligned}
$$

K-means

Optimizing the cost function $\mathcal{J}$ for fixed r_{ik}

$$\frac{\partial}{\partial \mu_j} \mathcal{J}(\mu_1, \ldots, \mu_K) = -2 \sum_{i=1}^{n} r_{ij} \mathbf{x}_i + 2\mu_j \sum_{i=1}^{n} r_{ij}$$

Thus, the minimum is obtained when

$$\mu_j = \frac{\sum_i r_{ij} \mathbf{x}_i}{\sum_i r_{ij}} = \frac{1}{n_j} \sum_{\substack{i:\mathbf{x}_i \text{ is assigned} \\ \text{to cluster } j}} \mathbf{x}_i$$

where $n_j = \sum_i r_{ij}$ is the number of points assigned to cluster j.

The optimal cluster center is given by the **centroid** (average) of all points assigned to it.

K-means pseudocode

1. Initialize cluster centers $\mu_1, \ldots, \mu_K$
2. **repeat until convergence**
   - assign each point to the cluster with closest center
   - recompute cluster centers as $\mu_k := \frac{1}{n_k} \sum_{r_{ik}=1} \mathbf{x}_i$ for all $k = 1, \ldots, K$

K-means

Advantages:

- easy to implement
- fast, can be run many times even on large datasets

Limitations:

- converges to **local minimum**
- needs number of clusters K as input
- **hard** cluster assignments
- sensitive to outliers and clusters of different sizes and densities
- very sensitive to initialization (so run it many times, and keep the best)

K-means++

k-means++ is a variant of k-means that uses a heuristic for initializing cluster centers as follows:

1. Choose first center μ_1 uniformly at random from all available examples
2. For $k = 2, \ldots, K$
   - Choose next center μ_k at random, where a point is chosen with probability proportional to $\|\mathbf{x}_i - \mu_l\|$, where μ_l is its closest cluster center picked so far (among $\mu_1, \ldots, \mu_{k-1}$)
3. Proceed with standard k-means

Choosing the number of clusters K

The number of clusters is a hyper-parameter that has to be set by the user; unfortunately there is no obvious way to choose an *optimal* K, since oftentimes such optimal K does not exist, or is not unique, or there is no way to know.

The undelying difficulty is that in clustering there is no **true clustering** (known or unknown) so there is nothing we can compare against.

Despite this, there are many reasonable cluster quality criteria that can be used for selecting K.

These criteria measure a balance between separation of clusters and their comptactness; there is no measure that works for all datasets, and it is up to the preferences of the analyst and/or practical considerations of the problem at hand that one or another is used.

Popular ones are the **Calinski-Harabasz index**, the silhouette index, or the Davis-Bouldin score, but many others exist.[2]

[2] This paper contains an empirical comparison of many existing indices for cluster evaluation.\

The Calinski-Harabasz index

The **CH index** uses, like k-means, Euclidean distances to measure cluster quality and so it is very much used together with k-means.

It measures the ratio between separation of cluster centers (sum of distances of cluster centers to overall mean - in the numerator) and cluster compactness (sum of distances from each point to its assigned cluster center - in the denominator):

$$\frac{(N-K)}{(K-1)} \frac{\sum_{k=1}^{K} n_k \|\mu_k - \bar{\mathbf{x}}\|^2}{\sum_{k=1}^{K} \sum_{i=1}^{n} r_{ik} \|\mathbf{x}_i - \mu_k\|^2}$$

where $\bar{\mathbf{x}}$ is the overall average of points in the dataset, i.e. $\bar{\mathbf{x}} = \frac{1}{n} \sum_i \mathbf{x}_i$.

The quantities are normalized by $\frac{(N-K)}{(K-1)}$ in order to avoid larger K having better values.

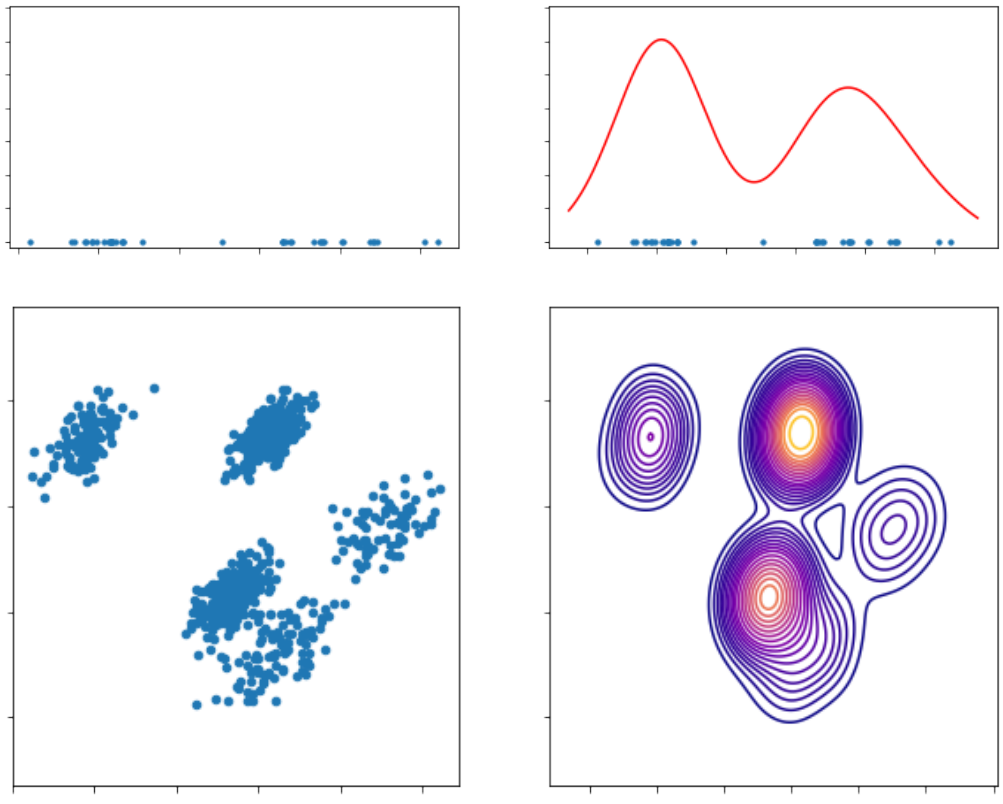
Typically, we will run k-means for different values of k, and will select the k that maximizes this index.

Learning Gaussian mixtures with Expectation Maximization

Mixture of Gaussians

A way of modelling unknown density

When we have data that are clearly not Gaussian, it may be a useful choice to describe the data given:



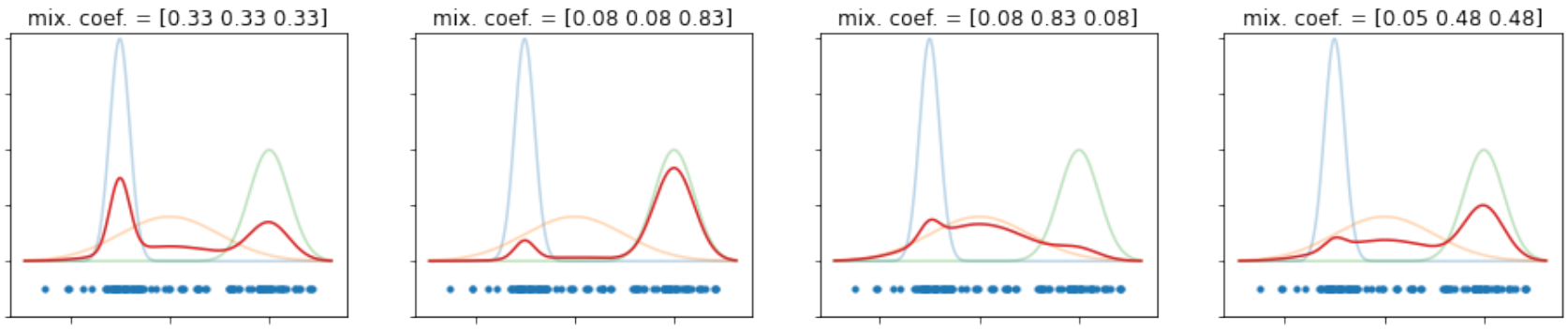
Mixture of Gaussians, cont.

A mixture of Gaussians is a distribution that is built using a **convex sum of Gaussians**; so it is more flexible than a single Gaussian.

$$p(\mathbf{x}|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$$

- Each $\mathcal{N}(\mu_k, \Sigma_k)$ is a **component** of the mixture (Gaussian, with parameters μ_k and Σ_k)
- The π_k are the **mixing coefficients** of each component, such that $0 \leq \pi_k \leq 1$, and $\sum_k \pi_k = 1$
- The parameters of this distribution are $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1,\ldots,K}$

Mixture of Gaussians, cont.



Clustering with a Gaussian mixture

One cluster == one component of mixture

So, to cluster data $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ into K clusters:

- use EM to estimate mixture, results in $\hat{\pi}_k, \hat{\mu}_k, \hat{\Sigma}_k$ for each $k = 1, \ldots, K$
- find assignments[3] of each $\mathbf{x}_i$ to clusters

[3] As we will see, these assignments under the new probabilistic model are going to be **soft**. More on this when we introduce EM.

Mixture of Gaussians, cont.

Generative model perspective

To sample from such a mixture, we can think of the following **generative model**; it uses a **latent** (unobserved) variable $\mathbf{z} = (z_1, \ldots, z_K)$ whose components are all 0 except one which denotes the component from which we will sample

1. Pick **component** k with probability π_k (that is, $z_k = 1$ w.p. π_k)
2. Generate sample $\mathbf{x}$ according to $\mathcal{N}(\mu_k, \Sigma_k)$

The probability of generating a sample $\mathbf{x}$ using this generative model is

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_k p(\mathbf{x}, z_k = 1) = \sum_k p(\mathbf{x}|z_k = 1)p(z_k = 1) = \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$$

The joint distribution of $\mathbf{x}$ and $\mathbf{z}$ is thus given by (notational trick):

$$p(\mathbf{x}, \mathbf{z}) = \prod_k \pi_k^{z_k} \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)^{z_k}$$

Mixture of Gaussians, cont.

So far, we have:

- **joint distribution** $p(\mathbf{x}, \mathbf{z}) = \prod_k \pi_k^{z_k} \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)^{z_k}$
- **marginal distribution over x** $p(\mathbf{x}) = \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$
- **marginal distribution over z** $p(z_k = 1) = \pi_k$ for all $k = 1, \ldots, K$ so $p(\mathbf{z}) = \prod_k \pi_k^{z_k}$
- **conditional distribution of x given z** $p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$

Using Bayes, we compute the **conditional distribution of z given x**:

$$\begin{aligned} p(z_k = 1|\mathbf{x}) &= \frac{p(\mathbf{x}|z_k = 1)p(z_k = 1)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x}|z_k = 1)p(z_k = 1)}{\sum_{k'} \pi_{k'} \mathcal{N}(\mathbf{x}; \mu_{k'}, \Sigma_{k'})} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(\mathbf{x}; \mu_{k'}, \Sigma_{k'})} \\ &=: \gamma_k(\mathbf{x}) \end{aligned}$$

Mixture of Gaussians, cont.

Soft assignments

The quantity $\gamma_k(\mathbf{x}) := p(z_k = 1|\mathbf{x})$ states how probable it is that a particular data point $\mathbf{x}$ has been generated by mixture component k. Or, in the context of clustering, how probable it is that $\mathbf{x}$ **belongs to cluster** k.

We use these quantities as **soft membership** to each cluster. If you want a **hard membership** then $\mathbf{x}$ should be assigned to $k = \arg\max_{k'} \gamma_{k'}(\mathbf{x})$. But in many contexts having soft memberships is desirable and certainly more flexible.

Learning Gaussian mixtures with EM

Maximum likelihood

We are given an iid sample of unlabelled observations $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with each $\mathbf{x}_1 \in \mathbb{R}^d$. We want to model this sample as a Gaussian mixture. The unknown parameters are $\theta = \{\pi_k, \mu_k, \Sigma_k\}_k$; K is assumed fixed and given as input.

$$
\begin{aligned}
l(\theta) = \log \mathcal{L}(\theta) &= \log \prod_{i=1}^{n} p(\mathbf{x}; \theta) \\
&= \log \prod_i \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) \\
&= \sum_i \log \left\{ \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) \right\}
\end{aligned}
$$

This is hard to optimize; the log-likelihood surface is complex with many local maxima.

Learning Gaussian mixtures with EM, cont.

Maximum likelihood estimate for μ_k

$$l(\theta) = \sum_i \log \left\{ \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) \right\}$$

Nevertheless, we can compute some partial derivatives to see what conditions should hold in any local maximum.

- $\frac{\partial l(\theta)}{\partial \mu_k} = 0$ leads to

$$\hat{\mu}_k = \frac{\sum_i \gamma_k(\mathbf{x}_i)\mathbf{x}_i}{\sum_i \gamma_k(\mathbf{x}_i)} = \frac{\sum_i P(z_k = 1|\mathbf{x}_i)\mathbf{x}_i}{\sum_i P(z_k = 1|\mathbf{x}_i)}$$

which is a **weighted average** of the points in our data, with weights being the **soft assignments** of each point to cluster k.

Problem: we do not know $\gamma_k(\mathbf{x})$ without μ_k, Σ_k, π_k.

Learning Gaussian mixtures with EM, cont.

Maximum likelihood estimate for Σ_k

$$l(\theta) = \sum_i \log \left\{ \sum_k \pi_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) \right\}$$

- $\frac{\partial l(\theta)}{\partial \Sigma_k} = 0$ leads to

$$\hat{\Sigma}_k = \frac{\sum_i \gamma_k(\mathbf{x}_i)(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T}{\sum_i \gamma_k(\mathbf{x}_i)} = \frac{\sum_i P(z_k = 1|\mathbf{x}_i)(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T}{\sum_i P(z_k = 1|\mathbf{x}_i)}$$

which is the **sample covariance matrix** of all $\mathbf{x}_i$ weighted by the **soft assignments** of ech point to cluster k (i.e., weighted by the posterior probability that component k generated $\mathbf{x}_i$)

Problem: we do not know $\gamma_k(\mathbf{x})$ without μ_k, Σ_k, π_k.

Learning Gaussian mixtures with EM, cont.

Maximum likelihood estimate for π_k

Now we maximize the **Lagrangian** $l(\theta) - \lambda(\sum_k \pi_k - 1)$ since we have an equality constraint on the π_k

- $\frac{\partial l(\theta)}{\partial \Sigma_k} = 0$ and $\sum_k \pi_k = 1$ lead to

$$\hat{\pi}_k = \frac{1}{n} \sum_i \gamma_k(\mathbf{x}_i)$$

which is the **average** of all **soft assignments** for all data point $\mathbf{x}_i$.

Problem: we do not know $\gamma_k(\mathbf{x})$ without μ_k, Σ_k, π_k.

Learning Gaussian mixtures with EM, cont.

- We can estimate π_k, Σ_k, μ_k if we know $\gamma_k(\cdot)$
- We can compute $\gamma_k(\cdot)$ from estimates $\hat{\pi}_k, \hat{\Sigma}_k, \hat{\mu}_k$

Learning Gaussian mixtures with EM, cont.

Pseudocode

1. Initialize $\hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k$
2. **repeat until convergence**
   - **E-step** recompute soft assignments $\gamma_k(\mathbf{x}_i)$

$$\gamma_k(\mathbf{x}_i) = \frac{\pi_k \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(\mathbf{x}_i; \mu_{k'}, \Sigma_{k'})}$$

   - **M-step** recompute ML estimates

$$\hat{\mu}_k = \frac{\sum_i \gamma_k(\mathbf{x}_i)\mathbf{x}_i}{\sum_i \gamma_k(\mathbf{x}_i)}$$

$$\hat{\Sigma}_k = \frac{\sum_i \gamma_k(\mathbf{x}_i)(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T}{\sum_i \gamma_k(\mathbf{x}_i)}$$

$$\hat{\pi}_k = \frac{1}{n} \sum_i \gamma_k(\mathbf{x}_i)$$

Learning Gaussian mixtures with EM, cont.

Initialization

Commonly we initialize $\hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k$ using the result of k-means:

1. Run k-means with $k = K$ (maybe a few times, pick best)
2. Set $\hat{\mu}_k$ to k-mean's cluster centers
3. Set each $\hat{\Sigma}_k$ to the sample covariance of each cluster of k-means
4. Set $\hat{\pi}_k$ as the fraction of examples assigned to cluster k

EM, special cases

We can restrict the **shape** of Gaussians for each component, which results in special cases of mixtures:

- No restriction on Σ_k; this is the general case (most flexible); each cluster can have general Gaussian shape
- Σ_k are **diagonal**; each Gaussian component is forced to have no correlation among input dimensions (i.e. axis-aligned)
- $\Sigma_k = \sigma^2 I$ are **isotropic** or **spherical**; each Gaussian component is forced to be spherical, so no correlation among input variables and same scaling accross each input variable.

  In fact, k-means is a degenerate case of this scenario: if $\sigma^2 \to 0$, then $\gamma_k(\mathbf{x}_i) \to r_{ik}$