Gaussian Mixture Models

Sandro Cumani

sandro.cumani@polito.it

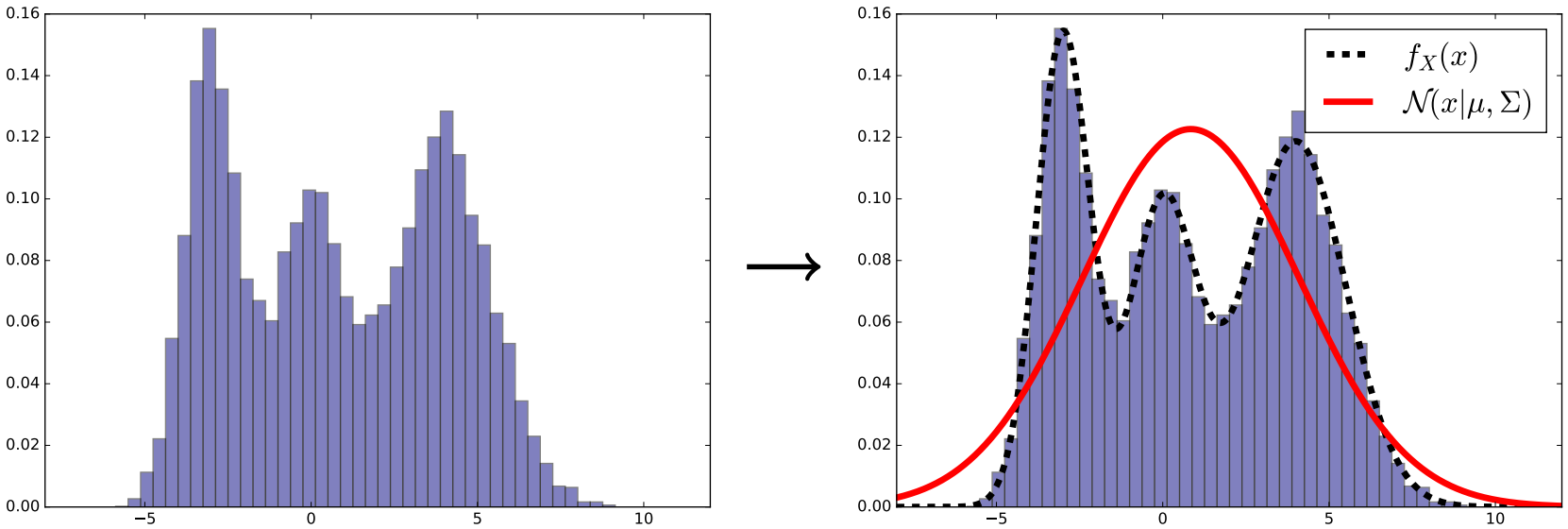
Politecnico di Torino

Gaussian Mixture models

We have seen that we can solve classification problems by building generative models that describe the distribution of samples

The Gaussian classifier is an example that assumes that class-conditional distributions are Gaussian

In many cases, however, the assumption can be quite inaccurate



Density estimation

Different distributions may be used in such cases

Depending on the task, we may be able to identify a reasonably good family of distributions

Gaussian Mixture Models are an alternative to model a generic distribution

They allow approximating any sufficiently regular distribution to a desired degree

Of course, since we are estimating the density from data, we require a sufficient amount of data to obtain good estimates

Density estimation

The use of GMMs is not restricted to classification

GMMs can be employed also in other tasks that require estimating a population density

As we will see, they also allow to solve different kind of problems

For example, GMMs provide an alternative to K-means for clustering

Gaussian Mixture Models

We have already encountered an example of GMM

Let's consider again the Gaussian classifier

The samples of each class are modeled by a Gaussian density

$$f_{X|C}(\boldsymbol{x}|c) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

To compute class posterior probabilities we had to compute the marginal density $f_X(\boldsymbol{x})$

If the class prior probabilities are $P(C = c) = \pi_c, c = 1 \dots K$, then $f_X(\boldsymbol{x})$ is given by

$$f_X(\boldsymbol{x}) = \sum_{c=1}^{K} f_{X|C}(\boldsymbol{x}|c)P(C = c) = \sum_{c=1}^{K} \pi_c \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \qquad (1)$$

Gaussian Mixture Models

Expression (1) is an example of a K-components Gaussian Mixture Model:

$$\boldsymbol{X} \sim GMM(\boldsymbol{M}, \boldsymbol{\Sigma}, \boldsymbol{\Pi})$$

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \sum_{c=1}^{K} \pi_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

More in general, a Gaussian Mixture Model is a density model obtained as a weighted combination of Gaussians

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

The distribution parameters are the component means

$$\boldsymbol{M} = [\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_K]$$

the component covariances

$$\boldsymbol{\mathcal{S}} = [\boldsymbol{\Sigma}_1 \dots \boldsymbol{\Sigma}_K]$$

and the weights

$$\boldsymbol{w} = [w_1 \dots w_K]$$

Remember that, for f_X to be a density, we need that its integral is equal to 1. Integrating w.r.t. $\boldsymbol{x}$ we have:

$$\int f_X(\boldsymbol{x}) = \int \sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) d\boldsymbol{x} = \sum_{c=1}^{K} w_c = 1$$

i.e., the weights must sum to 1

Gaussian Mixture Models

Given a dataset $\mathcal{D} = [\boldsymbol{x}_1 \dots \boldsymbol{x}_n]$ we can thus assume that the samples have been independently generated by a GMM

We assume that R.V.s describing the samples $\boldsymbol{X}_i$ are i.i.d., with $\boldsymbol{X}_i \sim \boldsymbol{X} \sim GMM(\boldsymbol{M}, \boldsymbol{\mathcal{S}}, \boldsymbol{w})$

Note: we are considering a density estimation problem

If we are using GMMs for classification $\mathcal{D}$ may correspond to the samples of a given class — however, in the following we do not assume any specific task, so that $\mathcal{D}$ is just a set of samples that we want to model by means of a GMM

In particular, we consider the dataset $\mathcal{D}$ as unlabeled

Gaussian Mixture Models

As we did with the Gaussian density, we can resort to Maximum Likelihood to estimate the model parameters of the GMM that best describes the dataset $\mathcal{D}$

In contrast with the Gaussian model, ML estimation for GMMs is an ill-posed problem

Indeed, as long as we have more than 1 component, we can devise degenerate solutions for which the likelihood is not bounded above

Care has to be taken to avoid these pathological solutions

In practice, the ML approach, combined with heuristics to avoid degeneracy, provides good density estimates

Gaussian Mixture Models

We can write the likelihood for the model parameters $\boldsymbol{\theta} = [\boldsymbol{M}, \boldsymbol{\mathcal{S}}, \boldsymbol{w}]$ as

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{n} f_{\boldsymbol{X}_i}(\boldsymbol{x}_i) = \prod_{i=1}^{n} GMM(\boldsymbol{x}_i | \boldsymbol{M}, \boldsymbol{\mathcal{S}}, \boldsymbol{w})$$
$$= \prod_{i=1}^{n} \left(\sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}_i | \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right)$$

and the corresponding log-likelihood

$$\ell(\boldsymbol{\theta}) = \log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \left(\sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}_i | \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right)$$

Gaussian Mixture Models

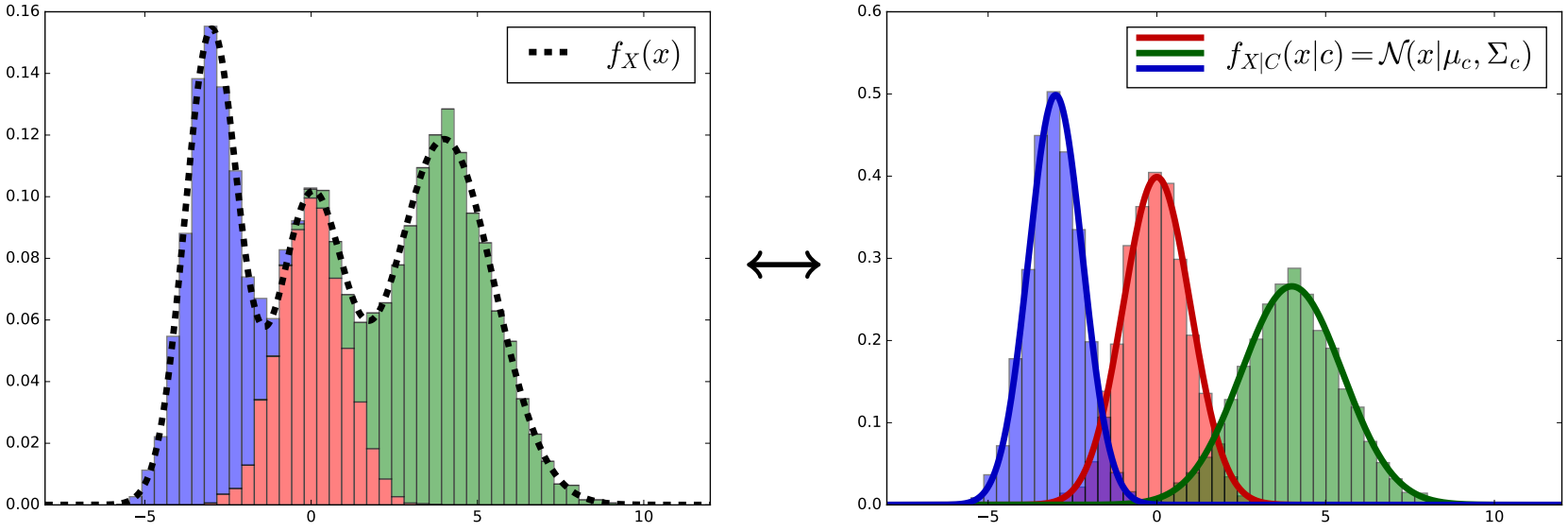
A GMM can be interpreted as the marginal of a joint distribution of data points and corresponding clusters

$$f_{X_i}(\boldsymbol{x}_i) = \sum_{c=1}^{K} f_{X_i|C_i}(\boldsymbol{x}_i|c)P(C_i = c) = \sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

Although our training set cannot be well modeled by a Gaussian distribution, we can imagine that the set can be partitioned into subsets (components or clusters), in such a way that the distribution of the points of each component can be modeled by a Gaussian p.d.f.

If we knew the component responsible for each sample (i.e. its cluster label), we could estimate the parameters of each Gaussian by ML from the points of each cluster

Gaussian Mixture Models



Gaussian Mixture Models

Unfortunately, in general the clusters are unknown

We treat cluster membership as an unobserved (latent) random variables[1]

Intuitively, we want to estimate both cluster assignments and model parameters as to maximize the *marginal* distribution of the data

[1] Note that the model is not identifiable: for example, exchanging any two components results in the same marginal likelihood

Gaussian Mixture Models

Let's consider a set of GMM parameters $\boldsymbol{\theta} = (\boldsymbol{M}, \boldsymbol{\mathcal{S}}, \boldsymbol{w})$

The GMM defines a joint density of components (the clusters) and patterns. The density for sample $\boldsymbol{x}_i$ and component c is:

$$f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c) = w_c \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

We can compute cluster (component) posterior probabilities:

$$\gamma_{c,i} = P(C_i = c | \boldsymbol{X}_i = \boldsymbol{x}_i) = \frac{f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c)}{f_{\boldsymbol{X}_i}(\boldsymbol{x}_i)}$$

$$= \frac{w_c \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{c'} w_{c'} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{c'}, \boldsymbol{\Sigma}_{c'})}$$

$\gamma_{c,i}$ are also called responsibilities

Gaussian Mixture Models

As a first approximation, we might decide to assign a point to the cluster c with highest posterior probability $P(C_i = c|\boldsymbol{X}_i = \boldsymbol{x}_i)$

We thus associate a cluster label

$$c_i^* = \arg\max_c P(C_i = c|\boldsymbol{X}_i = \boldsymbol{x}_i)$$

to each sample

Given the cluster assignments, we can then estimate by ML the new GMM parameters $\boldsymbol{\theta}^{new} = (\boldsymbol{M}^{new}, \boldsymbol{\mathcal{S}}^{new}, \boldsymbol{w}^{new})$

Gaussian Mixture Models

We treat the cluster assignments as if they were known class labels

The log-likelihood is similar to that of a (multivariate) Gaussian classifier:

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[\log f_{\boldsymbol{X}_i|C_i}(\boldsymbol{x}_i|c_i^*) + \log P(C_i = c_i^*)\right]$$
$$= \sum_{i=1}^{n} \left[\log \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_{c_i^*}, \boldsymbol{\Sigma}_{c_i^*})\right] + \sum_{i=1}^{n} \left[\log w_{c_i^*}\right]$$

and corresponds to a sum of two terms that depend on different subsets of the parameters

$$\ell(\boldsymbol{\theta}) = \ell_{\mathcal{N}}(\boldsymbol{M}, \boldsymbol{\mathcal{S}}) + \ell_{\mathcal{C}}(\boldsymbol{w})$$

We can observe that the first term

$$\ell_{\mathcal{N}}(\boldsymbol{M}, \boldsymbol{\mathcal{S}}) = \sum_{i=1}^{n} \left[\log \mathcal{N}(\boldsymbol{x}_i | \boldsymbol{\mu}_{c_i^*}, \boldsymbol{\Sigma}_{c_i^*}) \right]$$

corresponds to the log-likelihood of a (multivariate) Gaussian classification model, where the class labels are assumed to be the estimated c_i^*.

Let N_c be the number of samples for which $c_i^* = c$. The solution for $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ is thus

$$\boldsymbol{\mu}_c^* = \frac{1}{N_c} \sum_{i|c_i^*=c} \boldsymbol{x}_i \ , \quad \boldsymbol{\Sigma}_c^* = \frac{1}{N_c} \sum_{i|c_i^*=c} (\boldsymbol{x}_i - \boldsymbol{\mu}_c^*)(\boldsymbol{x}_i - \boldsymbol{\mu}_c^*)^T$$

The second term

$$\ell_{\mathcal{C}}(\boldsymbol{w}) = \sum_{i=1}^{n} \left[\log w_{c_i^*}\right] = \sum_{c=1}^{K} \sum_{i|c_i^*=c} \log w_c$$

corresponds to the log-likelihood of a categorical model with parameters w_c. The ML solution is thus

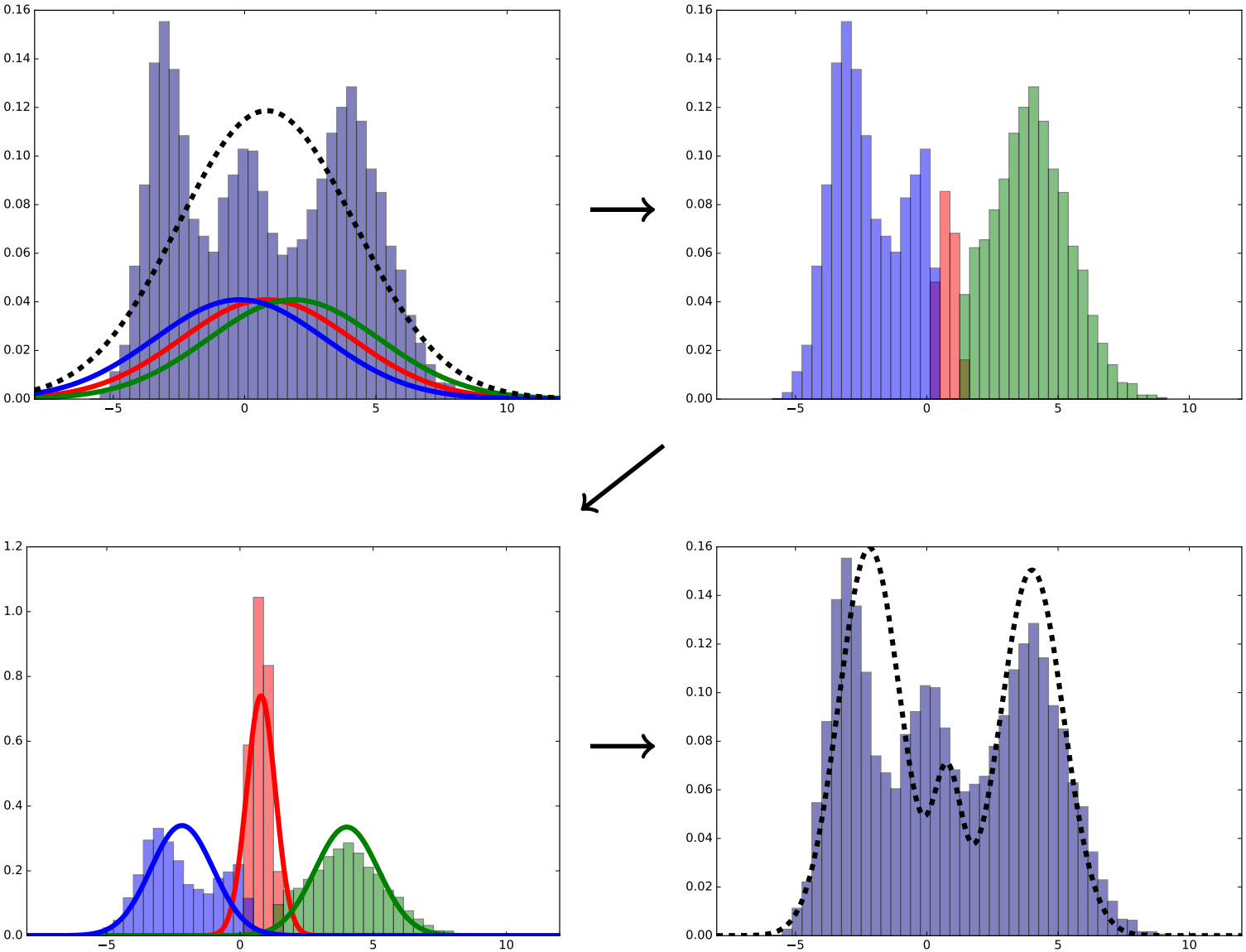
$$w_c^* = \frac{N_c}{\sum_{c=1}^{K} N_c}$$

Gaussian Mixture Models

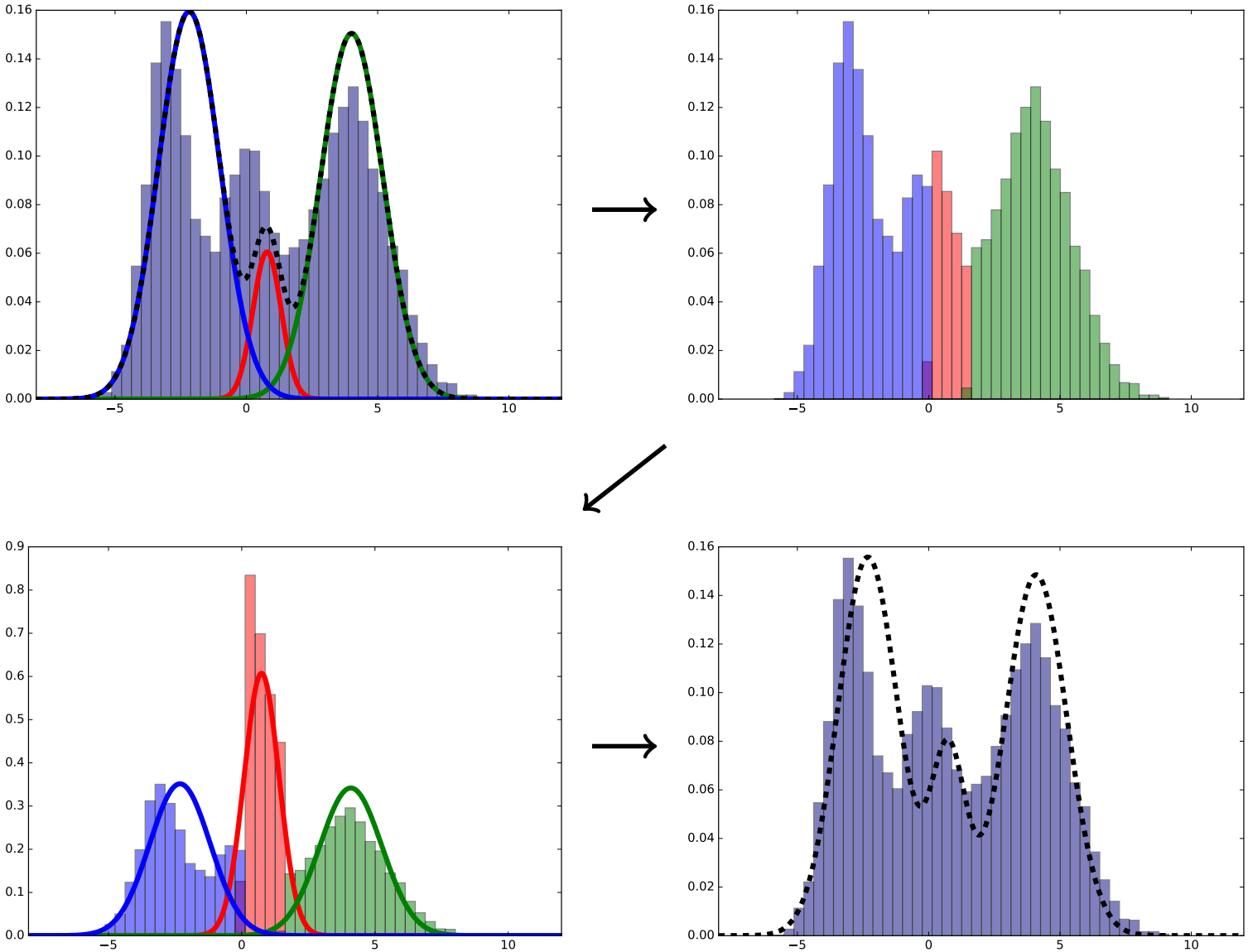
We could then obtain the updated set of model parameters $\boldsymbol{\theta}^{new} = (\boldsymbol{M}^*, \boldsymbol{\mathcal{S}}^*, \boldsymbol{w}^*)$

We could iterate the process by computing new cluster assignments using $\boldsymbol{\theta}^{new}$, and using the updated assignments to update once again the model parameters, stopping when some criterion is met

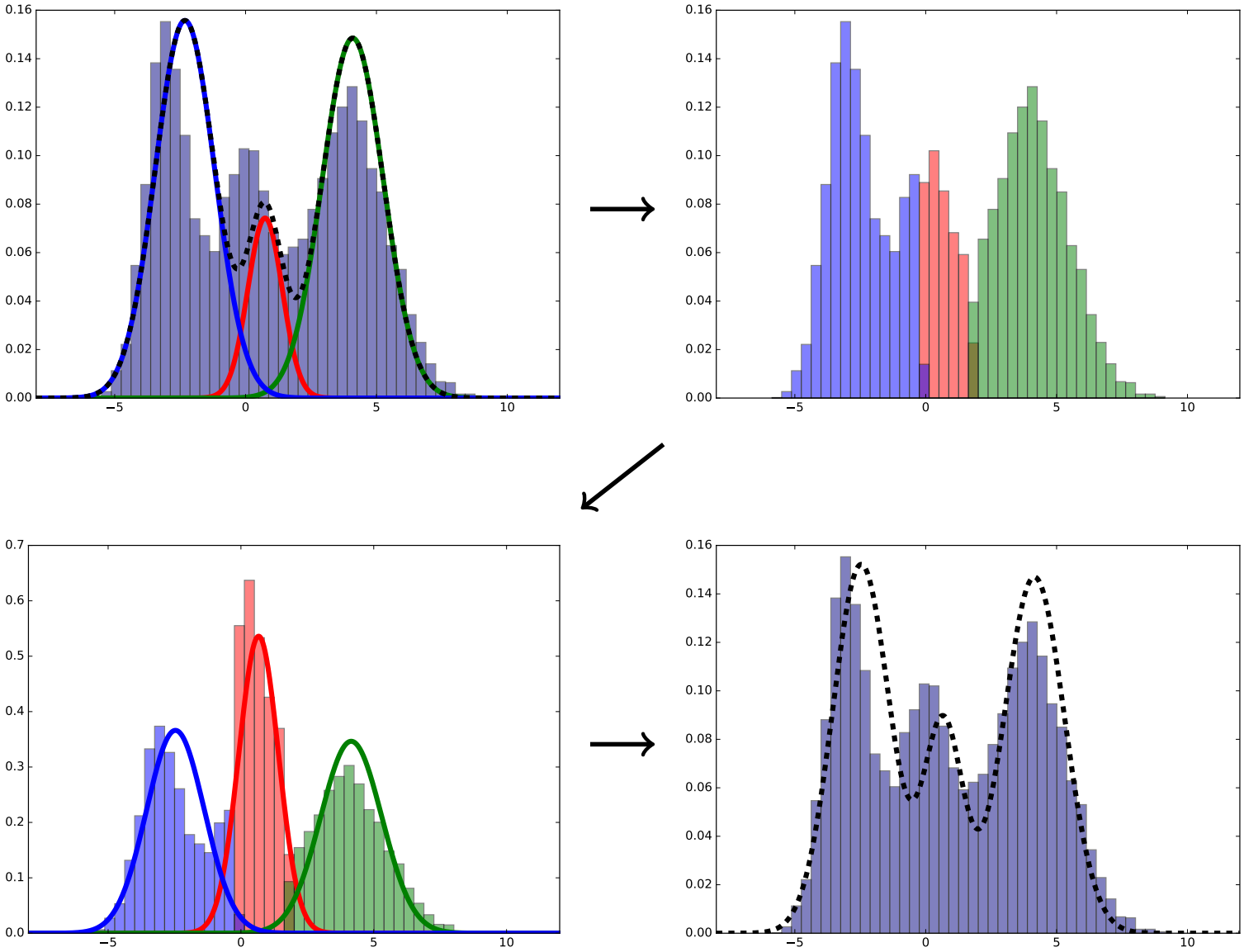
Gaussian Mixture Models



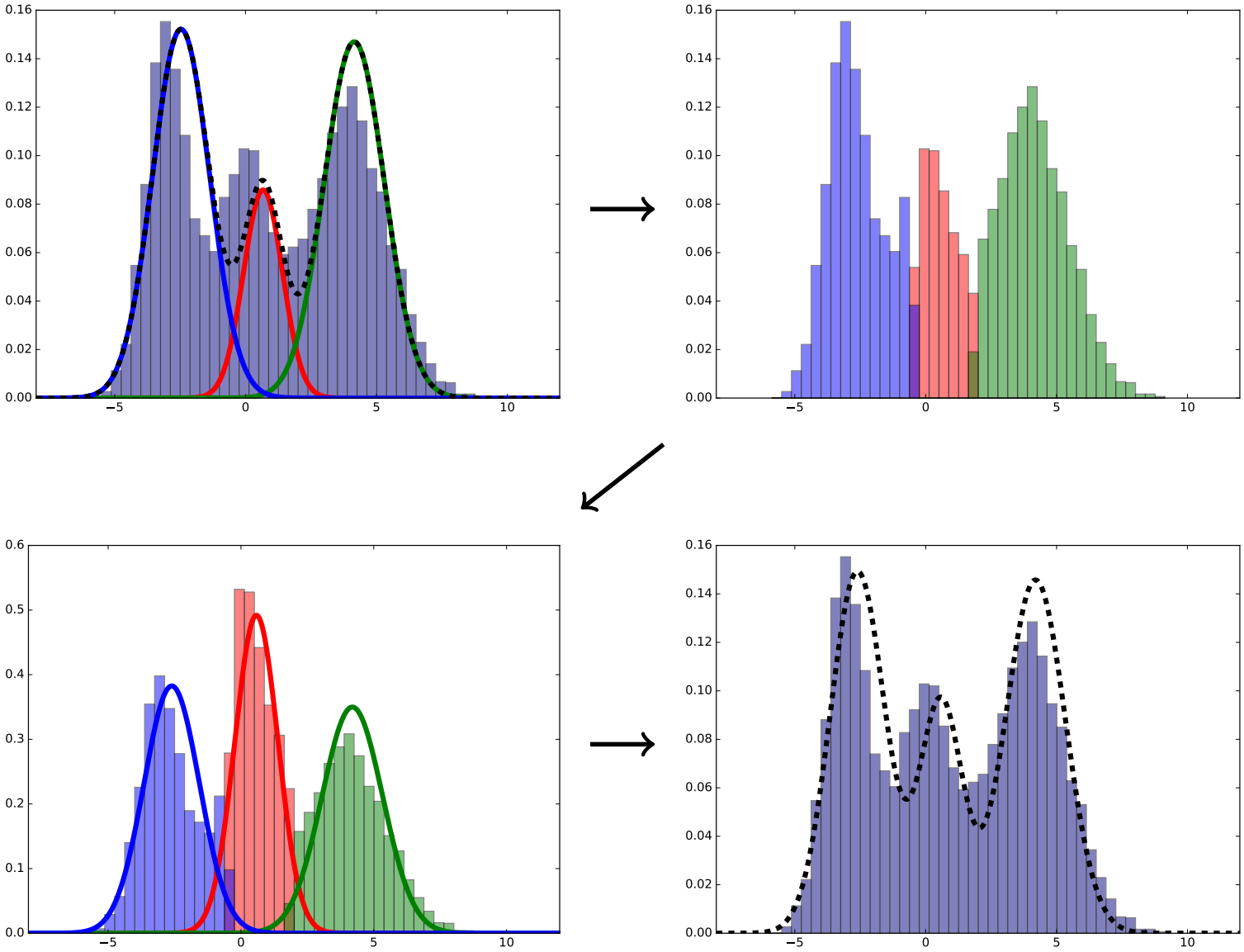
Gaussian Mixture Models



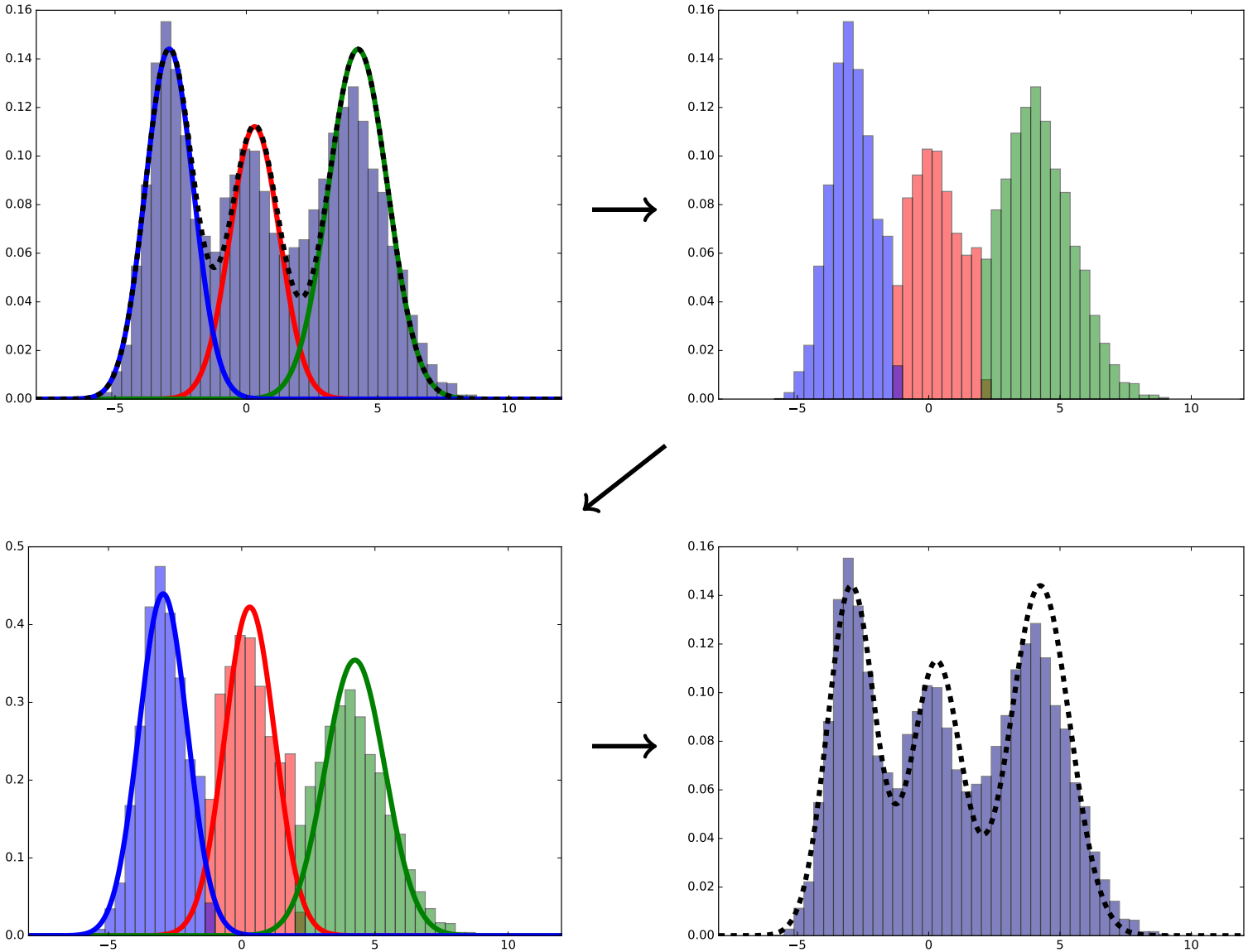
Gaussian Mixture Models



Gaussian Mixture Models



Gaussian Mixture Models



Gaussian Mixture Models

Problem: we form hard clusters — a point is assigned to one, and only one, component of the GMM

If $P(C_i = c_i^* | \boldsymbol{X}_i = \boldsymbol{x}_i) \approx 1$ then we can correctly assume that the point belongs to that component

However, when $P(C_i = c_1 | \boldsymbol{X}_i = \boldsymbol{x}_i) \approx P(C_i = c_2 | \boldsymbol{X}_i = \boldsymbol{x}_i)$ we are making a crude approximation: both c_1 and c_2 might have been responsible for the generation of $\boldsymbol{x}_i$

In general, the algorithm we discussed is not maximizing the likelihood of the observed samples $\boldsymbol{x}_i$

Gaussian Mixture Models

We will shortly see a method to estimate a local maximum of the likelihood

However, let's still consider hard-assignments

Let's also assume that we fix the covariance matrices of our GMM to $\boldsymbol{\Sigma}_c = \boldsymbol{I}$

We also fix the weights as $\boldsymbol{w}_c = \frac{1}{K}$

In this case cluster assignment corresponds to the rule

$$c_i^* = \arg\max_c P(C_i = c | \boldsymbol{X}_i = \boldsymbol{x}_i) = \arg\min_c \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|^2$$

Gaussian Mixture Models

Our algorithm becomes

- Compute the component or cluster c_i^* whose centroid $\boldsymbol{\mu}_{c_i^*}$ is closest to our point and assign $\boldsymbol{x}_i$ to that cluster
- Re-estimate the cluster centroids from the given points, and iterate until convergence

This is the K-Means clustering algorithm

GMMs can also be applied to clustering tasks as a generalization of K-Means

Gaussian Mixture Models

The algorithm we considered can be extended to handle soft assignments

We will see that a point is not completely associated to a single Gaussian component, but contributes to the estimation of different components according to its cluster (component) posterior probability

Consider the log-likelihood for our data (we now make explicitly the dependency on the model parameters in the conditional densities):

$$\sum_{i=1}^{n} \log f_{\boldsymbol{X}_i}(\boldsymbol{x}_i|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \left(\sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right)$$

Let's take the gradient with respect to $\boldsymbol{\mu}_c$

$$\begin{aligned} \mathbf{0} &= -\sum_i \frac{w_c \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{c'} w_{c'} \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_{c'}, \boldsymbol{\Sigma}_{c'})} \boldsymbol{\Sigma}_c (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \\ &= -\sum_i \gamma_{c,i} \boldsymbol{\Sigma}_c (\boldsymbol{x}_i - \boldsymbol{\mu}_c) \end{aligned} \tag{2}$$

which gives

$$\boldsymbol{\mu}_c = \frac{\sum_i \gamma_{c,i} \boldsymbol{x}_i}{\sum_i \gamma_{c,i}} \tag{3}$$

Notice that the responsibilities $\gamma_{c,i}$ depend on $\boldsymbol{\mu}_c$. If we knew the responsibilities we could compute $\boldsymbol{\mu}_c$ as in (3)

We can interpret (3) as a weighted empirical mean. The weight of each sample is the corresponding responsibility.

The terms

$$N_c = \sum_{i=1}^{N} \gamma_{c,i}$$

and

$$\boldsymbol{F}_c = \sum_{i=1}^{N} \gamma_{c,i} \boldsymbol{x}_i$$

are also called zero and first order statistics

Note that we are summing over all samples in $\mathcal{D}$

Gaussian Mixture Models

We can adopt a similar strategy for the covariance matrix, obtaining

$$\boldsymbol{\Sigma}_c = \frac{1}{N_c} \sum_i \gamma_{c,i} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_c\right) \left(\boldsymbol{x}_i - \boldsymbol{\mu}_c\right)^T = \frac{1}{N_c} \sum_i \gamma_{c,i} \boldsymbol{x}_i \boldsymbol{x}_i^T - \boldsymbol{\mu}_c \boldsymbol{\mu}_c^T$$

The terms

$$\boldsymbol{S}_c = \sum_i \gamma_{c,i} \boldsymbol{x}_i \boldsymbol{x}_i^T$$

are also called second order statistics

The weights can be re-estimated as

$$w_c = \frac{N_c}{N}$$

where N is the number of samples $N = \sum_{c=1}^{K} N_c$

Gaussian Mixture Models

Since we are not given $\gamma_{c,i}$, we can follow the same procedure we used for hard assignments:

- Given $\boldsymbol{\theta}$, we estimate the responsibilities, i.e. the cluster or component posterior probabilities

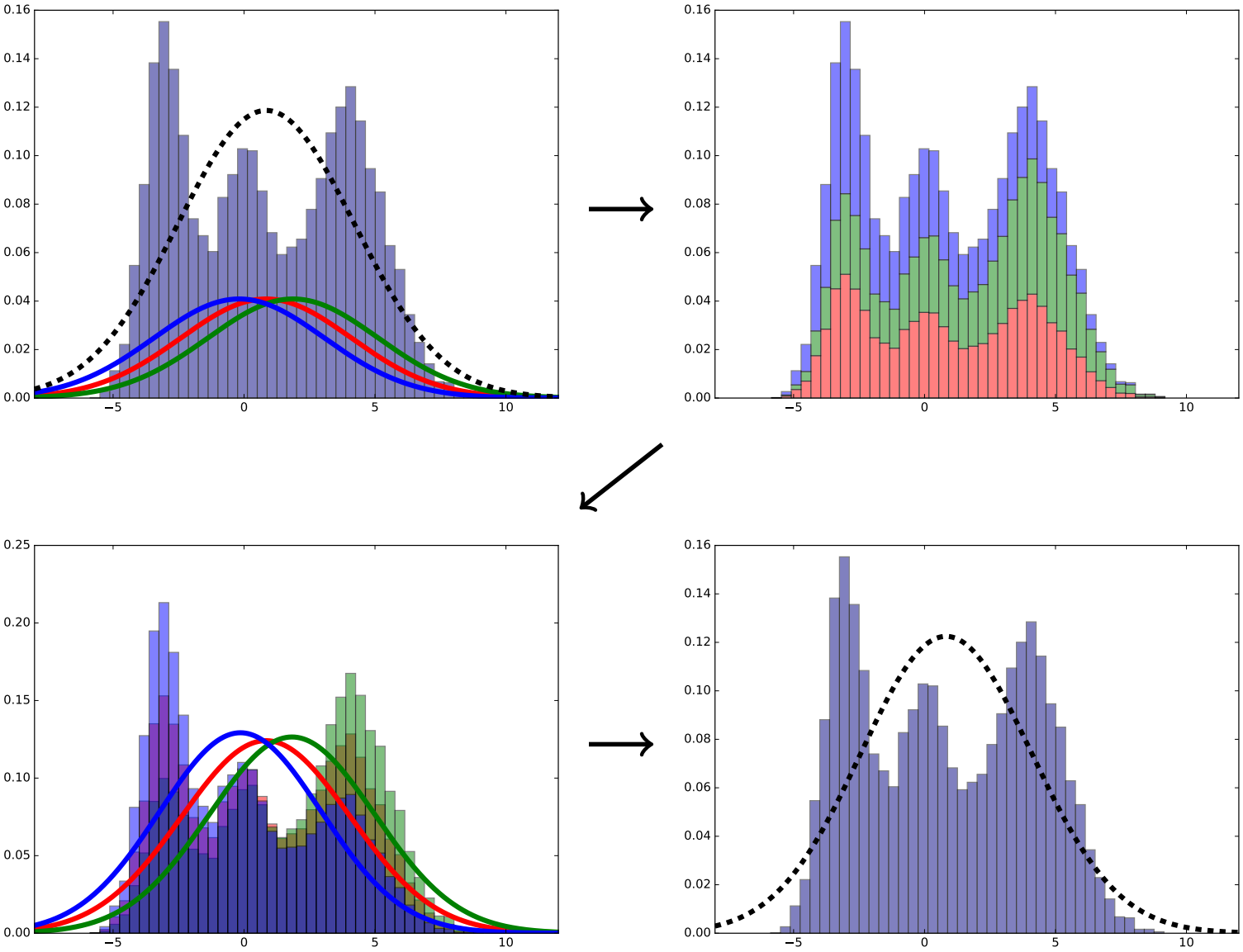
  $$\gamma_{c,i} = P(C_i = c | \boldsymbol{X}_i = \boldsymbol{x}_i)$$

  for each sample of our dataset $\mathcal{D}$

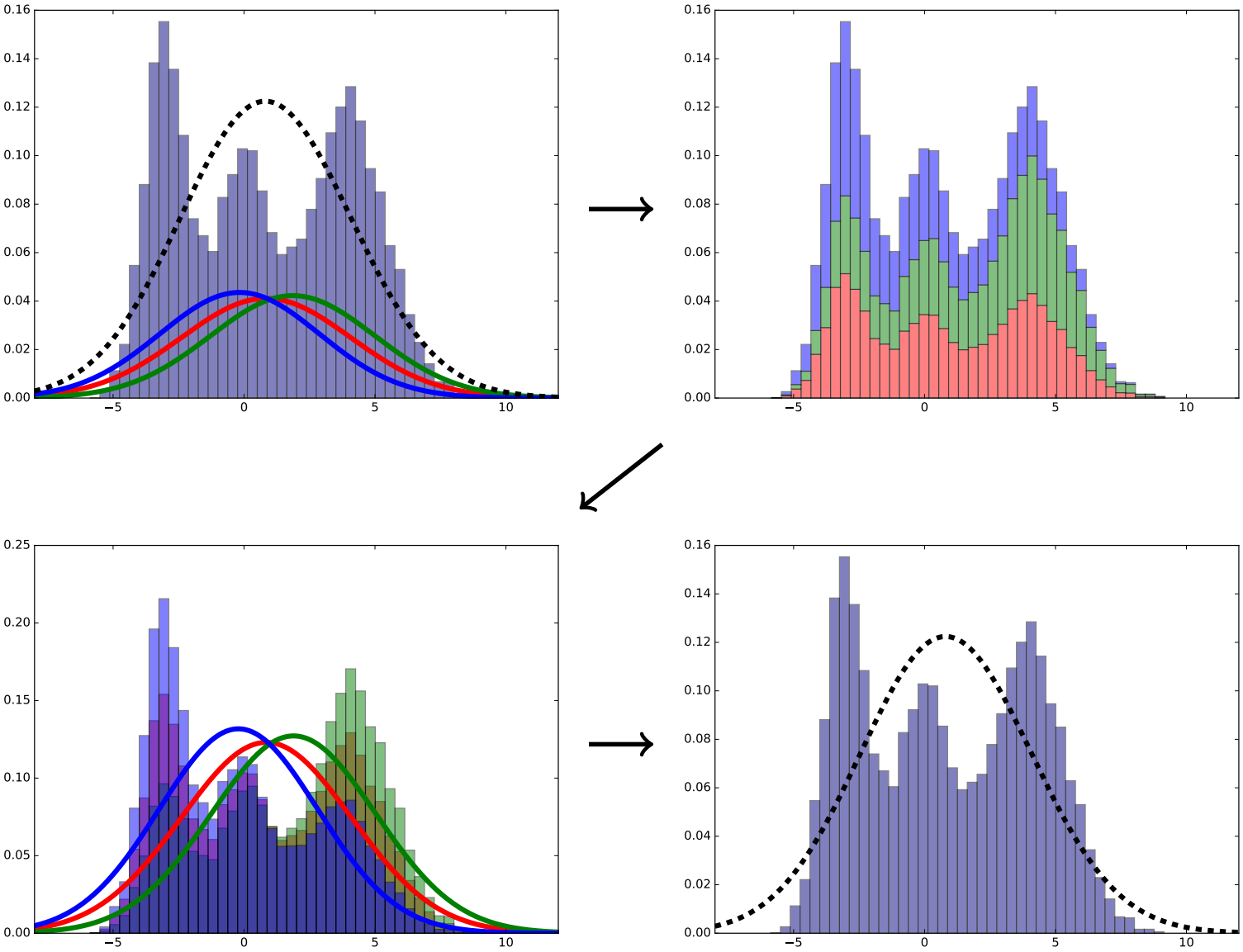
- Given the responsibilities, we re-estimate the GMM parameters $\boldsymbol{\theta}$ using the previous expressions

As we will shortly see, this procedure is a particular instance of an algorithm known as Expectation-Maximization

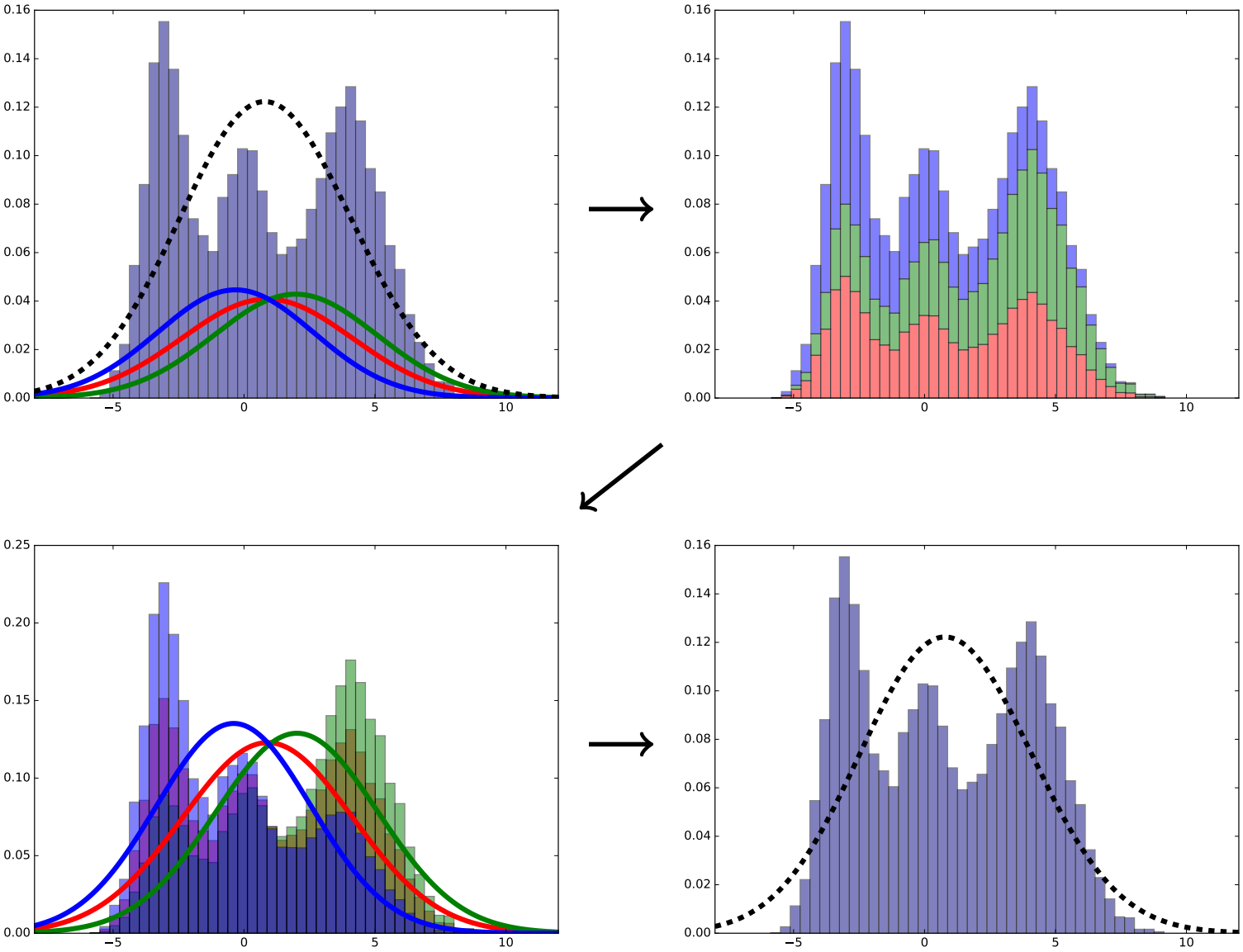
Gaussian Mixture Models



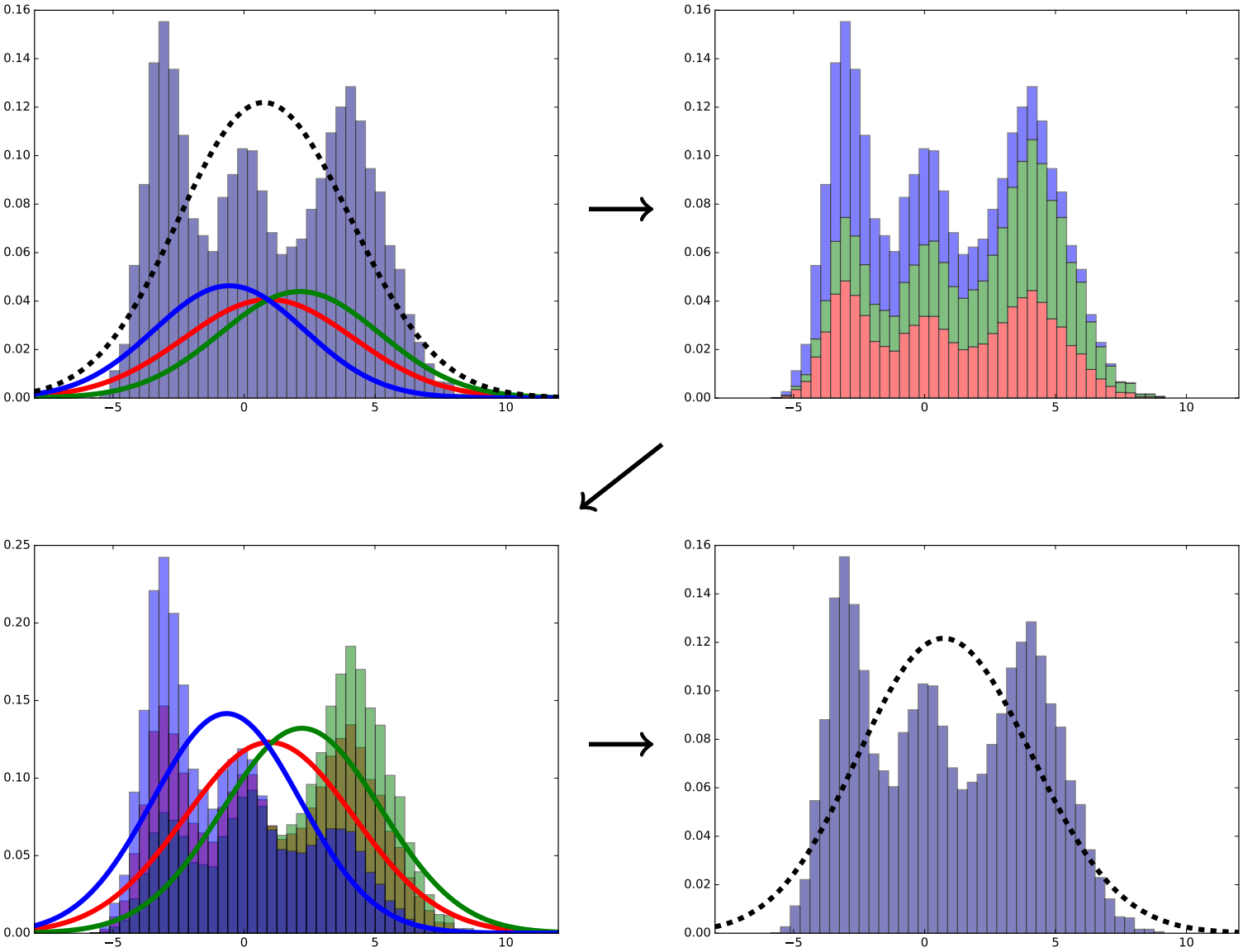
Gaussian Mixture Models



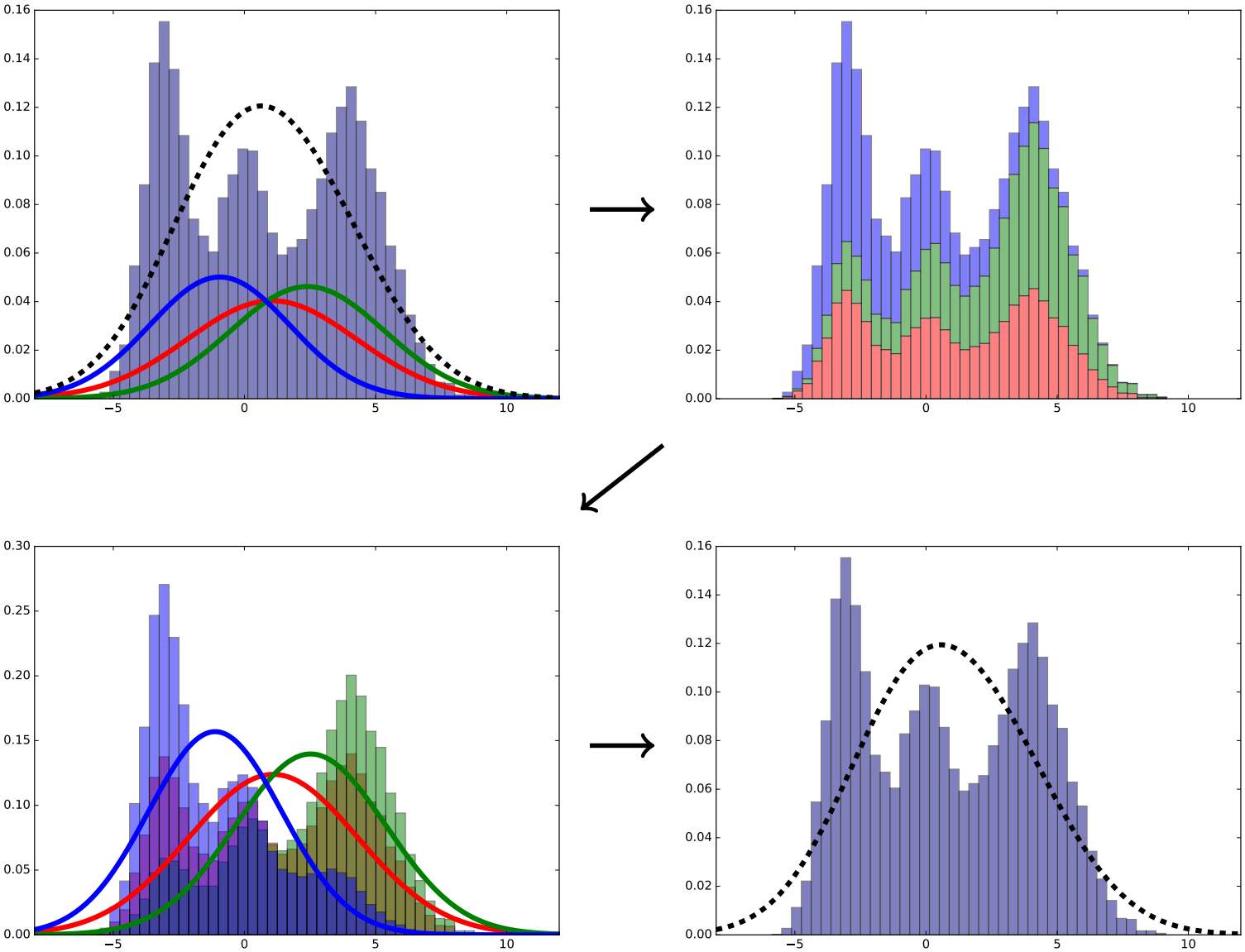
Gaussian Mixture Models



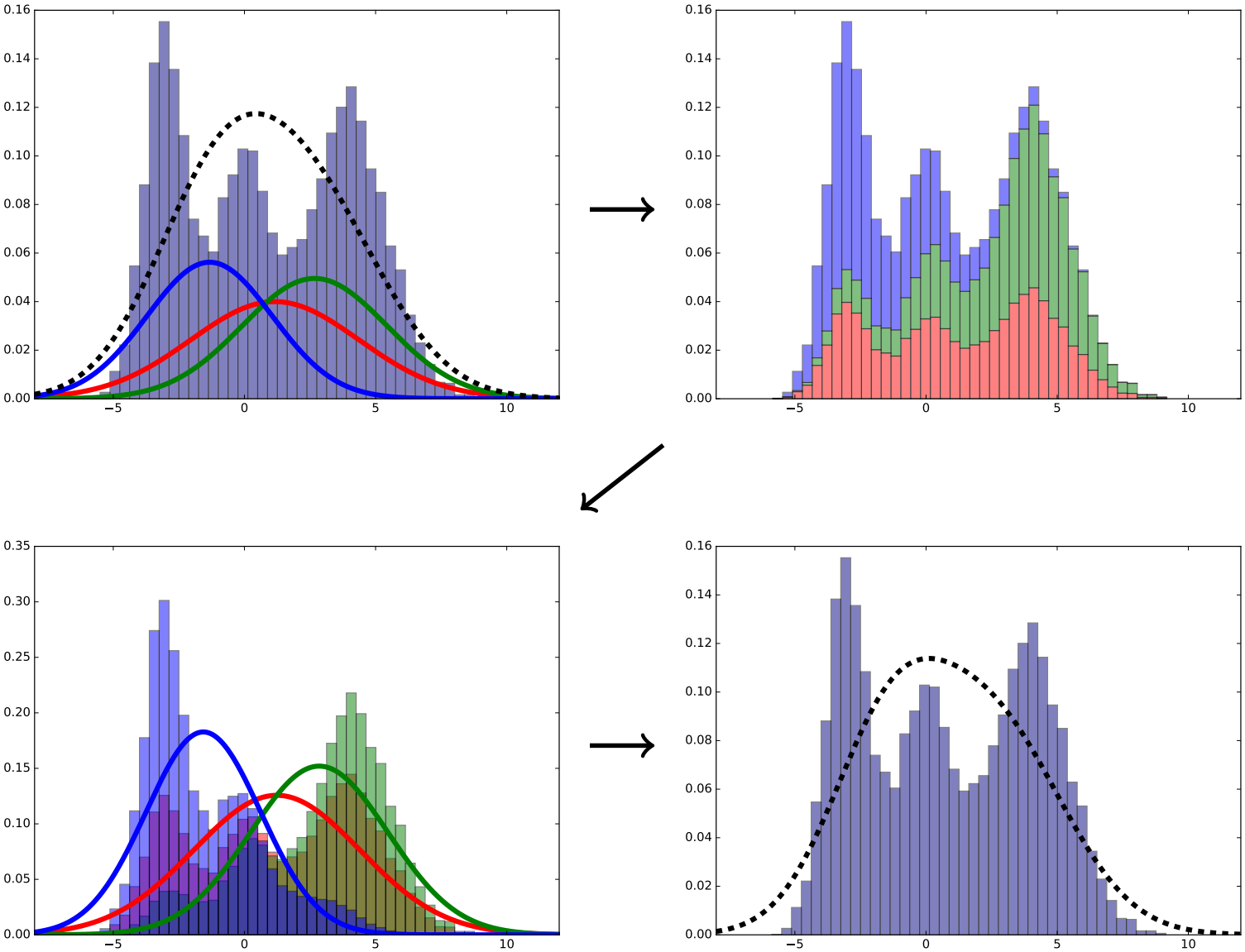
Gaussian Mixture Models



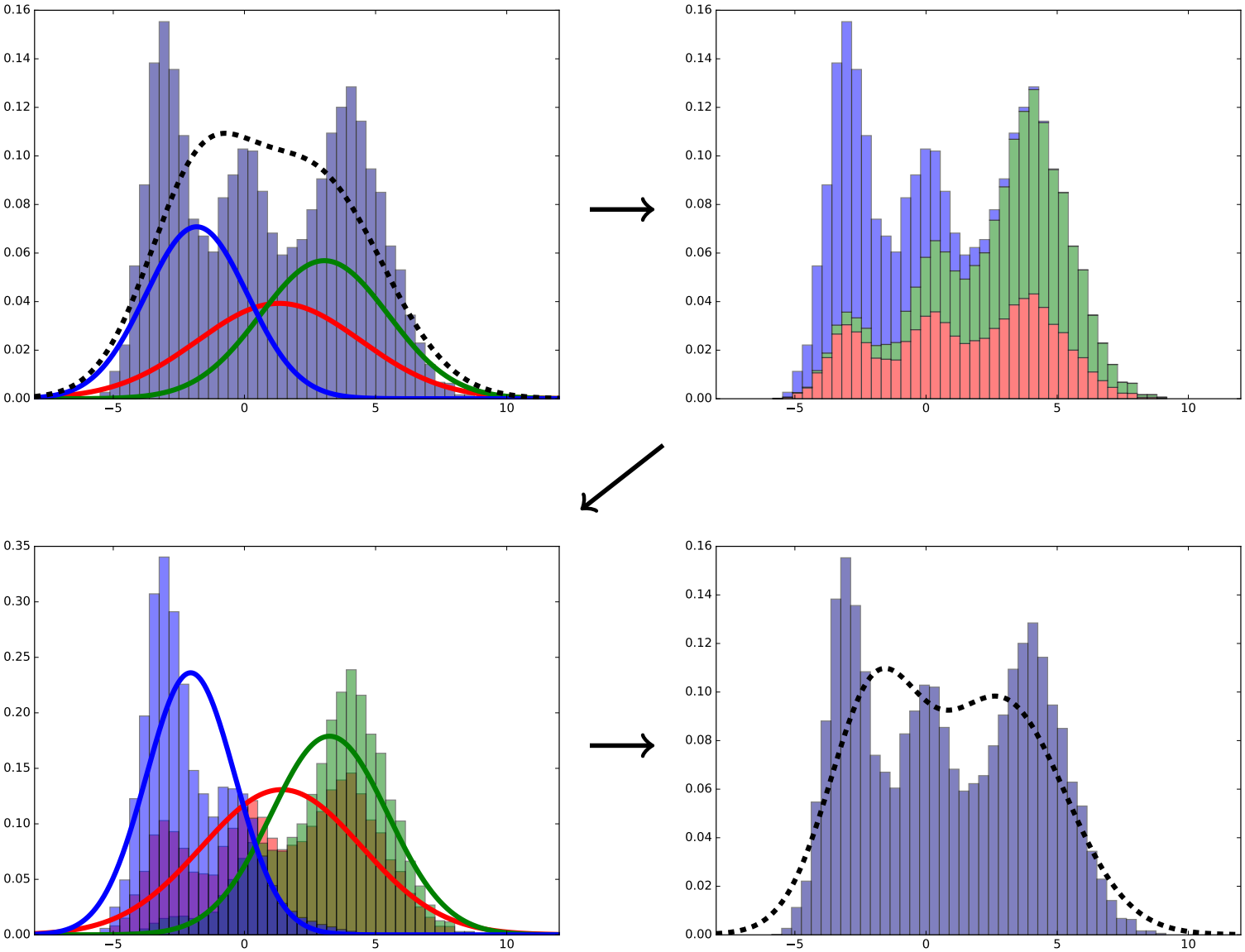
Gaussian Mixture Models



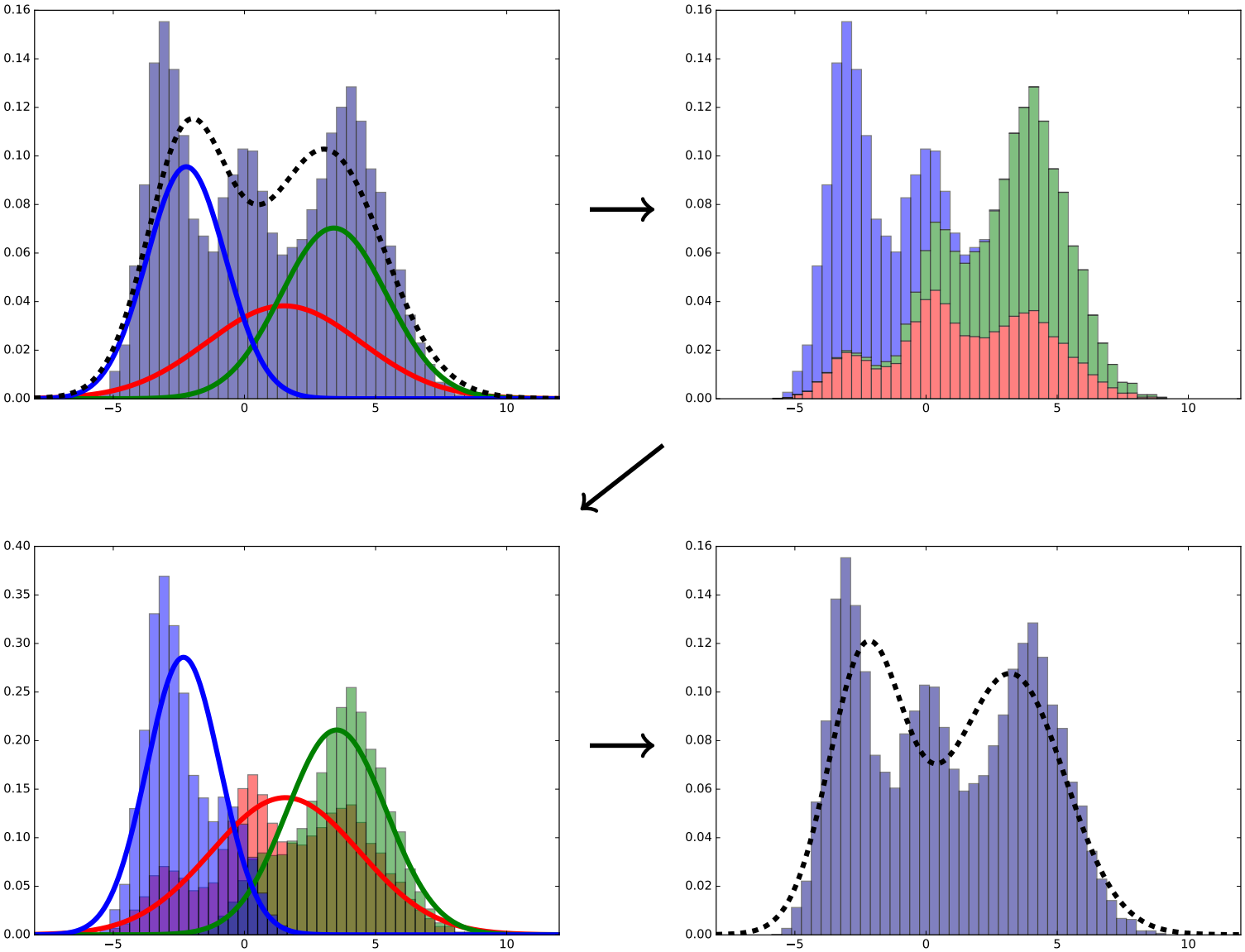
Gaussian Mixture Models



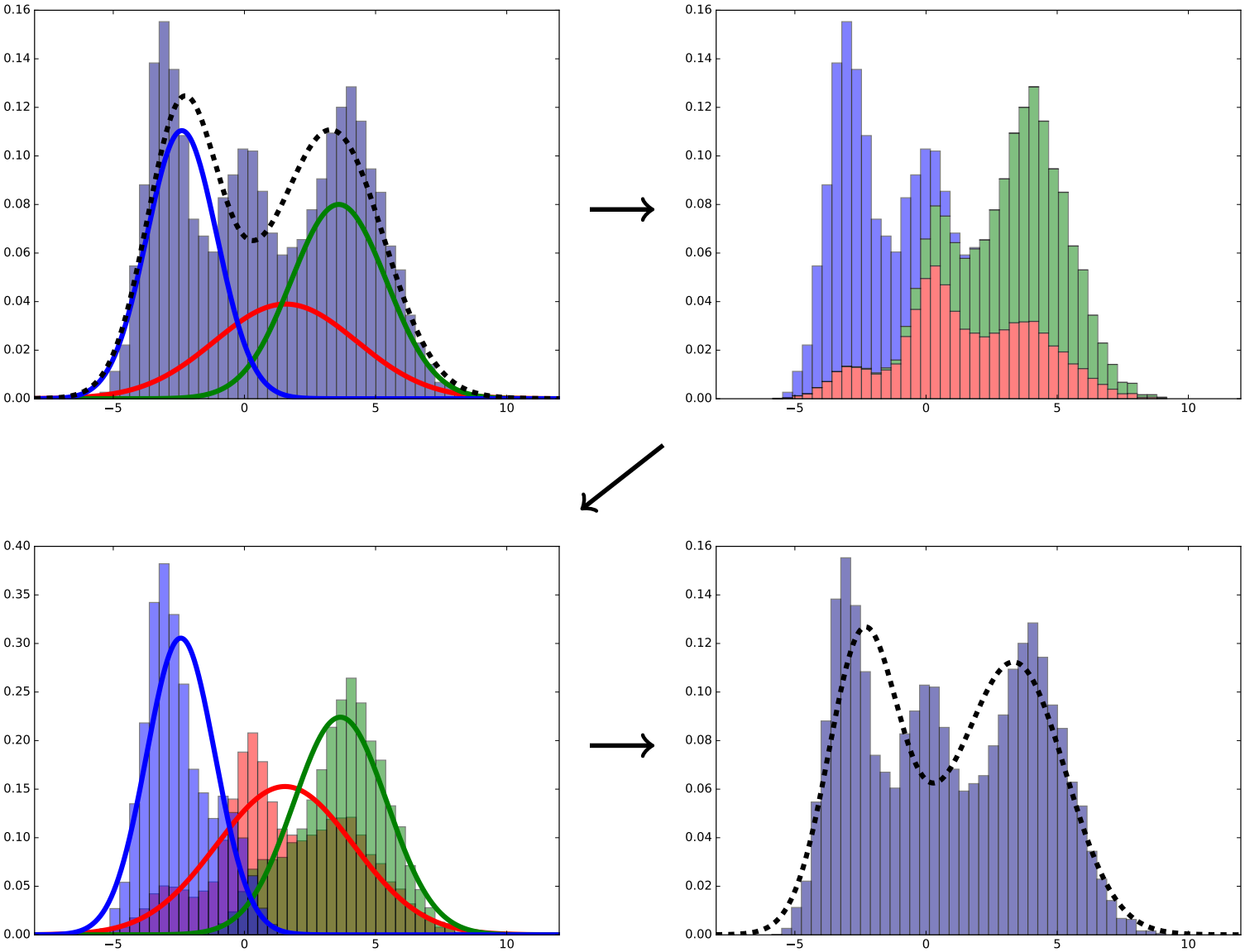
Gaussian Mixture Models



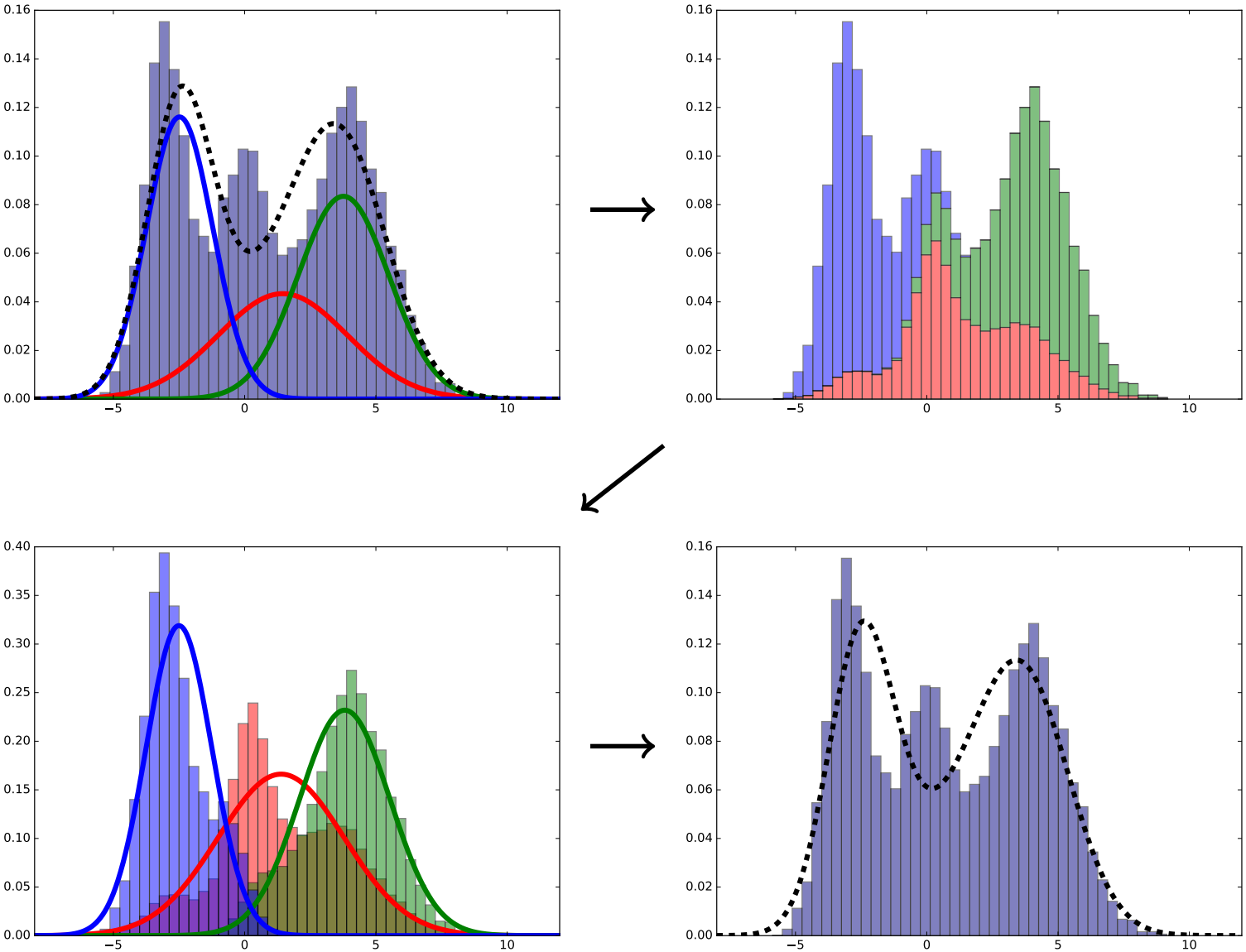
Gaussian Mixture Models



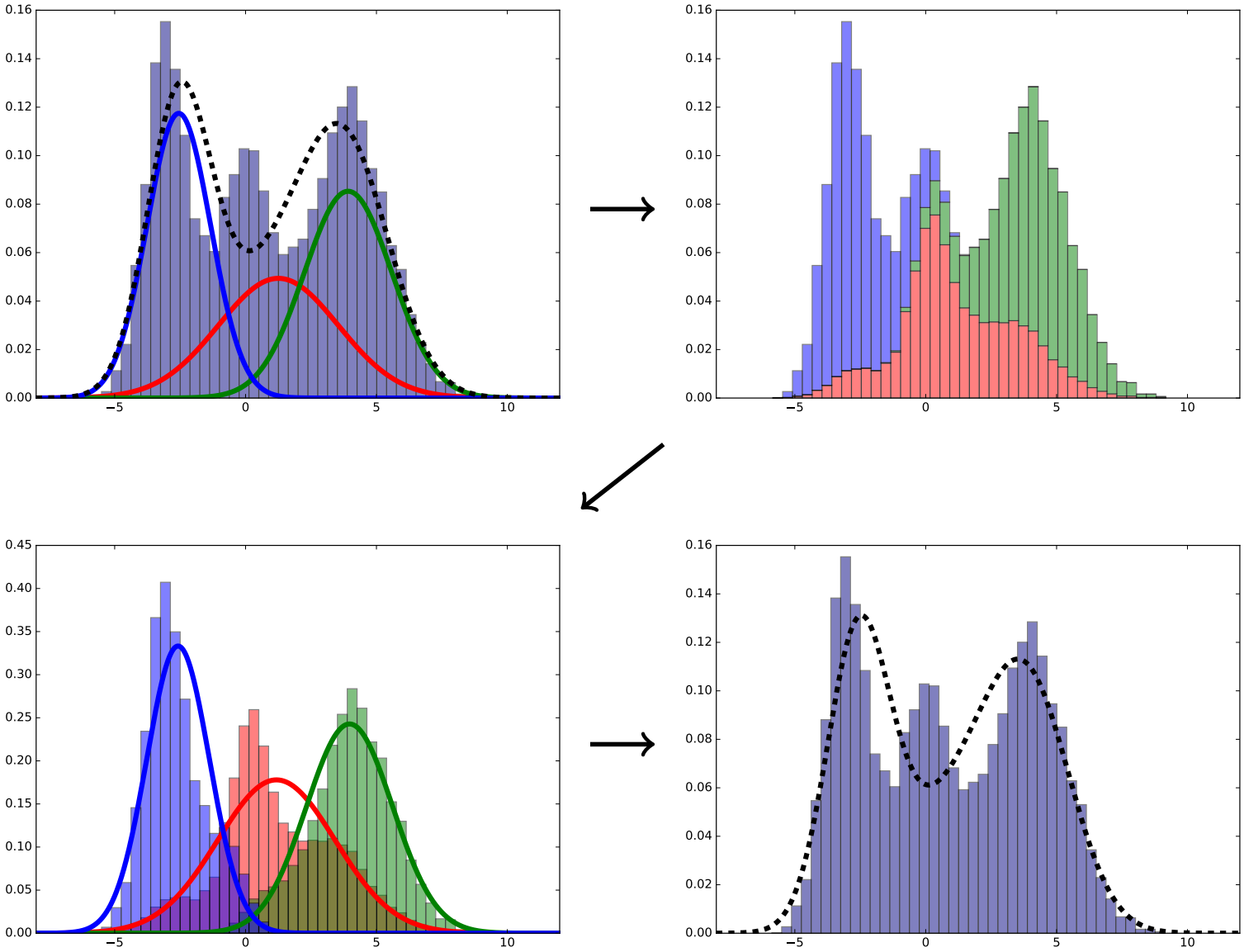
Gaussian Mixture Models



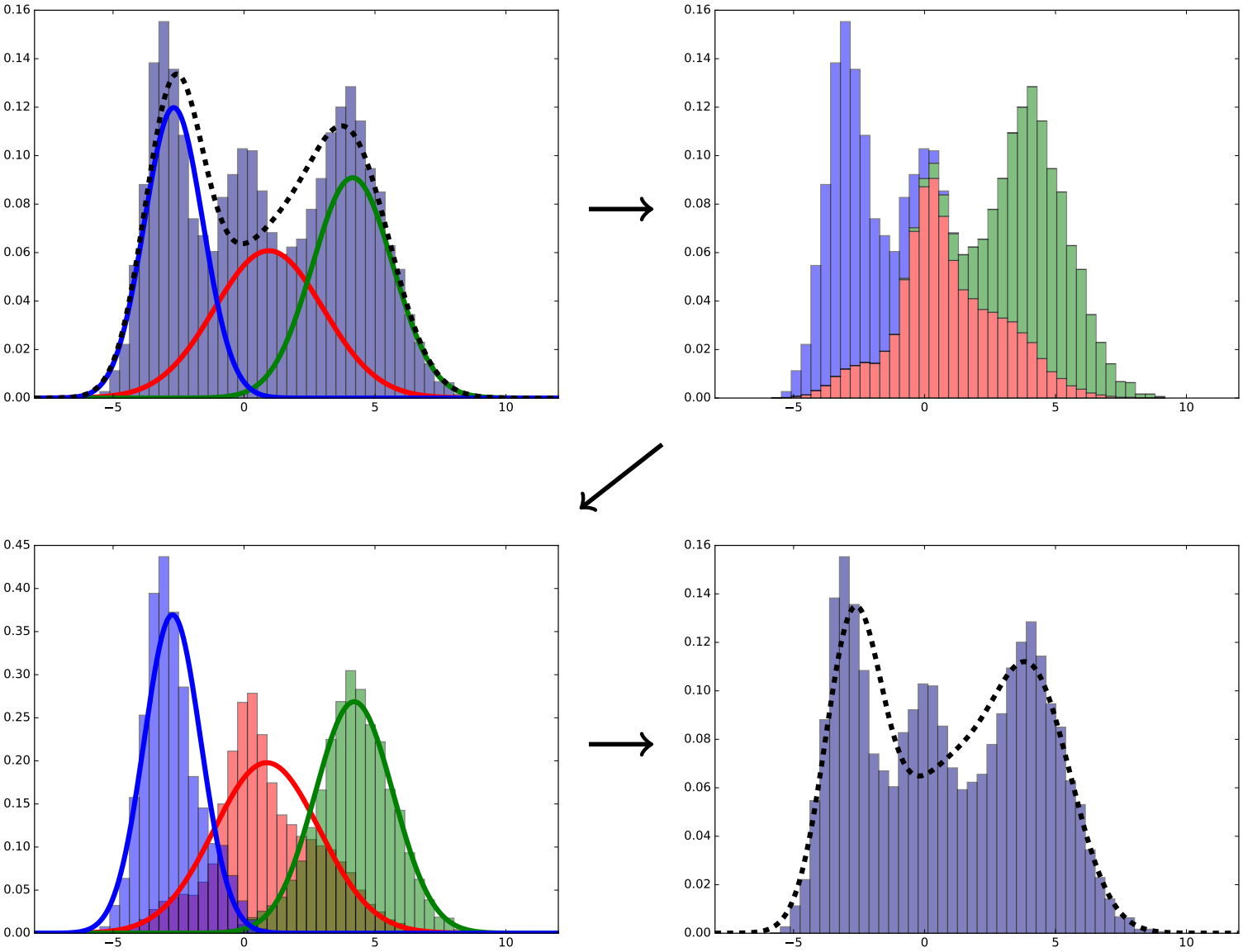
Gaussian Mixture Models



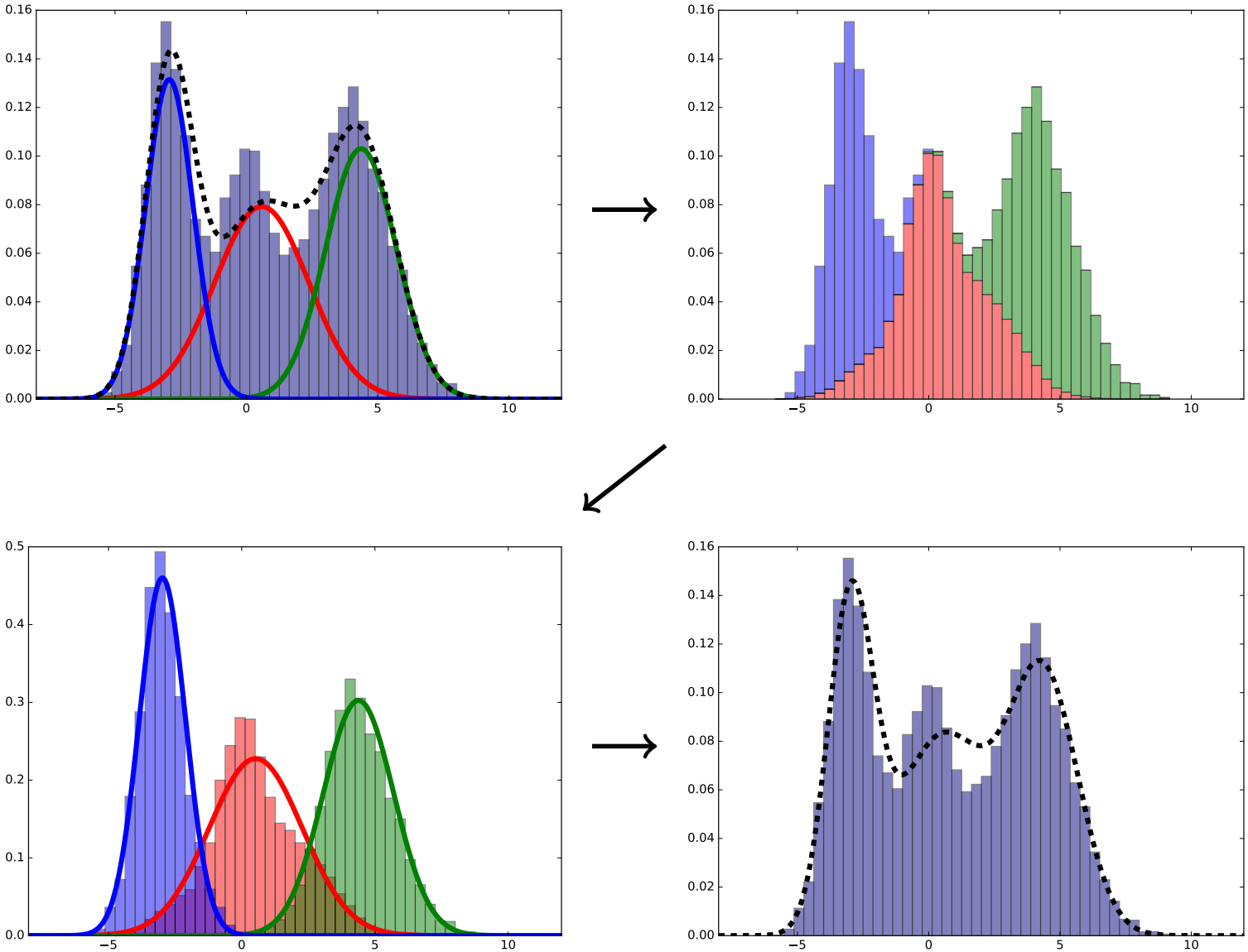
Gaussian Mixture Models



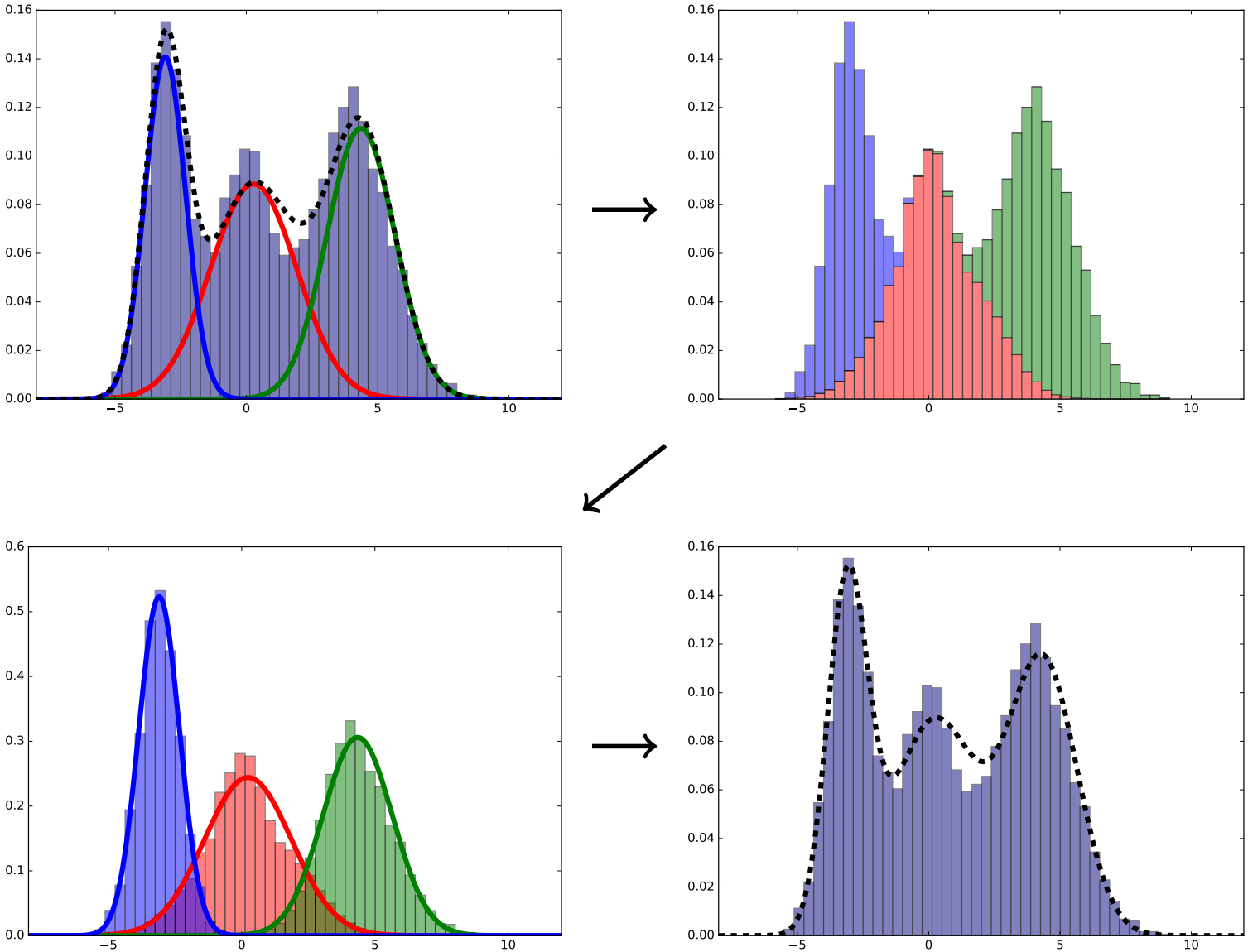
Gaussian Mixture Models



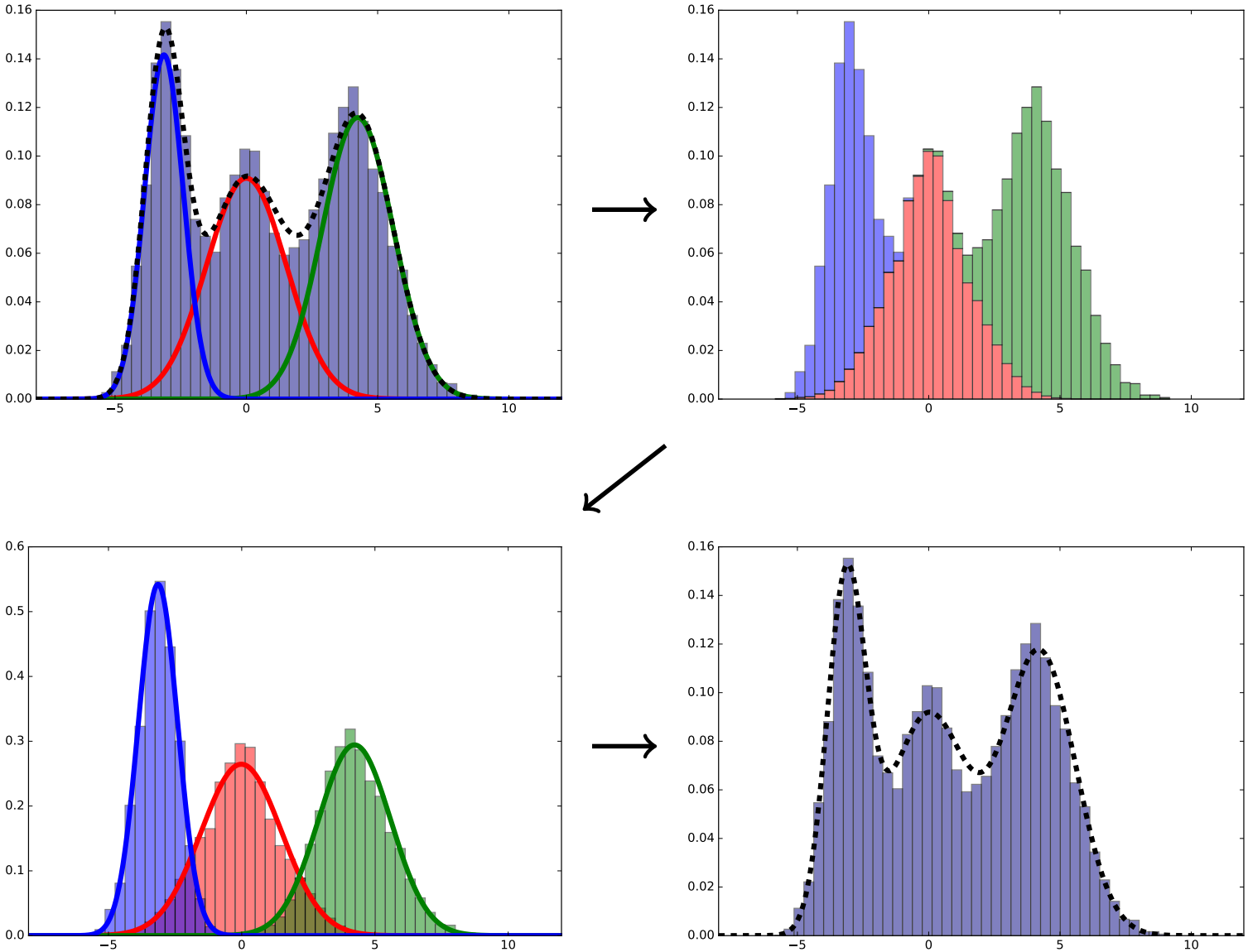
Gaussian Mixture Models



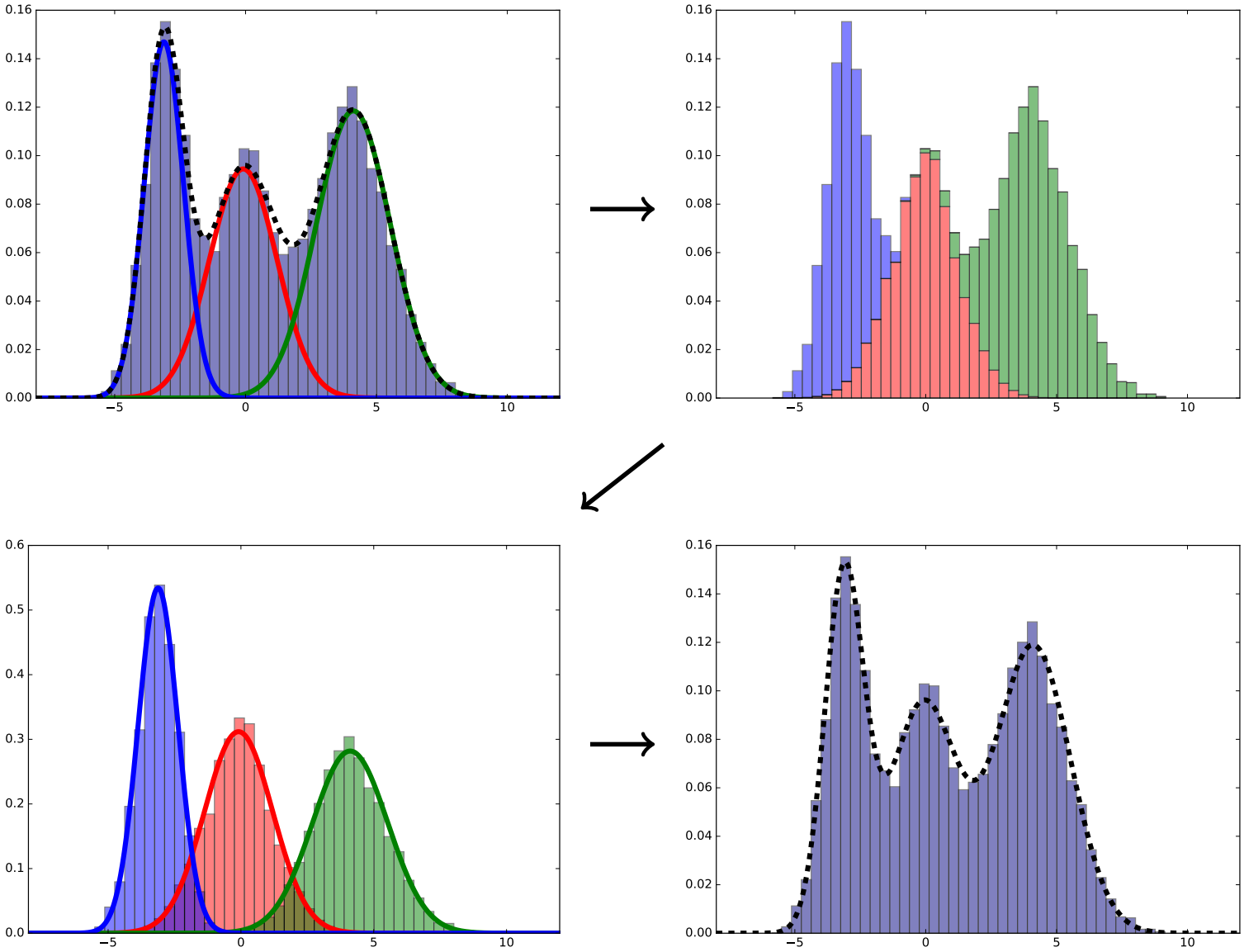
Gaussian Mixture Models



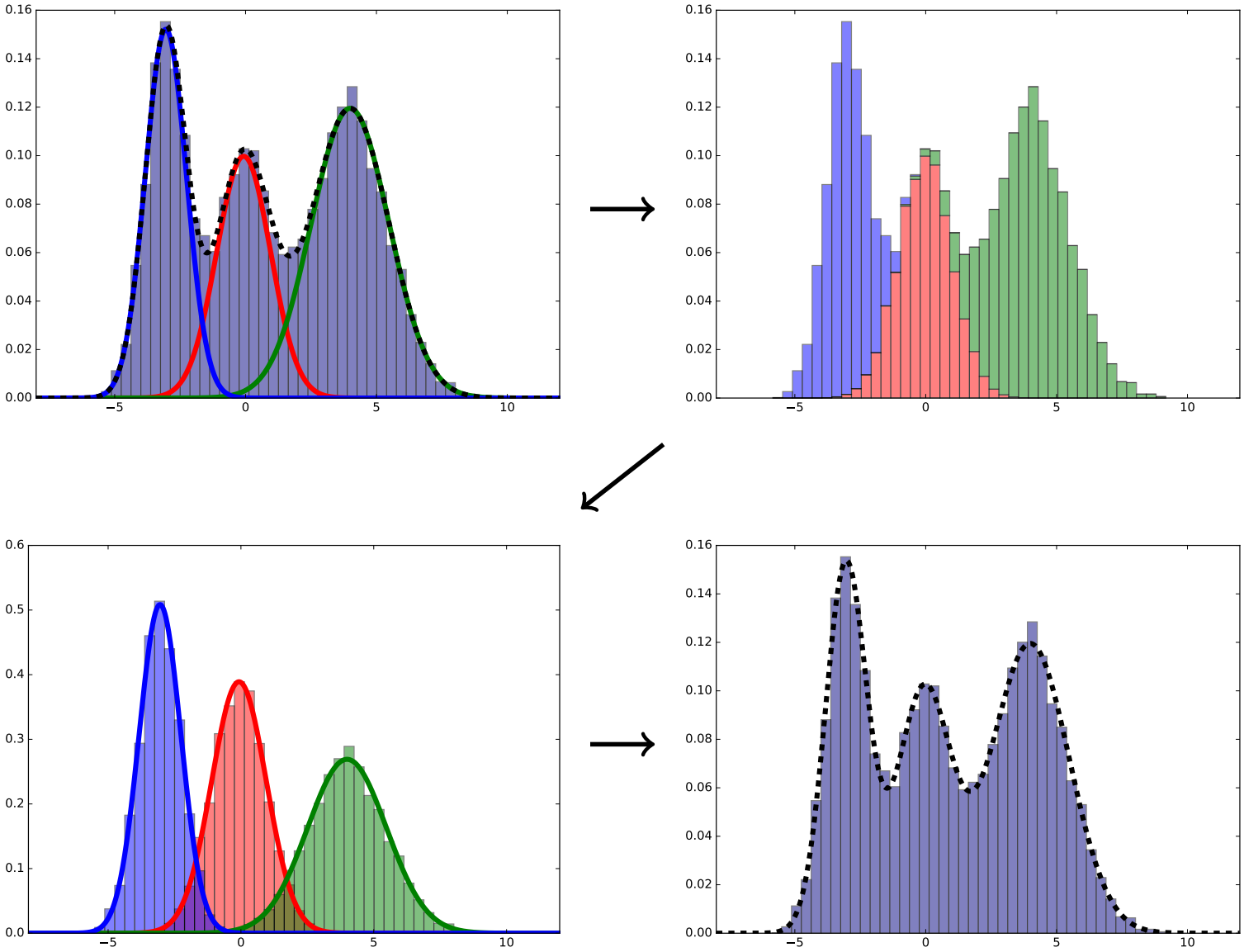
Gaussian Mixture Models



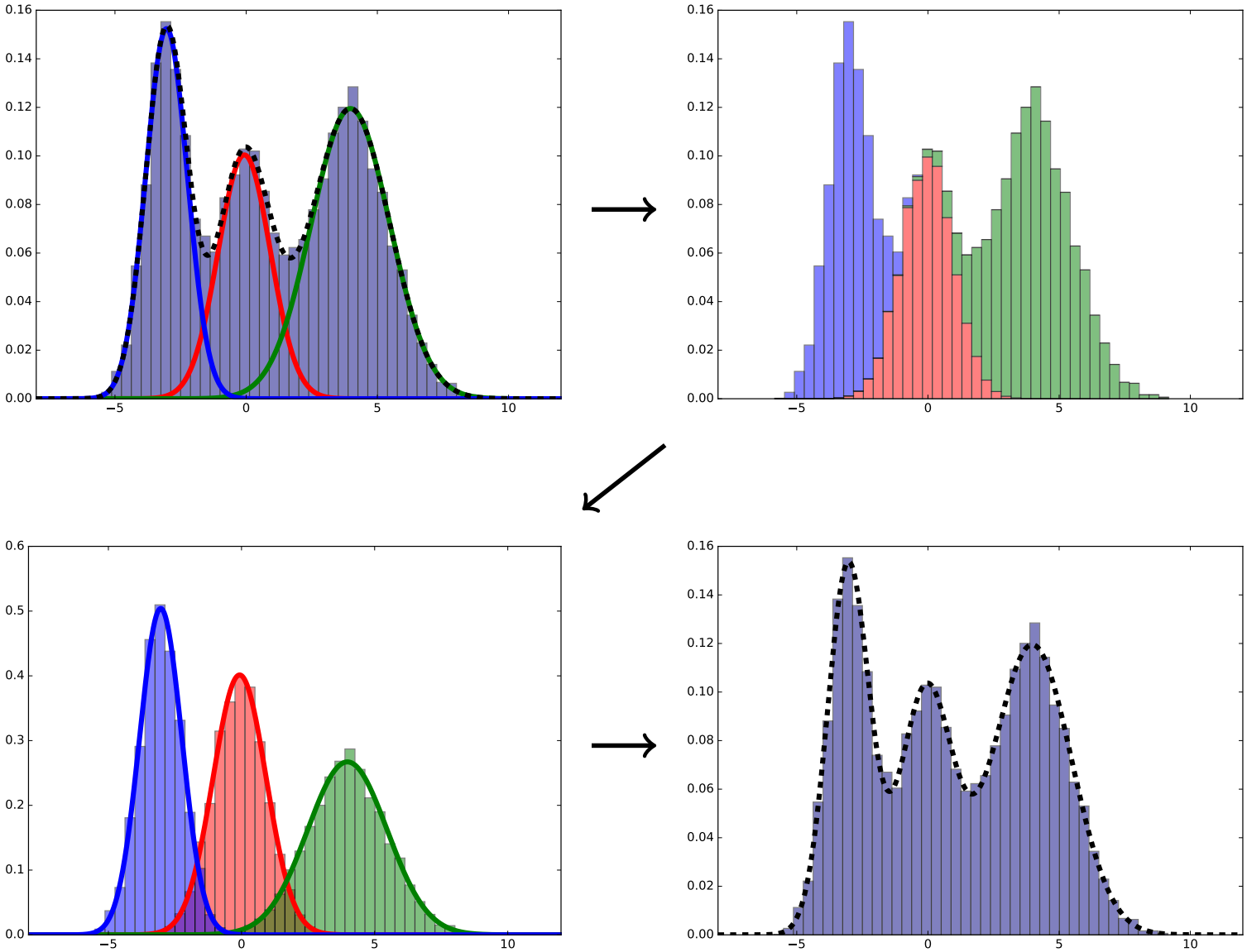
Gaussian Mixture Models



Gaussian Mixture Models

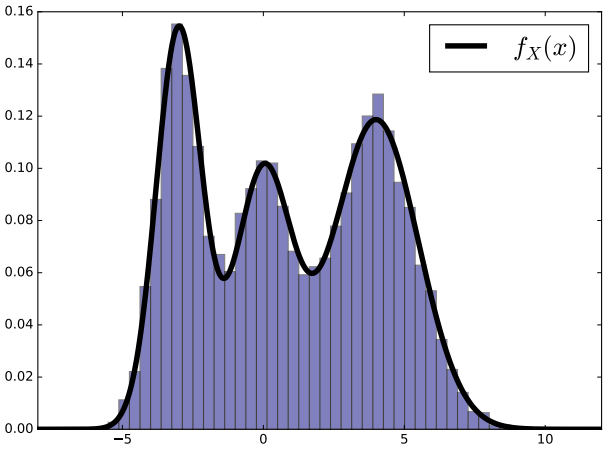


Gaussian Mixture Models

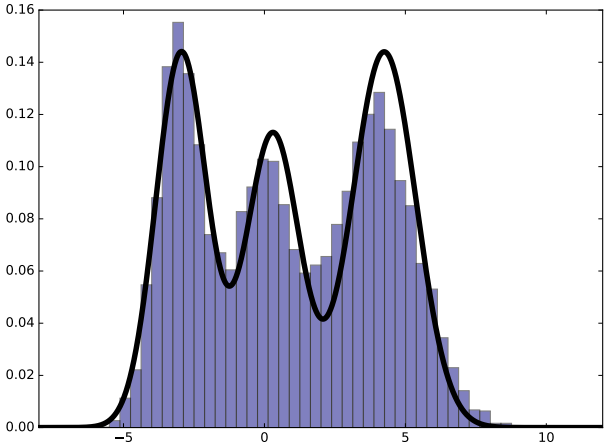


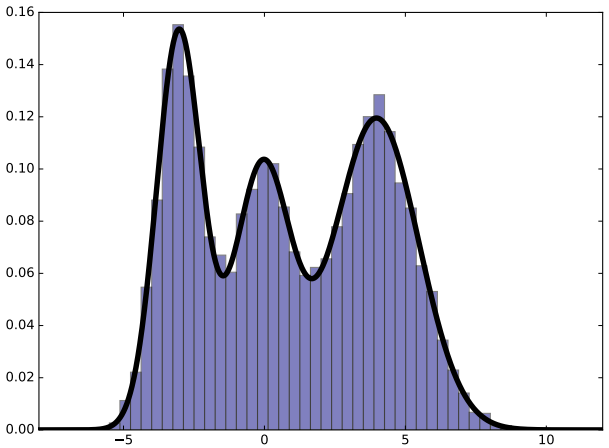
Gaussian Mixture Models

P.d.f.



Hard assignment





Soft assignment

Sampling parameters and estimated values

	π_1	π_2	π_3
P.d.f.	0.25	0.45	0.30
Hard	0.27	0.41	0.32
EM	0.25	0.45	0.30

	μ_1	μ_2	μ_3
P.d.f.	0.00	4.00	-3.00
Hard	0.30	4.25	-2.95
EM	-0.07	3.98	-3.03

	σ_1^2	σ_2^2	σ_3^2
P.d.f.	1.00	2.25	0.64
Hard	0.89	1.27	0.82
EM	0.99	2.23	0.63

Expectation Maximization

Direct maximization of the GMM log-likelihood proved difficult because of the form of the marginal log-density

$$\log f_{\boldsymbol{X}}(x|\boldsymbol{\theta}) = \log \left(\sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right)$$

On the contrary, we have seen that the optimization of joint cluster-feature likelihoods is straightforward: the joint likelihood consists of the product of cluster-conditional normal log-densities and cluster prior probabilities

$$\log f_{X,C}(\boldsymbol{x}, c) = \log \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) + \log w_c$$

and has a simple expression

Expectation Maximization

The EM is an iterative procedure suited for the ML estimation of the parameters of complex likelihoods[2] $f_X(x|\theta)$ that can be expressed through marginalization of joint likelihoods $f_{X,H}(x,h|\theta)$:

$$f_X(x) = \int f_{X,H}(x,h)dh = \int f_{X|H}(x|h)f_H(h)dh$$

NOTE: Here and in the following we do not make any assumption on what X represents. We simply assume that it's a random variable or a random vector, for which we observe a value x. We shall see later that, for our GMM estimation task, X represents the set of random vectors that describe the feature vectors in our dataset $\mathcal{D}$

[2]The derivations hold for both continuous and discrete latent variables, replacing integrals with sums

Expectation Maximization

H represents a latent (or hidden) random variable (or vector) i.e. a R.V. whose value has not been observed, i.e., is unknown

As we shall see, the EM transforms the maximization of a log-likelihood $\log f_X(x|\theta)$ into a sequence of optimizations of expectations of the joint log-likelihood $\log f_{X,H}(x, h|\theta)$

Expectation Maximization

Let's consider again the marginal log-likelihood

$$\ell(\theta) = \log f_X(x|\theta) = \log \frac{f_{X,H}(x,h|\theta)}{f_{H|X}(h|x,\theta)}$$

Given a density $Q(h)$ with the same support of $f_H(h)$, we can rewrite the log-pdf as

$$\begin{aligned}
\log f_X(x|\theta) &= \int Q(h) \log f_X(x|\theta) dh \\
&= \int Q(h) \log \frac{f_{X,H}(x,h|\theta)}{f_{H|X}(h|x,\theta)} dh \\
&= \int Q(h) \log \frac{f_{X,H}(x,h|\theta)}{Q(h)} - \int Q(h) \log \frac{f_{H|X}(h|x,\theta)}{Q(h)} dh
\end{aligned} \tag{4}$$

The term

$$D_h(Q(h)\|f_{H|X}(h|x,\theta)) = -\int Q(h)\log\frac{f_{H|X}(h|x,\theta)}{Q(h)}dh$$
$$= -\mathbb{E}_Q\left[\log\frac{f_{H|X}(h|x,\theta)}{Q(h)}\right]$$

is called Kullback-Leibler (KL) divergence (usually denoted simply as $D(Q\|f_{H|X})$

As we will shortly see, the term

$$\mathcal{L}_h(Q(h),\theta) = \int Q(h)\log\frac{f_{X,H}(x,h|\theta)}{Q(h)}dh = \mathbb{E}_{Q(h)}\left[f_{X,H}(x,h|\theta)\right] + \mathcal{H}\left(Q(h)\right)$$

where $\mathcal{H}\left(Q(h)\right)$ is the entropy of distribution $Q(h)$, provides a lower bound of the log-likelihood (again, the suffix h is usually omitted, but we keep it to remember we are integrating w.r.t. h)

Expectation Maximization

Let's consider the KL divergence

$$D_h(Q(h)\|f_{H|X}(h|x,\theta)) = -\int Q(h)\log\frac{f_{H|X}(h|x,\theta)}{Q(h)}dh$$

Since for every $z > 0$, we have

$$\log z \leq z - 1$$

and $\log z = z - 1$ if and only if $z = 1$, then, for every value of h in the support of Q:

$$-\log\frac{f_{H|X}(h|x,\theta)}{Q(h)} \geq -\frac{f_{H|X}(h|x,\theta)}{Q(h)} + 1$$

with the equality holding if and only if

$$Q(h) = f_{H|X}(h|x,\theta)$$

Expectation Maximization

We thus have

$$-\int Q(h)\log\frac{f_{H|X}(h|x,\theta)}{Q(h)}dh \geq -\int Q(h)\frac{f_{H|X}(h|x,\theta)}{Q(h)}dh+\int Q(h)dh = 0$$

with the quality holding if and only if

$$Q(h) = f_{H|X}(h|x,\theta)$$

almost everywhere (a. e.)[3]

Therefore

$$D_h(Q(h)||f_{H|X}(h|x,\theta)) \geq 0$$

and

$$D_h(Q(h)||f_{H|X}(h|x,\theta)) = 0 \iff Q = f_{H|X} \text{ a. e.}$$

[3] i.e. over all the domain of H, except for at most a subset of zero measure

Expectation Maximization

We have decomposed the log-likelihood as

$$\log f_X(x|\theta) = \mathcal{L}_h(Q(h), \theta) + D_h\left(Q(h) \| f_{H|X}(h|x, \theta)\right)$$

Notice that the left hand side of the equation does not depend on the choice of Q

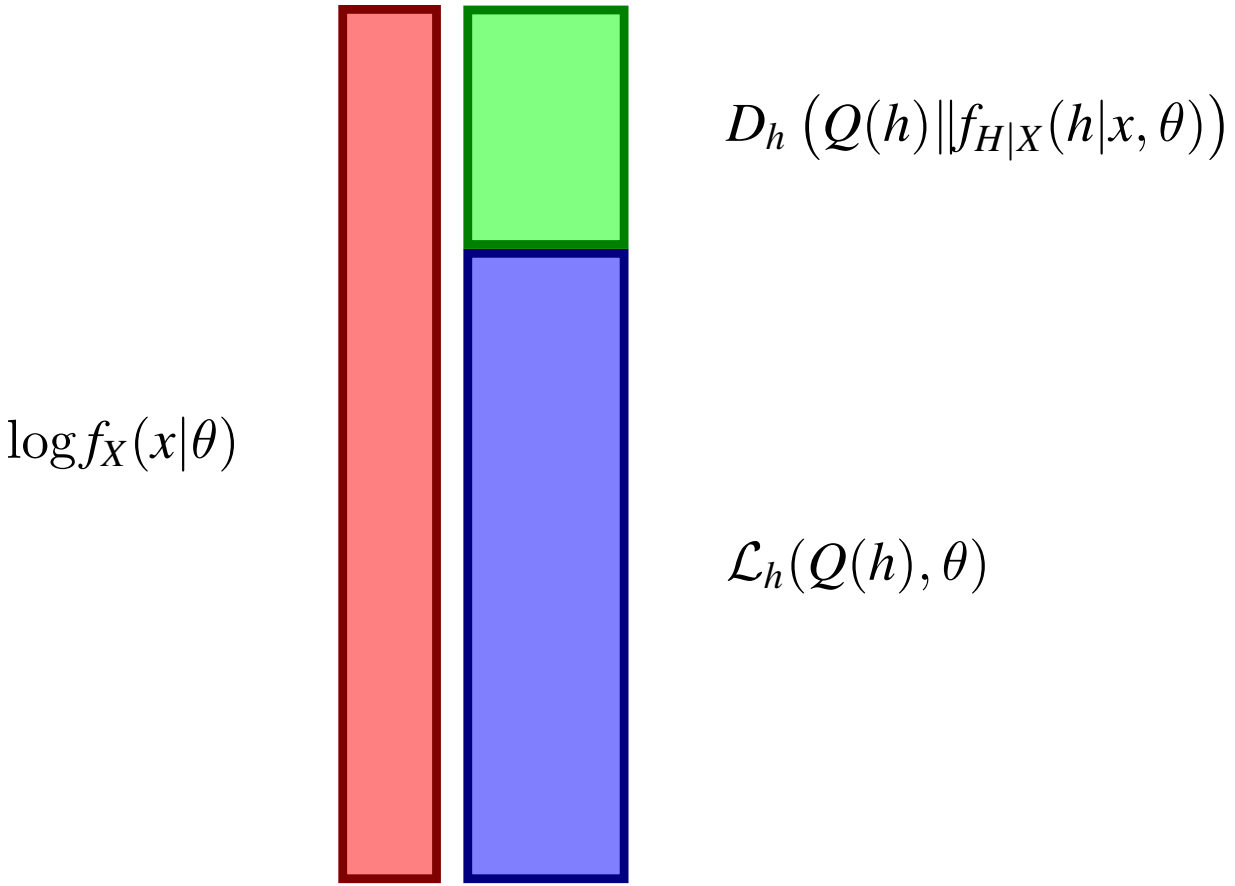
Furthermore,

$$D_h\left(Q(h) \| f_{H|X}(h|x, \theta)\right) \geq 0 \implies \mathcal{L}_h(Q(h), \theta) \leq \log f_X(x|\theta)$$

thus $\mathcal{L}_h(Q(h), \theta)$ is a lower bound on the log-likelihood

Expectation Maximization

Decomposition of $\log f_X(x|\theta) = \mathcal{L}_h(Q(h), \theta) + D_h\left(Q(h) \| f_{H|X}(h|x, \theta)\right)$



Expectation Maximization

The EM algorithm optimizes the log-likelihood by iteratively

- Maximizing the lower bound $\mathcal{L}_h(Q(h), \theta)$ with respect to Q
- Maximizing the lower bound $\mathcal{L}_h(Q(h), \theta)$ with respect to θ

From an initial sets of parameters θ_0:

- $Q_0 = \arg\max_Q \mathcal{L}_h(Q(h), \theta_0)$
- $\theta_1 = \arg\max_\theta \mathcal{L}_h(Q_0(h), \theta)$
- $Q_1 = \arg\max_Q \mathcal{L}_h(Q(h), \theta_1)$
- $\theta_2 = \arg\max_\theta \mathcal{L}_h(Q_1(h), \theta)$
- ...

Expectation Maximization

Let's consider the maximization of the lower bound w.r.t. $Q(h)$, with θ fixed: $\theta = \theta_t$

We have shown that

$$\mathcal{L}_h(Q, \theta_t) \leq \log f_X(x|\theta_t)$$

and

$$Q(h) = f_{H|X}(h|x, \theta_t) \implies \mathcal{L}_h(Q(h), \theta_t) = \log f_X(x|\theta_t)$$

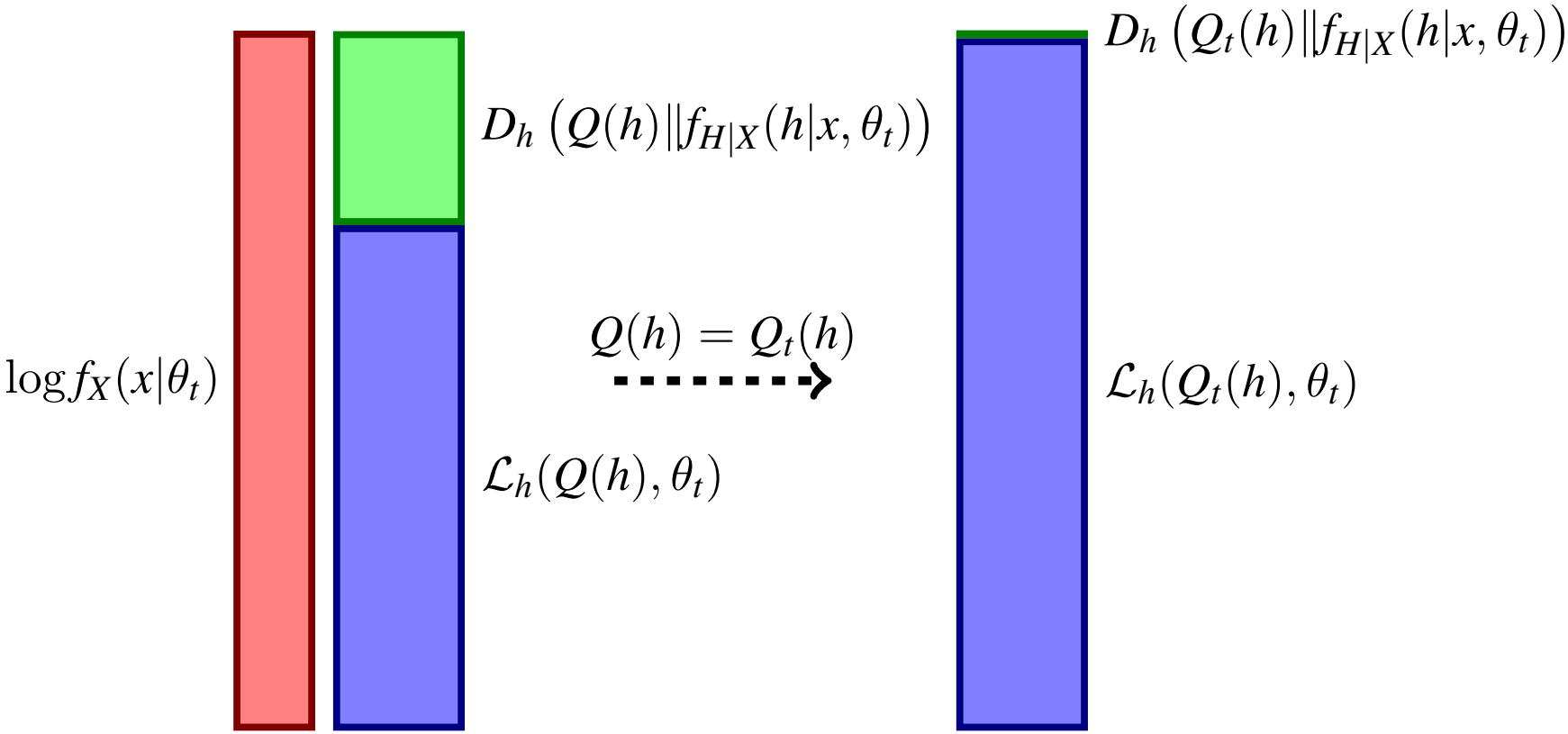
Therefore, we can maximize $\mathcal{L}_h(Q(h), \theta_t)$ w.r.t. Q by simply selecting

$$Q_t(h) = f_{H|X}(h|x, \theta_t)$$

i.e., the posterior for H given X and the fixed value θ_t for the parameters

Expectation Maximization

Setting $Q_t(h) = f_{H|X}(h|x, \theta_t)$ does not change the log-likelihood $\log f_X(x|\theta_t)$, but reduces to zero the KL divergence:



Expectation Maximization

We can now maximize $\mathcal{L}_h(Q_t, \theta)$ w.r.t. θ

This requires computing

$$\theta_{t+1} = \arg\max_{\theta} \mathbb{E}_{Q_t(h)} \log f_{X,h}(x, h|\theta)$$

Notice that the distribution we are taking the expectation with respect to does not involve θ anymore:

$$\begin{aligned}\mathbb{E}_{Q_t(h)} \log f_{X,h}(x, h|\theta) &= \int Q_t(h) \log f_{X,H}(x, h|\theta) dh \\ &= \int f_{H|X}(h|x, \theta_t) \log f_{X,H}(x, h|\theta) dh\end{aligned}$$

since the parameters used in the distribution

$$Q_t(h) = f_{H|X}(h|x, \theta_t)$$

are fixed to θ_t ($Q_t(h)$ does not depend on θ, but on θ_t)

Expectation Maximization

We can also rewrite the problem as maximization of

$$\mathbb{E}_{Q_t(h)} \log f_{X,H}(x,h|\theta) = \mathbb{E}_{Q_t(h)} \log f_{X|H}(x|h,\theta) + \mathbb{E}_{Q_t(h)} \log f_H(h|\theta)$$

which corresponds to the expression we used for the GMM

Since we are maximizing $\mathcal{L}_h(Q_t(h),\theta)$, we have

$$\mathcal{L}(Q_t,\theta_{t+1}) \geq \mathcal{L}(Q_t,\theta_t)$$

Expectation Maximization

$\mathcal{L}_h(Q_t, \theta_{t+1})$ is a lower bound of $\log f_X(x|\theta_{t+1})$, thus

$$\log f_X(x|\theta_{t+1}) \geq \mathcal{L}_h(Q_t, \theta_{t+1})$$

Indeed,

$$\log f_X(x|\theta_{t+1}) = \mathcal{L}(Q_t(h), \theta_{t+1}) + D\left(Q_t, f_{H|X}(h|x, \theta_{t+1})\right)$$

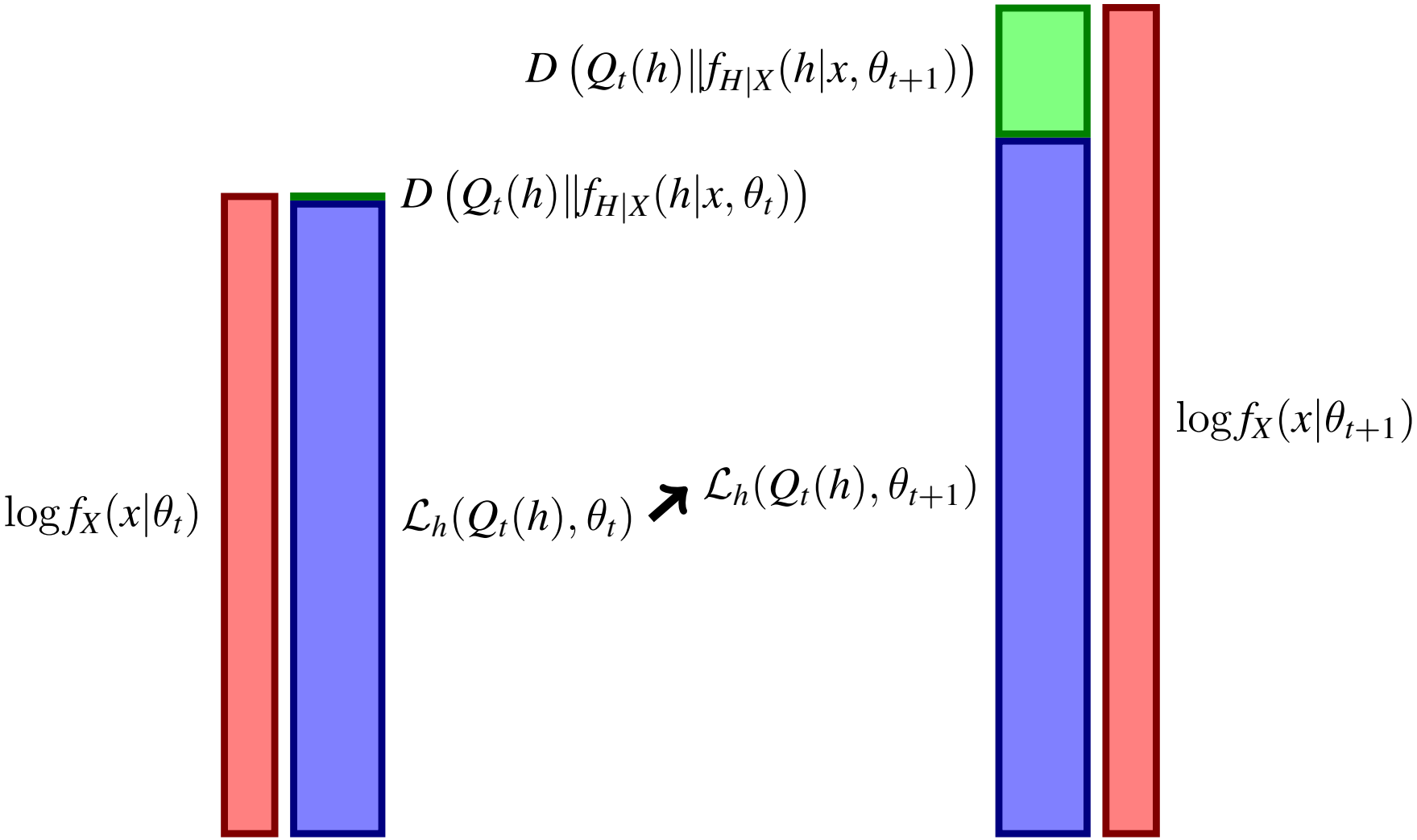
Maximization with respect to θ has increased both $\mathcal{L}_h$ and the KL-divergence, since, in general, $Q_t(h) \neq f_{H|X}(h|x, \theta_{t+1})$ unless we reached convergence

We thus have that

$$\log f_X(x|\theta_{t+1}) \geq \log f_X(x|\theta_t)$$

Expectation Maximization

Maximization w.r.t. θ of $\mathcal{L}_h(Q_t, \theta)$ increases the log-pdf $\log f_X(x|\theta)$



Expectation Maximization

The algorithm iterates between two steps

- **Expectation (E) step:** Compute the posterior distribution $f_{H|X}(h|x,\theta_t)$ and compute the auxiliary function:

$$\mathcal{Q}(\theta,\theta_t) = \mathbb{E}_{f_{H|X}(h|x,\theta_t)}\left[\log f_{X,H}(x,h|\theta)\right]$$

- **Maximization (M) step:** Maximize $\mathcal{Q}(\theta,\theta_t)$ w.r.t. θ to obtain

$$\theta_{t+1} = \arg\max_{\theta} \mathcal{Q}(\theta,\theta_t)$$

Expectation Maximization

Under very weak conditions it can be shown that the EM algorithm converges to a saddle point of the log-likelihood θ^*

Sufficient conditions for θ^* to be a local maximum exist, but are not easy to verify in practice

The saddle point will depend on the initial set of parameters

The choice of a good starting point is important for the estimation of good models

It is sometimes useful to apply several times the EM algorithm with different starting points

Gaussian Mixture Models

Let's apply the algorithm to the GMM

We have a set of n hidden variables $h = (C_1 \dots C_N)$ that represent the cluster assignments, i.e. the assignment of each sample to a component of the GMM

The GMM specifies the joint likelihood for samples and cluster assignments. For a single sample:

$$f_{X_i,C_i}(\boldsymbol{x}_i, c) = w_c \mathcal{N}(\boldsymbol{x}_i | \mu_c, \boldsymbol{\Sigma}_c)$$

The cluster R.V. C_i has a Categorical prior distribution

$$P(C_i = c) = w_c$$

whereas the sample conditional likelihood is

$$f_{X_i|C_i}(\boldsymbol{x}_i | c) = \mathcal{N}(\boldsymbol{x}_i | \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$

Gaussian Mixture Models

We assume that samples are independent given the model parameters, so that we can express the log-likelihood for all the training set samples as

$$\log f_{\boldsymbol{X}_1 \ldots \boldsymbol{X}_N, C_1 \ldots C_N}(\boldsymbol{x}_1 \ldots \boldsymbol{x}_N, c_1 \ldots c_N | \boldsymbol{\theta}) = \sum_{i=1}^{N} \log f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c_i | \boldsymbol{\theta})$$

The EM algorithm requires computing the posterior for the hidden variables $C_1 \ldots C_N | \boldsymbol{X}_1 \ldots \boldsymbol{X}_N, \boldsymbol{\theta}$.

Due to the independence assumptions, also the posterior distribution factorizes as

$$f_{C_1 \ldots C_N | \boldsymbol{X}_1 \ldots \boldsymbol{X}_N}(c_1 \ldots c_N | \boldsymbol{x}_1 \ldots \boldsymbol{x}_N, \boldsymbol{\theta}) = \prod_{i=1}^{N} P(C_i = c_i | \boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{\theta})$$

The **E-step** requires computing the auxiliary function

$$\begin{aligned}\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}_t) &= \mathbb{E}_{C_1 \dots C_N | \boldsymbol{X}_1 = \boldsymbol{x}_1 \dots \boldsymbol{X}_N = \boldsymbol{x}_N, \boldsymbol{\theta}_t} \left[\log f_{\boldsymbol{X}_1 \dots \boldsymbol{X}_N, C_1 \dots C_N}(\boldsymbol{x}_1 \dots \boldsymbol{x}_N, c_1 \dots c_N | \boldsymbol{\theta}) \right] \\ &= \sum_{i=1}^{N} \mathbb{E}_{C_1 \dots C_N | \boldsymbol{X}_1 = \boldsymbol{x}_1 \dots \boldsymbol{X}_N = \boldsymbol{x}_N, \boldsymbol{\theta}_t} \left[\log f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c | \boldsymbol{\theta}) \right] \\ &= \sum_{i=1}^{N} \mathbb{E}_{C_i | \boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{\theta}_t} \left[\log f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c | \boldsymbol{\theta}) \right]\end{aligned}$$

The EM algorithm becomes:

E-step: Compute $\gamma_{c,i} = P(C_i = c|X_i = \boldsymbol{x}_i, \boldsymbol{\theta}^t)$. The auxiliary function is

$$\begin{aligned}
\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}_t) = & \sum_{i=1}^{N}\sum_{c=1}^{K} P(C_i = c|\boldsymbol{X}_i = \boldsymbol{x}_i, \boldsymbol{\theta}_t) \log f_{\boldsymbol{X}_i, C_i}(\boldsymbol{x}_i, c|\boldsymbol{\theta}) \\
= & \sum_{i=1}^{N}\sum_{c=1}^{K} \gamma_{c,i} \left[\log \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) + \gamma_{c,i} \log w_c\right] \\
= & \sum_{i=1}^{N}\sum_{c=1}^{K} \gamma_{c,i} \left(\frac{1}{2}\log|\boldsymbol{\Lambda}_c| - \frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T \boldsymbol{\Lambda}_c (\boldsymbol{x}_i - \boldsymbol{\mu}_c)\right) \\
& + \sum_{i=1}^{N}\sum_{c=1}^{K} \gamma_{c,i} \log w_c
\end{aligned}$$

The EM algorithm becomes:

M-step: Maximize $\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$ w.r.t. $\boldsymbol{\theta} = (\boldsymbol{M}, \boldsymbol{\mathcal{S}}, \boldsymbol{w})$, subject to $\sum_{k=1}^{K} w_k = 1$:

$$\boldsymbol{\mu}_c^* = \frac{\sum_i \gamma_{c,i} \boldsymbol{x}_i}{\sum_i \gamma_{c,i}}$$

$$\boldsymbol{\Sigma}_c^* = \frac{\sum_i \gamma_{c,i} (\boldsymbol{x}_i - \boldsymbol{\mu}_c)(\boldsymbol{x}_i - \boldsymbol{\mu}_c)^T}{\sum_i \gamma_{c,i}}$$

$$w_c^* = \frac{\sum_i \gamma_{c,i}}{\sum_i \sum_c \gamma_{c,i}}$$

The new estimate of the parameters is $\theta_{t+1} = (\boldsymbol{M}_{t+1}, \boldsymbol{\mathcal{S}}_{t+1}, w_{t+1})$:

$$\boldsymbol{M}_{t+1} = [\boldsymbol{\mu}_1^* \dots \boldsymbol{\mu}_K^*] \ , \quad \boldsymbol{\mathcal{S}}_{t+1} = [\boldsymbol{\Sigma}_1^* \dots \boldsymbol{\Sigma}_K^*] \ , \quad \boldsymbol{w}_{t+1} = [w_1^* \dots w_K^*]$$

Gaussian Mixture Models for classification

Just as we used Gaussian densities for modeling the samples of different classes in a classification task, we can use GMM to model the class conditional distribution

We can, for example, assume that the samples of class c are generated by a GMM with parameters $(\boldsymbol{M}_c, \boldsymbol{S}_c, \boldsymbol{w}_c)$

For each class we want to recognize, we can compute the ML estimate of a GMM for the samples of that class. We can then use the estimated densities to compute class conditional log-likelihoods and class posterior distributions or log-likelihood ratios

Gaussian Mixture Models for classification

MVG for classification:

- Fit a guassian density to samples of each class

$$X_i|C_i = c \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$
$$f_{X_t|C_t}(\boldsymbol{x}_t|c) = \mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$$
$$P(C_t = c|\boldsymbol{X}_t = \boldsymbol{x}_t) = \frac{\mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)P(C_t = c)}{\sum_c' \mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_{c'}, \boldsymbol{\Sigma}_{c'})P(C_i = c')}$$

Gaussian Mixture Models for classification

GMM for classification:

- Choose the number of components for each class K_c
- Fit a GMM with K_c compoennts to samples of each class

$$X_i|C_i = c \sim GMM(\boldsymbol{M}_c, \boldsymbol{\mathcal{S}}_c, \boldsymbol{w}_c)$$

$$f_{\boldsymbol{X}_t|C_t}(\boldsymbol{x}_t|c) = \sum_{k=1}^{K_c} w_k \mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_{c,k}, \boldsymbol{\Sigma}_{c,k})$$

$$P(C_t = c|\boldsymbol{X}_t = \boldsymbol{x}_t) = \frac{P(C_t = c)\sum_{k=1}^{K_c} w_{c,k}\mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_{c,k}, \boldsymbol{\Sigma}_{c,k})}{\sum_{c'} P(C_t = c')\sum_{k=1}^{K'_c} w_{c',k}\mathcal{N}(\boldsymbol{x}_t|\boldsymbol{\mu}_{c',k}, \boldsymbol{\Sigma}_{c',k})}$$

Gaussian Mixture Models for classification

We can also exploit GMMs clustering capabilities for open-set multiclass classification

The difficulty in open-set classification consists in building a robust model for the none-of-the-others class

This class is usually very heterogeneous, and explicit modeling its sub-components requires labeled examples of its possible objects

We can partially alleviate the issue using a GMM.

Gaussian Mixture Models for classification

We can assume that samples of known classes can be modeled by MVG distributions

We can collect a large set of unlabeled samples for the none-of-the-others class

We can model the samples of the none-of-the-others class using a GMM

The GMM will find homogeneous clusters (sub-classes) of the none-of-the-others population, and can be used as an estimate for the conditional density of a test sample assuming that it belongs to the none-of-the-others class[4]

[4] Given the complexity of the task, and due to the fact that different amounts of parameters are used for modeling the none-of-the-others class these kind of models often provide class-conditional likelihoods that are not calibrated, and may require further score processing (e.g. score calibration) to obtain good decisions

Gaussian Mixture Models

As we did for MVG, we can train GMM with diagonal covariance matrices to reduce the number of parameters to estimate (reducing overfitting and computational costs).

The solution is again given by the diagonals of Σ_c^*'s we defined before

We may need more components to model more complex distributions

The diagonal covariance assumption DOES NOT correspond to the Naive Bayes assumption in this case

The Naive Bayes assumption would correspond to training a different GMM (possibly with different number of components) for each subset of features that is assumed independent from the other features

Gaussian Mixture Models for classification

We can also assume that all components of a GMM have the same covariance matrix (tied GMM)

Note that in this case we are tying the components of a single GMM, i.e. the Gaussian components of the GMM of a single class

This is different from the Tied Gaussian model, where parameters were shared across classes

Of course, we can extend GMM parameters tying across classes as well — we won't consider this model though, as it would require revisiting the EM estimation procedure, since sharing the parameters across classes would not allow us to independently estimate a GMM over the samples of each class

Gaussian Mixture Models for classification

MNIST — GMM (PCA 50)

Components:	1	2	4	8	16
FullCov	3.6%	3.4%	2.8%	2.3%	2.2%
Diagonal	12.3%	10.1%	8.9%	7.6%	6.2%

Components:	32	64	128	256
FullCov	2.3%			
Diagonal	5.1%	4.3%	4.3%	4.3%

Gaussian Mixture Models

Initialization plays an important role in GMM training

Hard-assignment with isotropic covariances $\longrightarrow$ K-means

K-means can be used as initializer

Alternative approach: LBG (can also be used for K-means)

LBG algorithm:

- Split the components of a G-components GMM

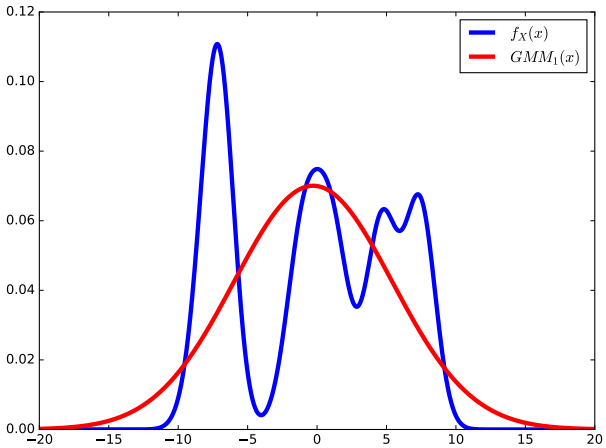
$$\boldsymbol{\mu}_c^+ = \boldsymbol{\mu}_c + \boldsymbol{\varepsilon}$$
$$\boldsymbol{\mu}_c^- = \boldsymbol{\mu}_c - \boldsymbol{\varepsilon}$$

  to obtain an initial $2G$-components GMM

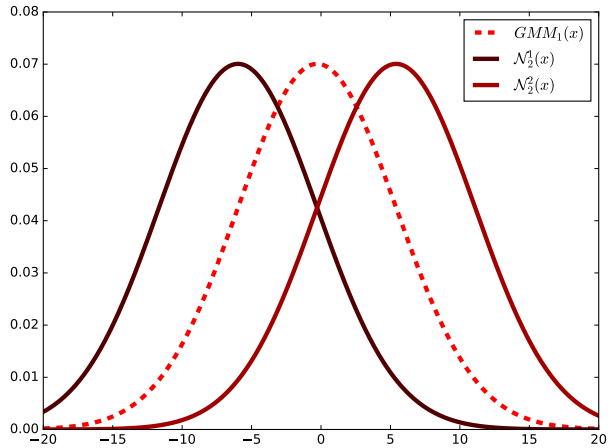
- Run the EM algorithm until convergence for the $2G$-components GMM
- Iterate until the desired number of Gaussians is reached

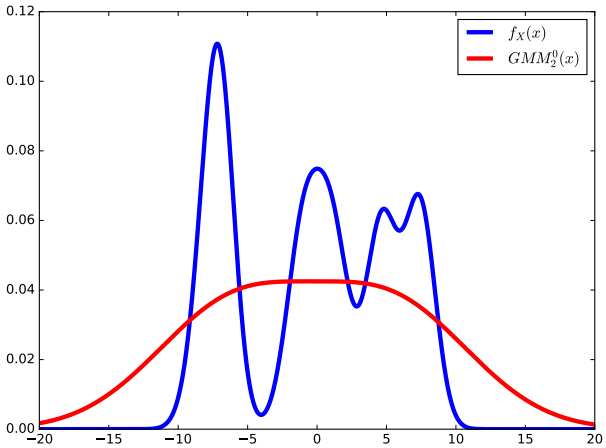
A good value for $\boldsymbol{\varepsilon}$ can be a displacement along the principal eigenvector of the covariance matrix $\boldsymbol{\Sigma}_c$

Gaussian Mixture Models

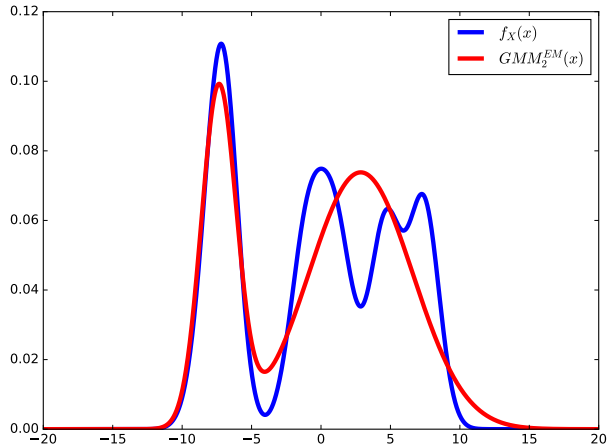


Split

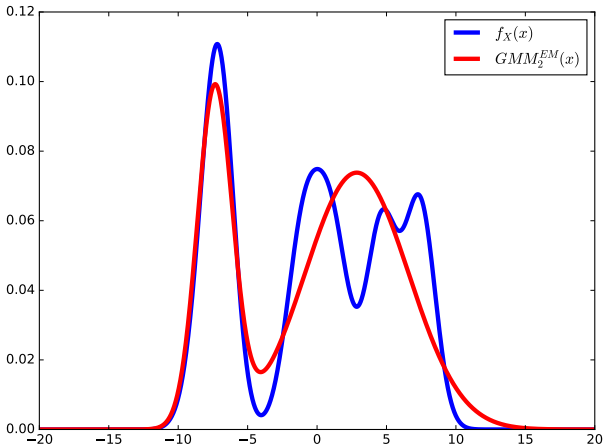




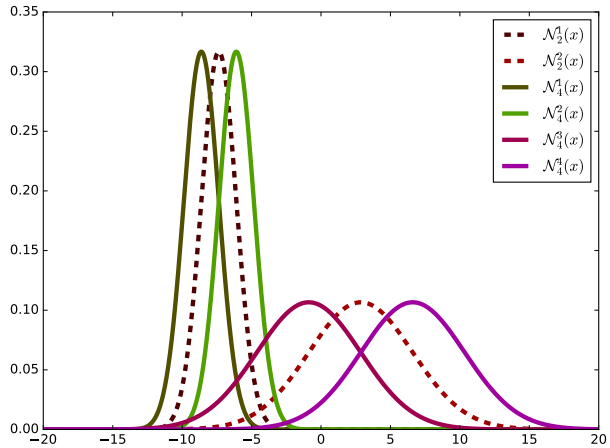
EM

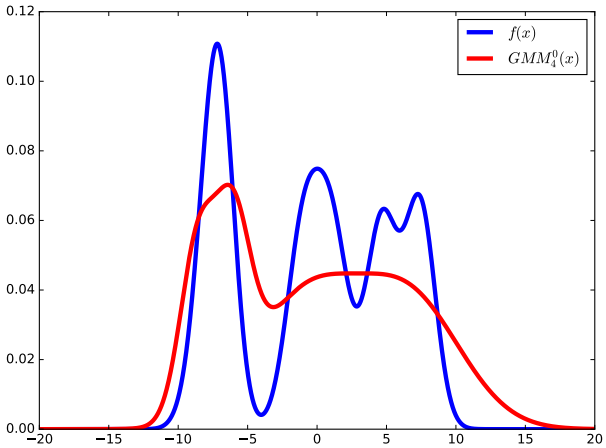


Gaussian Mixture Models

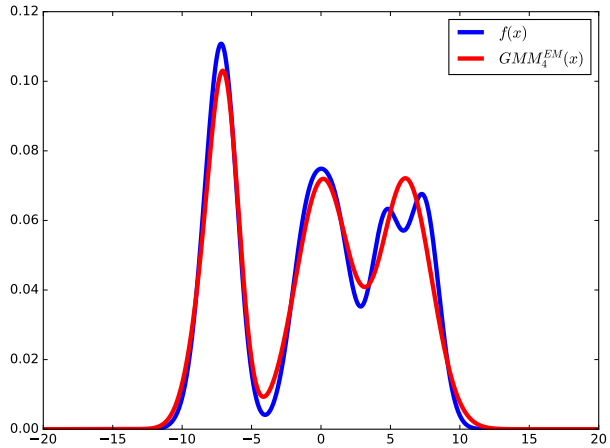


Split





EM



Gaussian Mixture Models

Problem: what is the “right” number of Gaussians?

If we increase the number of components, the likelihood will increase

We cannot choose based on likelihood alone

Several criteria, more or less successful, have been proposed (AIC, BIC)

We can also resort to cross-validation

Gaussian Mixture Models

We also need to pay attention to degenerate models

As we said at the beginning, the log-likelihood for a GMM, as long as we have at least two components, is unbounded

The EM algorithm will usually find local maxima that are well-behaved, however, especially if we have too many components, we may obtain degenerate models which cause numerical issues

Some heuristics can be used to force models to be well-behaved (e.g. impose minimum values for the eigenvalues of the covariance matrices, tie the covariance of different components)

We can also modify our initialization so that the algorithm may end up in a different local maximum

Score calibration and fusion

Sandro Cumani

sandro.cumani@polito.it

Politecnico di Torino

Score calibration

We have seen that, in many cases, classifiers output scores that have a weak, or no probabilistic interpretation, i.e. they do not represent LLRs for the application (evaluation) population

- SVMs — scores have no probabilistic interpretation
- Regularized logistic regression — strong regularization adversely influence the probabilistic interpretation of the score
- Overfitting or distribution mismatch may affect even generative models

Decisions made with the classifier scores may be sub-optimal (miscalibration)

Score calibration (binary tasks)

For binary problems we have seen that decision rules take the form of a comparison of the classifier score with a threshold

To reduce miscalibration we can adopt different strategies

We can use a validation set to find a (close-to) optimal threshold for a given application

- We need to know the target application when estimating the threshold

Alternatively, we can look for functions that transform the classifier scores into approximately well-calibrated LLRs, so that optimal decisions can be obtained for different applications

- Application independent (to some degree)
- Prior knowledge of the application can still help for some models, but a rough estimate of possible applications is often sufficient

Score calibration (binary tasks)

Let's consider the second approach

We want to compute a transformation function f that maps a classifier score s into a well-calibrated score $s_{cal} = f(s)$

We consider monotone functions f, as to preserve the fact that higher non-calibrated scores should favor class $\mathcal{H}_T$ and lower non-calibrated scores should favor class $\mathcal{H}_F$

- Isotonic regression
- Prior-weighted logistic regression (with affine scoring function)
- Generative score models (with constraints)

Score calibration (binary tasks)

Isotonic regression

- Non-linear, monotonic transformation that provides optimal calibration for the data it's trained on
- Piece-wise non-linear, may require some sort of interpolation for unseen scores
- Does not allow extrapolating outside of training scores range
- Expensive to evaluate when the calibration training set is large

Score calibration (binary tasks)

Score models

- Require assumptions on the calibration transformation (e.g. affine mapping) or on the distribution of class scores
- Estimated over a training set, but allow for extrapolation outside of training score ranges
- Typically, fast to evaluate
- May provide good fit only for a small range of operating points

Example: Prior-weighted logistic regression

Score calibration (binary tasks)

Calibration: prior-weighted logistic regression

We consider the non-calibrated scores as 1-D feature vectors

We assume an affine mapping from non-calibrated scores to calibrated scores

$$f(s) = \alpha s + \gamma$$

Since $f(s)$ should produce well-calibrated LLRs, $f(s)$ can be interpreted as the log-likelihood ratio for the two class hypotheses

$$f(s) = \log \frac{f_{S|C}(s|\mathcal{H}_T)}{f_{S|C}(s|\mathcal{H}_F)} = \alpha s + \gamma$$

The class posterior probabilities for prior $\tilde{\pi}$ correspond to

$$\log \frac{P(C = \mathcal{H}_T|s)}{P(C = \mathcal{H}_F|s)} = \alpha s + \gamma + \log \frac{\tilde{\pi}}{1 - \tilde{\pi}} = \alpha s + \beta$$

Score calibration (binary tasks)

We can employ the (non-regularized) prior-weighted logistic regression model to learn the model parameters α, β from a set of training scores (we will refer to the set of samples employed to estimate the calibration parameters as calibration training set in the following)

In practice, we are treating socres as if they were 1-D feature vectors

As for model training and validation set, also the calibration set should be an independent dataset that does not overlap with neither the model training nor the validation set (we will see later how to effectively split the data)

Score calibration (binary tasks)

The calibration transformation corresponds to the transformation of a prior-weighted logistic regression model

Once we have estimated α and β, the calibrated score $f(s)$ can be computed as

$$f(s) = \alpha s + \gamma = \alpha s + \beta - \log \frac{\tilde{\pi}}{1-\tilde{\pi}}$$

Note that we have to specify a prior $\tilde{\pi}$, and we are still effectively optimizing the calibration for a specific application $\tilde{\pi}$. However, often this approach will provide good calibration for a wider range of different applications similar to the target one

We can also modify the prior-weighted logistic regression objective to compute directly α and γ:

$$R(\alpha,\gamma) = \sum_i w_i \log\left(1 + e^{-z_i\left(\alpha s+\gamma+\log\frac{\tilde{\pi}}{1-\tilde{\pi}}\right)}\right), \quad w_i = \begin{cases} \tilde{\pi}/n_T & \text{if } z_i = +1 \\ (1-\tilde{\pi})/n_F & \text{if } z_i = -1 \end{cases}$$

Score calibration (binary tasks)

Alternatively to logistic regression, we can employ generative models to estimate calibrated LLRs

We model the class-conditional score distribution for the two classes

$$S|\mathcal{H}_T \ , \quad S|\mathcal{H}_F$$

For example, we can employ a Gaussian model to estimate the two densities

$$f_{S|\mathcal{H}_T}(s) = \mathcal{N}(s|\mu_T, v_F) \ , \quad f_{S|\mathcal{H}_F}(s) = \mathcal{N}(s|\mu_T, v_F)$$

and the calibration transformation is given by the LLR for the score model

$$f_{cal}(s) = \log \frac{f_{S|\mathcal{H}_T}(s)}{f_{S|\mathcal{H}_F}(s)}$$

Tied models $v_T = v_F = v$ are usually employed, as they result in monotonic transformations

Score calibration

In all cases, we need a calibration set to estimate the transformation

The calibration set must differ from the validation / evaluation sets (otherwise we would get biased results)

Typically, two scenarios:

1. Miscalibration due to non-probabilistic scores, or to overfitting or underfitting models, but evaluation and training populations are similar: the calibration set can be extracted from the training set material

   In this case, calibration allows recovering a probabilistic interpretation of the scores, and, since we are training on 1-D data, the risk of overfitting is drastically reduced (and typically we don't need to add regularization to the calibration model objective function)

Score calibration

Typically, two scenarios:

2. Miscalibration due to mismatch between training and the application / evaluation population: the calibration set (and in this case the calibration validation set as well, if we want to assess the goodness of the calibration) should mimic the application population

   We need to collect data similar to the application population

   The amount of required matching data, however, is usually small compared to the amount required to train a complete classifier

   Some models (e.g. Gaussian) can be extended to exploit unlabeled samples (unsupervised or semi-supervised training), which are less expensive to acquire

Score calibration

A note on projects: our use-case falls in the first scenario

The calibration training and validation sets can be taken from the training set we employed up to now (the evaluation set, that will be disclosed at the end of the course, cannot be used to estimate any part of the model)

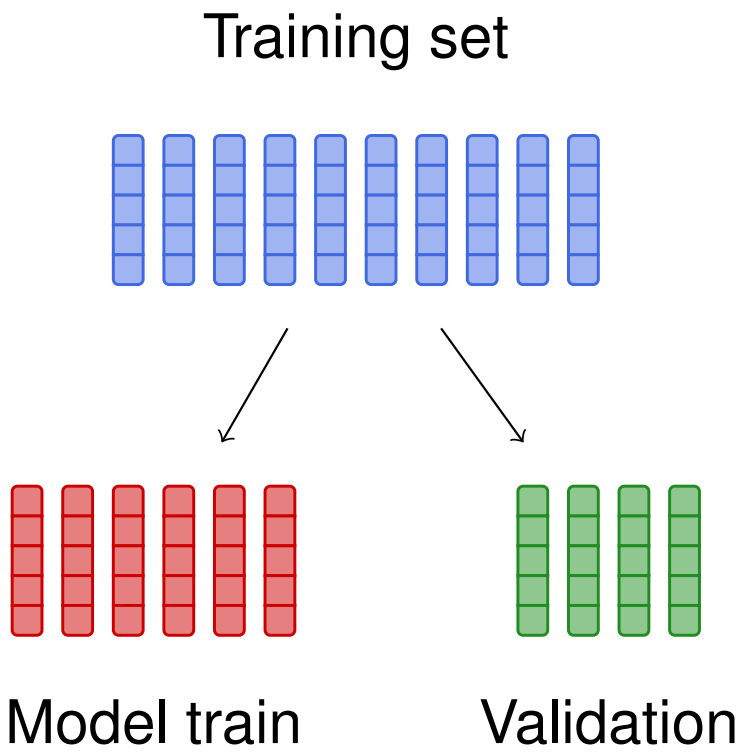
Score calibration

The approach we followed up to now would require that we split the training material in 3 parts:

- Model training
- Calibration training set (samples that are scored with the trained classifier and are used to compute calibration parameters)
- (Calibration) validation set (samples that are scored with the trained model and whose scores are calibrated with the calibration model, used to evaluate the performance of the complete system, including the calibration model)

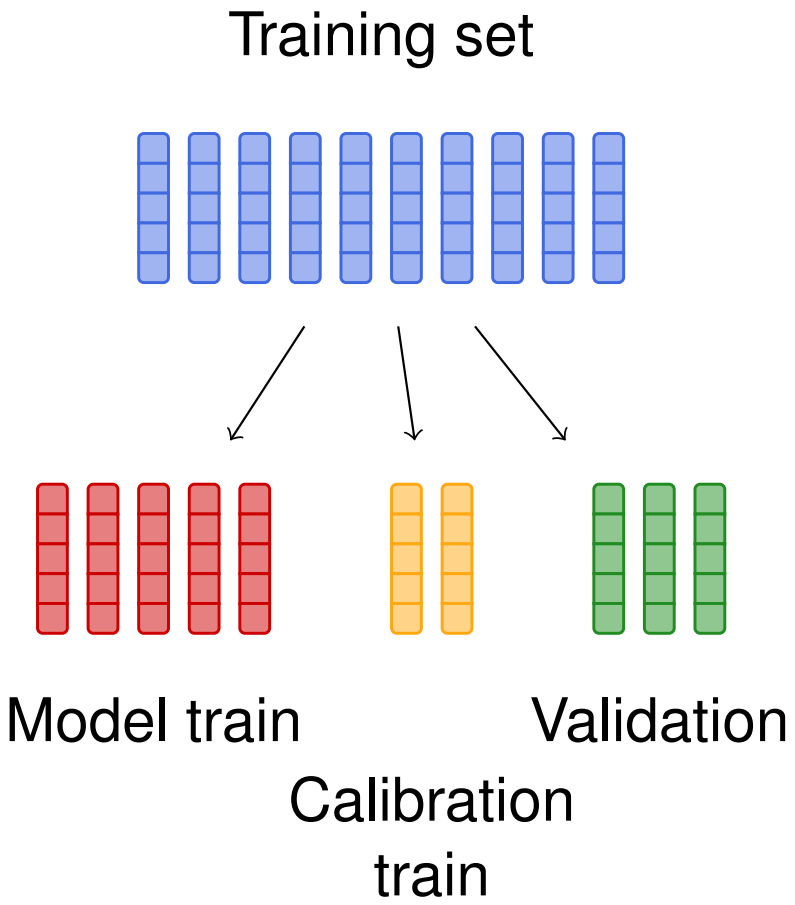
Score calibration

Current set-up (without calibration dataset):



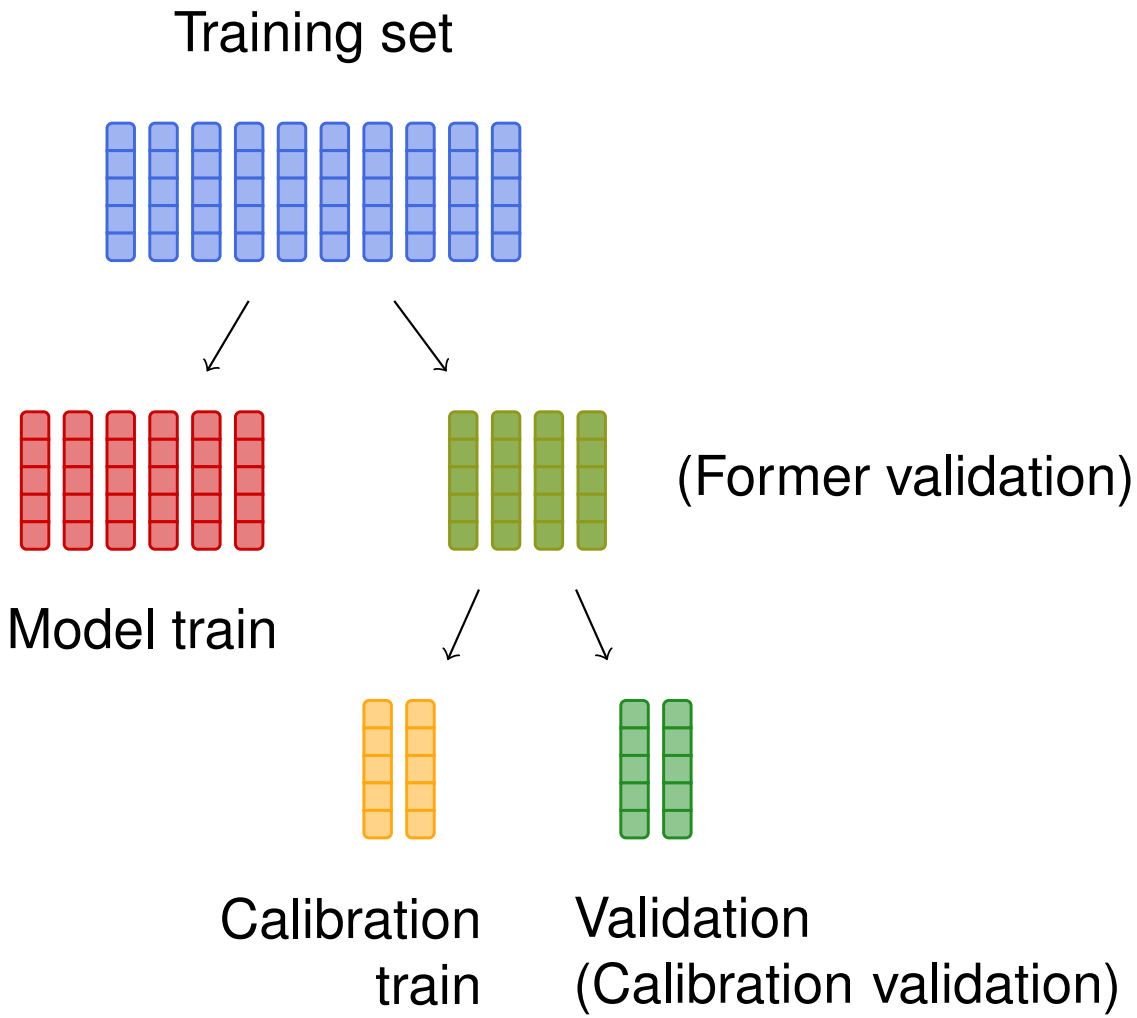
Score calibration

Train - calibration - validation:



Score calibration

Splits can be performed in several ways, for simplicity, we assume that our calibration training and validation sets are extracted from our former validation set (this allows us to re-use models we already trained)



Score calibration

Issue: the calibration and validation sets become both smaller

If we enlarge the calibration set, our evaluation metrics computed over the validation set become less reliable

If we reduce the calibration set, our calibration model becomes less reliable

In general, all three partitions would benefit from more data (with the calibration partition being usually the least critical)

A possible approach to address the limited amount of data is to resort to K-fold cross-validation

K-fold

Single-fold partitions the dataset in separate sets

K-fold extends the approach by performing K times the split, in a way that allows us to exploit all data both for model / calibration training and evaluation

We start considering a simple set-up, where we need two partitions:

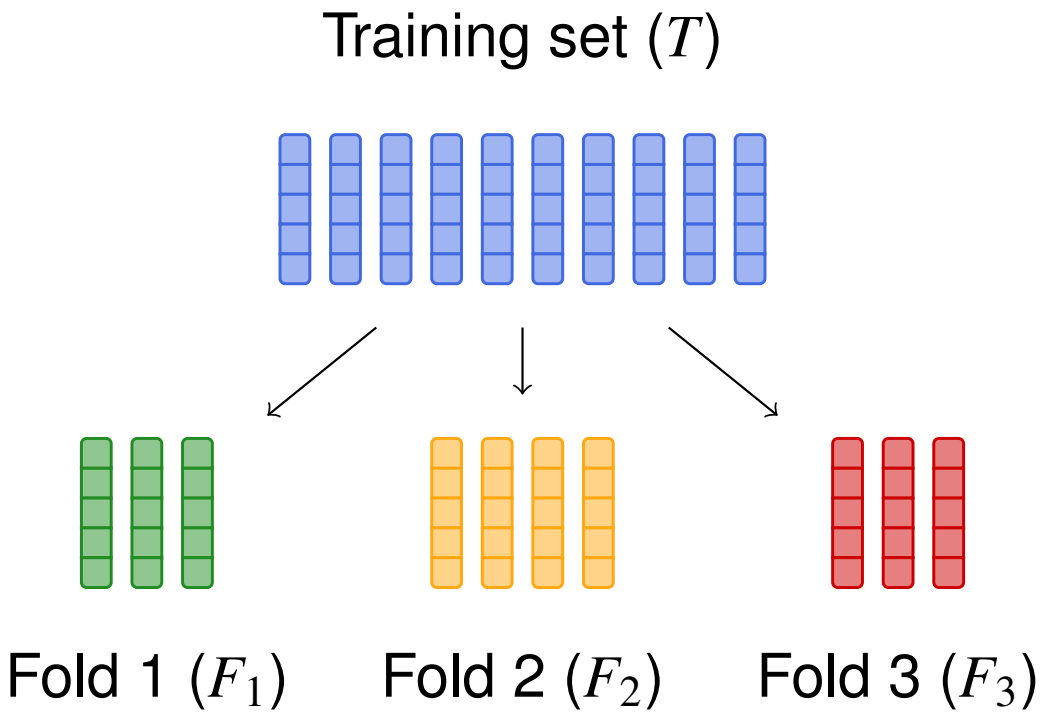
- these could be model training and validation sets extracted from a training dataset
- these could be the calibration and actual validation sets extracted from our former validation set

In both cases, we would like to use the data to train some model (classifier or calibration model) and to evaluate its performance

K-fold

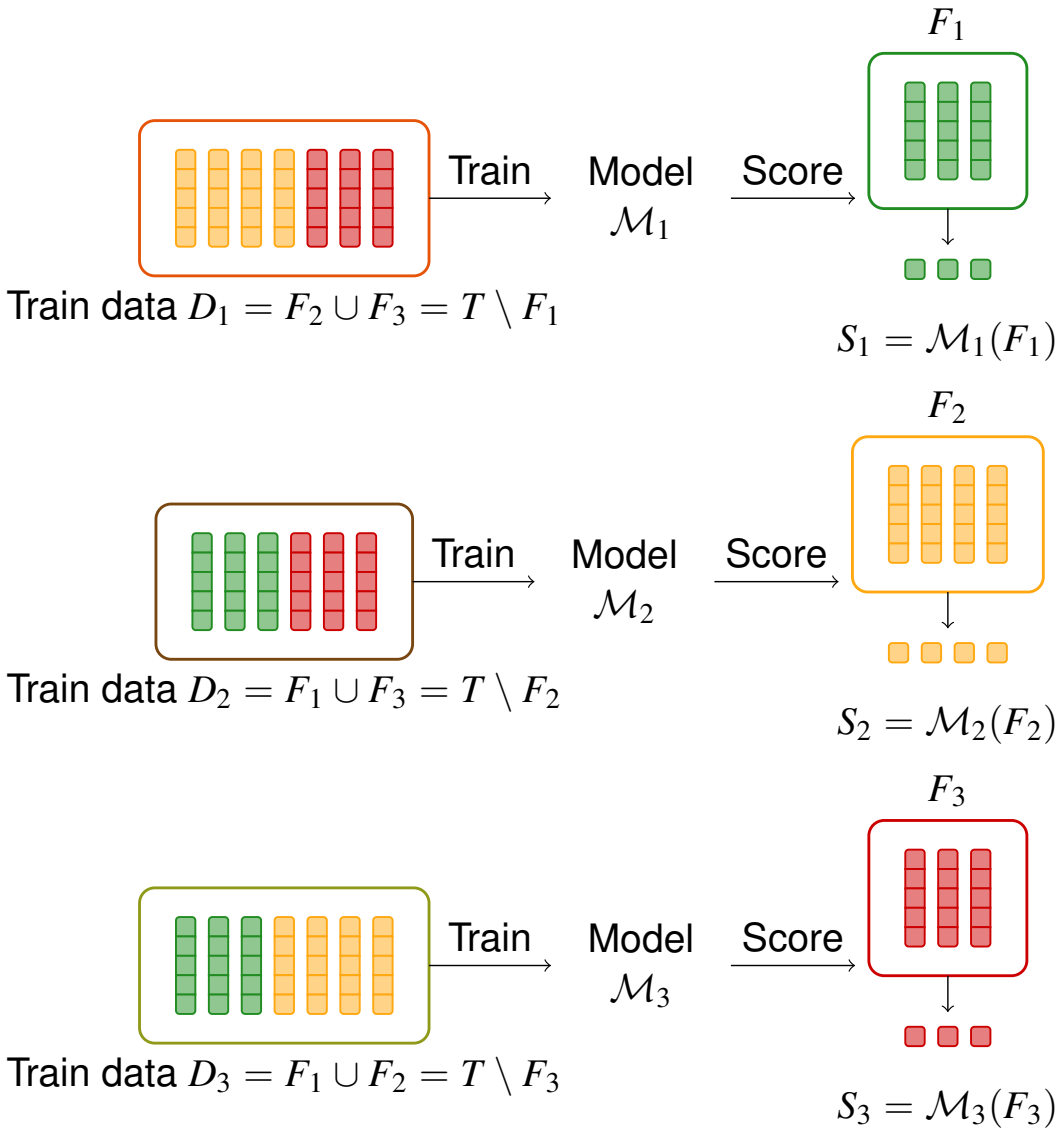
K-fold: split the dataset in K partitions (usually with similar amounts of samples, keeping the same class ratios as the original set)

With K = 3:



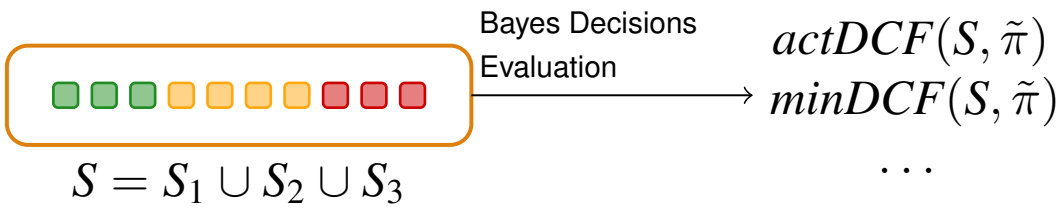
K-fold

Iteratively train using K-1 folds, and scoring the left-out fold:



K-fold

Pool together the scores of the different folds and evaluate the desired metric

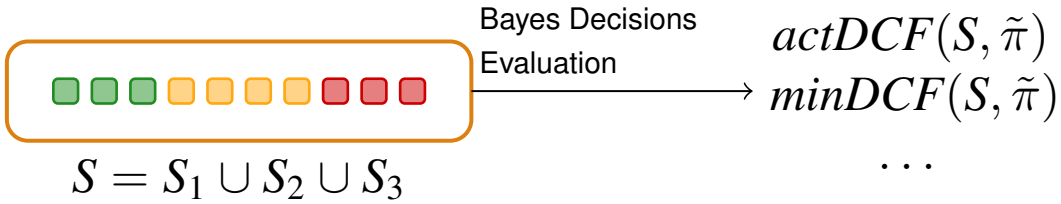


Use the chosen metric to perform model selection / hyperparameter tuning for each model

NOTE (1): all the K models $\mathcal{M}_1 \ldots \mathcal{M}_K$ must be trained with the same set-up (i.e., same pre-processing, same values for the hyperparameters, ...)

K-fold

Pool together the scores of the different folds and evaluate the desired metric



Use the chosen metric to perform model selection / hyperparameter tuning for each model

NOTE (2): For some metrics, including minDCF, it's essential that we compute the metric over the pooled scores (i.e., over the set obtained by merging the scores of all folds)

- Averaging the metric computed over each fold would lead to biased results (and is thus wrong)
- We need to use the same threshold for all scores

K-fold

Once we have selected the approach (model selection) and tuned the model parameters, we still need to choose which model to use in practice

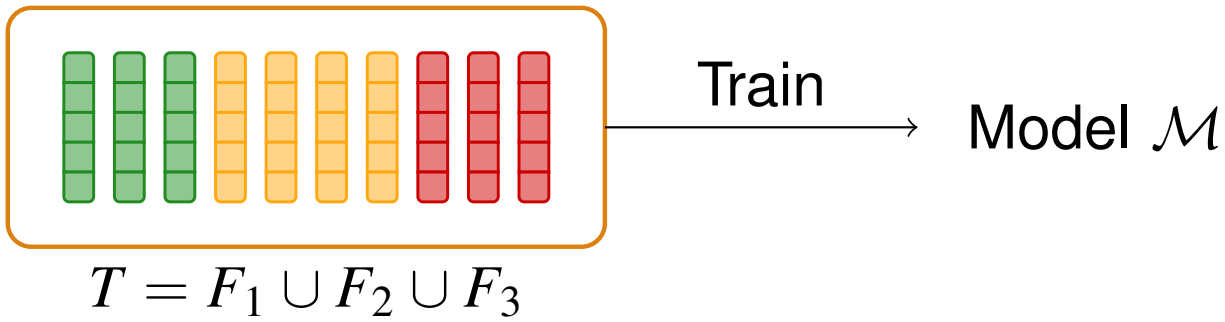
We have trained K models, all of them using a subset of the data

We cannot chose any of them based on their performance on the corresponding validation fold - fold data are different, and the comparison would then make little sense

Moreover, in many cases increasing the training set size is beneficial, but each model was trained using only approximately a fraction $\frac{K-1}{K}$ of the training set

K-fold

Solution: we train one more model over the whole training set, using the method we selected and the hyperparameters we estimated in the previous step



The model $\mathcal{M}$ will then be used to compute the scores for the application data

K-fold

Note that model selection and hyperparameter tuning has been performed using models that are different from $\mathcal{M}$

The selected values may not generalize well for the final model (e.g., the optimal regularization parameters λ or C of logistic regression or SVM may depend on the dataset size)

Furthermore, each model $\mathcal{M}_i$ may be affacted by different levels of miscalibration

Ideally, to minimize these issue we want each model $\mathcal{M}_i$ to be as similar as possible to the final model $\mathcal{M}$

K-fold

Leave-one-out (LOO): K-fold with K equal to the number of training samples N

Each time, we remove a single sample from training, and we score it with a model trained over all remaining samples

We train each model with $N - 1$ samples

- The models will be usually very similar
- The pooled validation set consists of N samples obtained from similar models

K-fold

Unfortunately, leave-one-out may require training an excessively large number of models (we need to train N times a model over $N-1$ samples

We choose K as large as we can afford

- Large values of K: more robust analysis, more time required
- Small values of K: less robust analysis, but faster

K-fold

We can extend the approach to deal with our three-set partitioning using a two-step approach

First step: apply K-fold to train the classifier

- Train K classifiers, each without fold k (model $\mathcal{R}_k$)
- Score each fold k with model $\mathcal{R}_k$
- Train a classifier $\mathcal{R}_{\mathcal{F}}$ over the whole training set (we will need this later)
- Pool the scores of each fold to obtain a score set, that we will use as calibration set

K-fold

Second step: apply K-fold over the calibration scores, using a different randomized shuffle

- Train K calibration models $\mathcal{C}_k$
- Train a calibration model over all pooled scores $\mathcal{C}_\mathcal{F}$ (we will need this later)
- Calibrate the scores of each fold k with model $\mathcal{C}_k$
- Pool the calibrated scores and evaluate the model performance over the pooled scores (to choose the best configuration)

Chose the classification system according to pooled scores results

Classify the application samples: apply classification model $\mathcal{R}_\mathcal{F}$ for the chosen system followed by its calibration model $\mathcal{C}_\mathcal{F}$

K-fold

For the course project you can use a simple split for model training

In the last laboratory you will be asked to apply K-fold for *the calibration part only*

You will have to apply the K-fold procedure for the estimation of a calibration model from the scores of the validation set you used up to now

Take the validation scores that you computed, split them in K-folds, train K calibration models, score the K folds with the calibration models, pool together the scores and evaluate the effectiveness of the calibration transformation

Score-level fusion

In many cases we may have competing models that achieve similar performance

In some cases, even when performance is significantly different, different classifiers may be able to extract different information from the feature vectors

Can we combine these models to further boost our classification performance for a given test sample?

Score-level fusion

Simple strategies exploit majority-voting schemes. Whenever we need to classify a sample $\boldsymbol{x}$:

- Compute the prediction of each of the m considered classifier $p_1 \dots p_m$ for sample $\boldsymbol{x}$
- Assign the label that is selected the most by the different models

While sometimes effective, this approach has some limitations:

- Requires tie-breaking rules (which becomes critical when we have only 2 recognizers)
- Does not account for the strength that each classifier gives towards each labeling hypothesis

Score-level fusion

Consider 3 classifiers that provide, for a sample $\boldsymbol{x}$, the scores $s_1 = 10.0, s_2 = -0.1, s_3 = -0.1$ (binary task, well calibrated scores)

- If our threshold is 0, then majority voting would label the sample as class $\mathcal{H}_F$
- The scores tell us that two systems are very uncertain, with a slight bias towards class $\mathcal{H}_F$, but the first one is very confident in favor of class $\mathcal{H}_T$

Score-level fusion

If our models provide scores as output, then we can try combining the scores for a given sample

For example, we may try computing the average or the sum of the scores

- If our models provide LLRs, and the class-conditional likelihoods provided by the systems were independent, then the sum would be correct (cfr. naive Bayes models)— however, the different systems usually employ the same or strongly correlated features
- If our models are strongly correlated the sum becomes incorrect, as some systems are providing only partial (or even no) additional information

Score-level fusion

We can consider weighting the contribution of the different systems. Given scores $s_1 \dots s_m$ for a test sample $\boldsymbol{x}$, we can compute a "fused" score as

$$s_{fused} = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3 + \cdots + \alpha_m s_m + \gamma$$

We have the problem of estimating the weights α_i and the bias term γ

If we stack the scores and the weights as

$$\boldsymbol{s} = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix}, \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}$$

the fused score can be represented as

$$s_{fused} = \boldsymbol{\alpha}^T \boldsymbol{s} + \gamma$$

Score-level fusion

The output score s_{fused} is obtained as an affine transformation of the score "feature" vector $\boldsymbol{s}$

We can then employ a method like logistic regression to estimate the weights $\boldsymbol{\alpha}$ and the bias term γ (if we want a LLR-like fused scores we need to compensate the logistic regression model parameters for the training prior log-odds as in the calibration case to obtain γ)

This is similar to calibration (and typically provides calibrated scores as output), but rather than 1-D samples we now have m-dimensional vectors, containing the scores of m classifiers for each sample

We can exploit the calibration partition of our set to estimate the score fusion weights

If we had a single system, we would obtain exactly the calibration procedure we analyzed earlier

Multiclass calibration and fusion

For multiclass problems the optimal decisions cannot be represented anymore as a comparison of a score with a threshold

In this case, we have seen that it's more complex to separate the contributions to the cost due to poor discrimination and poor calibration

Calibration methods can be extended to calibrate also scores of multiclass recognizers, so that they can be interpreted as class-conditional log-likelihoods that can then be combined with application priors and costs to obtain optimal decisions

Typically, multiclass logistic regression models are employed to estimate calibration or fusion weights and biases for the different classes and classifiers