Parametric Estimation: from Mixture Models to Competitive Neural Nets

Edmondo Trentin

January 25, 2019
The Estimation Problem

In presenting Bayes decision theory we have been reasoning under the assumption that both the form of $p(x|\omega_i)$ and the parameters $\mu_i$ and $\Sigma_i$ were known, s.t. for each $x$ we could compute:

$$g_i(x) = \log p(x|\omega_i) + \log P(\omega_i)$$  

(1)

or, equivalently:

$$g_i(x) = w_i^t x + b_i$$  

(2)

If the parameters $\Theta = (\mu, \Sigma)$ are NOT known, a technique for their estimation is needed.

*Parametric estimation*: we search for $\Theta = (\mu, \Sigma)$ that characterizes $p(x|\omega_i)$ in a suitable manner.

*Non-parametric Estimation*: for instance, we search for suitable values of $w_i$ and $b_i$, forgetting the assumption on the Normal form of $p(x|\omega_i)$. 

Supervised Parametric Estimation
Let us assume \( c \) samples (a.k.a. the training set) \( Y_1, \ldots, Y_c \) are given, where \( Y_i = \{y_{i,1}, \ldots, y_{i,n_i}\} \) and the \( y_{i,k} \in \mathbb{X} \) are independently and identically distributed (iid) according to \( p(y|\omega_i) \).

We assume that the form of \( p(y|\omega_i) \) is known and uniquely determined by a parameter vector \( \Theta \) (we will write \( p(y|\omega_i, \Theta) \)).

**Problem**: exploit the information encapsulated within the data \( Y_1, \ldots, Y_c \) to estimate \( \Theta_1, \ldots, \Theta_c \).

**Simplification**: we assume that \( \Theta_i \) and \( \Theta_j \) are functionally independent if \( i \neq j \).

Then, the problem reduces to solving \( c \) separate and independent problems like this: "given the data \( Y = \{y_1, \ldots, y_n\} \) iid according to \( p(y|\Theta) \), find the estimate of \( \Theta = (\theta_1, \ldots, \theta_p) \)."
Given the **unlabeled data** \( Y = \{ y_1, ..., y_n \} \), iid according to an **unknown pdf** \( p(y) \), we assume a **parametric model** \( p(y|\Theta) \) of \( p(y) \) and we search for an estimate of \( \Theta = (\theta_1, ..., \theta_p) \) that respects the nature of the data at hand.

- It is seen that, for all intents and purposes, there is no technical (nor, algorithmic) difference between the unsupervised scenario and the aforementioned supervised scenario (once the latter is split into independent, class-specific sub-tasks).
Given $Y = \{y_1, \ldots, y_n\}$, due to the iid assumption we can write:

$$p(Y|\Theta) = \prod_{k=1}^{n} p(y_k|\Theta)$$

This function, called likelihood of $\Theta$ given $Y$, is a function of $\Theta$ (since the data are given).

The maximum likelihood (ML) estimate $\hat{\Theta}$ is the one which maximizes $p(Y|\Theta)$. 
Let us try to devise some **necessary conditions** that \( \hat{\Theta} \) must satisfy. First, let us focus on the logarithm (**log-likelihood**):

\[
l(\Theta) = \log p(Y|\Theta) = \sum_{k=1}^{n} \log p(y_k|\Theta) \tag{4}
\]

such that

\[
\nabla_{\Theta} l(\Theta) = \sum_{k=1}^{n} \nabla_{\Theta} \log p(y_k|\Theta) \tag{5}
\]

with the necessary conditions:

\[
\nabla_{\Theta} l(\hat{\Theta}) = 0 \quad (p \text{ equations}) \tag{6}
\]

Depending on the form you chose for \( p(y|\Theta) \), equation (6) may or may not allow for a unique, closed-form solution.
Maximum-Likelihood estimation of Mixture Densities
Let us consider the (unlabeled) sample $\tau = \{x_1, ..., x_n\}$ iid according to $p(x|\Theta) = \sum_{j=1}^{c} P(\omega_j) p(x|\omega_j, \Theta_j)$. The pdf $p(x|\Theta)$ is known as the mixture density, the pdfs $p(x|\omega_j, \Theta_j)$ are the component densities and $P(\omega_j)$ is the $j$-th mixing parameter.

Let us assume that:

- $c$ and $P(\omega_i)$ (for $i = 1, \ldots, c$) are known
- the form of $p(x|\omega_j, \Theta_j)$ is known
- $p(x|\Theta)$ is identifiable, i.e.
  \[ \Theta \neq \tilde{\Theta} \Rightarrow \exists x \in X : p(x|\Theta) \neq p(x|\tilde{\Theta}) \]
- $\Theta_i$ is functionally independent of $\Theta_j$ if $i \neq j$

Goal: estimate $\Theta$ relying on $\tau$. 
Maximum likelihood (ML) estimation

\[ p(x|\Theta) = \sum_{j=1}^{c} P(\omega_j)p(x|\omega_j, \Theta_j) \quad (7) \]

Likelihood:

\[ p(\tau|\Theta) = \prod_{k=1}^{n} p(x_k|\Theta) \quad (8) \]

The ML estimate \( \hat{\Theta} \) maximizes \( p(\tau|\Theta) \).

Log-Likelihood:

\[ l = \sum_{k=1}^{n} \log(p(x_k|\Theta)) \quad (9) \]

Assuming that \( p(x|\Theta) \) is differentiable, we have:

\[ \nabla_{\Theta_i} l = \sum_{k=1}^{n} \frac{1}{p(x_k|\Theta)} \nabla_{\Theta_i} \left\{ \sum_{j=1}^{c} P(\omega_j)p(x_k|\omega_j, \Theta_j) \right\} \quad (10) \]
Bayes theorem yields:

\[
P(\omega_i|x_k, \Theta) = \frac{p(x_k|\omega_i, \Theta)P(\omega_i|\Theta)}{p(x_k|\Theta)} = \frac{p(x_k|\omega_i, \Theta_i)P(\omega_i)}{p(x_k|\Theta)}
\]

that is, since $\Theta_i$ and $\Theta_j$ are functionally independent if $i \neq j$:

\[
\nabla_{\Theta_i} l = \sum_{k=1}^{n} \frac{1}{p(x_k|\Theta)} \nabla_{\Theta_i} \{ P(\omega_i) p(x_k|\omega_i, \Theta_i) \}
\]

\[
= \sum_{k=1}^{n} \frac{P(\omega_i)}{p(x_k|\Theta)} \nabla_{\Theta_i} p(x_k|\omega_i, \Theta_i)
\]

\[
= \sum_{k=1}^{n} \frac{P(\omega_i|x_k, \Theta)}{p(x_k|\omega_i, \Theta_i)} \nabla_{\Theta_i} p(x_k|\omega_i, \Theta_i)
\]

\[
= \sum_{k=1}^{n} P(\omega_i|x_k, \Theta) \nabla_{\Theta_i} \log \{ p(x_k|\omega_i, \Theta_i) \}
\]

The ML estimate $\hat{\Theta} = \{\hat{\Theta}_1,...,\hat{\Theta}_c\}$ is required to satisfy the constraint $\nabla_{\Theta} l = 0$, i.e.:

\[
\sum_{k=1}^{n} P(\omega_i|x_k, \hat{\Theta}) \nabla_{\hat{\Theta}_i} \log \{ p(x_k|\omega_i, \hat{\Theta}_i) \} = 0 \quad \forall i = 1,...,c
\]
Case study: mixture of Gaussian components, $\Theta_i = \mu_i$

A **Gaussian mixture model (GMM)** is a mixture density having form

$$ p(x|\Theta) = \sum_{j=1}^{c} P(\omega_j) N(x; \mu_j, \Sigma_j) $$

with the following component densities:

$$ p(x|\omega_j, \Theta_j) = \frac{1}{(2\pi)^{d/2}|\Sigma_j|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_j)^t \Sigma_j^{-1} (x - \mu_j) \right\} $$

The corresponding logarithm takes this form:

$$ \log\{p(x|\omega_j, \Theta_j)\} = -\log\{(2\pi)^{d/2}|\Sigma_j|^{1/2}\} - \frac{1}{2} (x - \mu_j)^t \Sigma_j^{-1} (x - \mu_j) $$

whose gradient w.r.t. parameters $\Theta_i$ is:

$$ \nabla_{\mu_j} \log\{p(x|\omega_j, \mu_j)\} = \Sigma_j^{-1} (x - \mu_j) $$
Let us re-write the necessary condition we obtained before:

\[
\sum_{k=1}^{n} P(\omega_j | x_k, \mu_j) \nabla_{\mu_j} \log \{ p(x_k | \omega_j, \mu_j) \} = 0 \quad \forall j = 1, \ldots, c
\]

where we set \( \Theta = \mu = \{ \mu_1, \ldots, \mu_c \} \). Thence:

\[
\sum_{k=1}^{n} P(\omega_j | x_k, \mu) \Sigma_j^{-1} (x_k - \mu_j) = 0
\]

from which:

\[
\sum_{k=1}^{n} P(\omega_j | x_k, \mu) \Sigma_j^{-1} x_k = \sum_{k=1}^{n} P(\omega_j | x_k, \mu) \Sigma_j^{-1} \mu_j
\]

that is:

\[
\mu_j = \frac{\sum_{k=1}^{n} P(\omega_j | x_k, \mu) x_k}{\sum_{k=1}^{n} P(\omega_j | x_k, \mu)} 
\] (13)
Problem: the latter equation does not admit any explicit, closed-form analytical solutions. In point of fact:

\[
\hat{\mu}_j = \frac{\sum_{k=1}^{n} P(\omega_j | x_k, \hat{\mu}) x_k}{\sum_{k=1}^{n} P(\omega_j | x_k, \hat{\mu})} \sum_{k=1}^{n} P(\omega_j | x_k, \hat{\mu})
\]

that is: the formulation is circular (i.e., recursive), since the calculation of \( \hat{\mu}_j \) (left-hand side) relies on \( \hat{\mu}_j \) (right-hand side). We thus resort to the following **iterative algorithm** (gradient ascent):

\[
\left\{ \begin{array}{l}
\hat{\mu}(0) = \text{initial ("arbitrary") estimate} \\
\hat{\mu}_j(t + 1) = \frac{\sum_{k=1}^{n} P(\omega_j | x_k, \hat{\mu}(t)) x_k}{\sum_{k=1}^{n} P(\omega_j | x_k, \hat{\mu}(t))}
\end{array} \right.
\]

to be iterated for \( t = 0, \ldots, t, \ldots, T \).
In a similar manner, if also $\Sigma_j$ and $P(\omega_j)$ are to be estimated from the data $\tau$, we obtain the equations:

\[
\hat{P}(\omega_j) = \frac{1}{n} \sum_{k=1}^{n} \hat{P}(\omega_j|\underline{x}_k, \hat{\Theta}) \tag{16}
\]

\[
\hat{\mu}_j = \frac{\sum_{k=1}^{n} \hat{P}(\omega_j|\underline{x}_k, \hat{\Theta})x_k}{\sum_{k=1}^{n} \hat{P}(\omega_j|\underline{x}_k, \hat{\Theta})} \tag{17}
\]

\[
\hat{\Sigma}_j = \frac{\sum_{k=1}^{n} \hat{P}(\omega_j|\underline{x}_k, \hat{\Theta})(x_k - \hat{\mu}_j)(x_k - \hat{\mu}_j)^t}{\sum_{k=1}^{n} \hat{P}(\omega_j|\underline{x}_k, \hat{\Theta}_j)} \tag{18}
\]
Let us focus again on $\Theta = \mu = (\mu_1, \ldots, \mu_c)$.

$$\hat{\mu}_j(t + 1) = \frac{\sum_{k=1}^{n} P(\omega_j|x_k, \hat{\mu}(t)) x_k}{\sum_{k=1}^{n} P(\omega_j|x_k, \hat{\mu}(t))}$$ (19)

$P(\omega_j|x_k, \hat{\mu}(t))$ is “inversely proportional” to the quadratic Mahalanobis distance $(x_k - \hat{\mu}_j)^T \Sigma_j^{-1} (x_k - \hat{\mu}_j)$.

We simplify things further by limiting ourselves to the Euclidean distance:

$$P(\omega_j|x_k, \hat{\mu}(t)) \approx \begin{cases} 1 & \text{if } j = \text{argmin}_{i=1,...,c} \| x_k - \hat{\mu}_i(t) \|^2 \\ 0 & \text{else} \end{cases}$$ (20)

What we just obtained is a popular clustering algorithm known as **K-Means**:

1. fix initial “arbitrary” (e.g., random) values $\hat{\mu}_1(0), \ldots, \hat{\mu}_c(0)$
2. assign each $x_k$ (for $k = 1, \ldots, n$) to its closest mean $\hat{\mu}_j(t)$
3. re-calculate $\hat{\mu}_j(t)$ for $j = 1, \ldots, c$ applying equation 19 (arithmetic mean of the patterns $x_k$ in cluster $\omega_j$)
4. if $\exists j \in \{1, \ldots, c\} : \hat{\mu}_j(t) \neq \hat{\mu}_j(t - 1)$ then goto 2
The k-Means is the most important instance of clustering algorithm, i.e. of non-parametric unsupervised technique describing the data in terms of their “natural” partitioning into clusters having high internal cohesion (according to certain topological/geometrical properties, rather than probabilistic):
Clustering algorithms are useful for:

1. pinpointing geometric/probabilistic properties of the data (e.g. data thickening, centers of mass, variance, ...)
2. describing the data in a concise fashion (centroids)
3. partitioning the data into $c$ sub-samples $\tau_1, \ldots, \tau_c$ (divide et conquer)
4. pointing out good starting points for the initialization of more sophisticated models (GMMs with Max Likelihood, RBFs, etc.)
5. realizing codebooks of codewords (the centroids, used as representative prototypes), useful for the discretization of continuous features (thus. allowing for the application of discrete models, e.g. histograms)
6. yielding prototypes (i.e. the centroids) of large datasets, thus replacing the original data in complex algorithms like K-NN or Parzen-window
Competitive Neural Networks (CNN)

**Architecture:** in the output layer, the lateral connections between units are inhibitory ($< 0$), while the recursive self-connections are excitatory ($> 0$). The weights $w_i$ that connect the input layer to $i$-th output unit are:

$$w_i = \{\mu_{i1}, ..., \mu_{id}\}$$ (21)

that is, the components of the corresponding centroid (or prototype, or mean) of cluster $\omega_i$. The MAXNET component realizes a **winner take all** strategy: only one output unit (cluster) wins over the others.

**Dynamics:** the input $x$ is projected on $w_1, ..., w_c$ (e.g. $w_i^t x$) and the MAXNET selects the winner unit by turning off the others.
On-line **learning** relies on the following rule:

\[ \Delta w_{ij} = \eta (x_j - w_{ij}) \]  

(22)

**Figure:** CNN learning: before (left) and after (right)

The CNN learning rule is theoretically grounded, being strictly related to the k-Means framework, that is, with the ML estimate of GMMs.
Max. likelihood estimation in GMMs:

$$\mu_i(t + 1) = \frac{\sum_{j=1}^{n} P(\omega_i|x_j, \mu_i(t))x_j}{\sum_{j=1}^{n} P(\omega_i|x_j, \mu_i(t))}$$  \hspace{1cm} (23)

Assumption:

$$P(\omega_i|x, \mu_i) \simeq \begin{cases} 1 & \text{if } \text{dist}(x, \mu_i) = \min_j \{\text{dist}(x, \mu_j)\} \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (24)

K-Means:

$$\mu_i(t + 1) = \frac{1}{n_i(t)} \sum_{j=1}^{n_i(t)} x_j^{(i)}$$  \hspace{1cm} (25)

where $n_i(t)$ is the number of patterns assigned to cluster $\omega_i$, the latter at time $t$ being $\{x_1^{(i)}, ..., x_{n_i(t)}^{(i)}\}$.

Now, let us seek an **incremental (online)** formulation of this algorithm. To this end, let us see what happens when we observe a new pattern (say, $x_{n_i(t)+1}^{(i)}$) which has to be assigned to $\omega_i$. 
Application of the formula for updating the mean $\mu_i$ yields:

$$\mu_i(t + 1) = \frac{1}{n_i(t) + 1} \left\{ \sum_{j=1}^{n_i(t)} x_j^{(i)} + x_{n_i(t)+1}^{(i)} \right\}$$

$$= \frac{n_i(t)}{n_i(t) + 1} \mu_i(t) + \frac{1}{n_i(t) + 1} x_{n_i(t)+1}^{(i)}$$

$$= \mu_i(t) + \frac{1}{n_i(t) + 1} \left\{ x_{n_i(t)+1}^{(i)} - \mu_i(t) \right\}$$

$$= \mu_i(t) + \delta_{t+1} \left\{ x_{n_i(t)+1}^{(i)} - \mu_i(t) \right\}$$

where

$$\delta_{t+1} = \frac{1}{n_i(t) + 1} \rightarrow 0^+ \quad (26)$$

Bearing in mind that the CNN weights are the components of the vectors $\mu_i$, and replacing $\delta_t$ with a constant learning rate $\eta \in \mathbb{R}^+$, we obtain the learning rule:

$$\Delta w_{ij} = \eta(x_j - w_{ij}) \quad (27)$$

where $w_i = \mu_i$ and $x = \{x_1, ..., x_d\}$. 
Fundamental question: may we exploit ML in order to train a feed-forward ANN $\phi(x)$, in an unsupervised manner, to estimate a pdf $p(x)$?  

**Answer:** yes we can (but it’s tricky).

\[
C(\tau, W) = p(\tau|W) \quad (28)
\]

and gradient ascent prescribes:

\[
\Delta w = \eta \frac{\partial C}{\partial w} \quad (29)
\]

If the network is a RBF and we constrain the hidden-to-output weights (i.e., the mixing parameters) to sum to 1, in so doing we realize a GMM (thus, not very interesting). On the other end, if the net is a MLP then the form of the pdf may be more general, but we are faced with the **divergence problem**: $\int \phi(x) dx \gg 1$ and, as long as we increase the likelihood $C(\tau, W) = p(\tau|W)$ via $\frac{\partial C}{\partial w}$, we even have $\int \phi(x) dx \to \infty$ (this is still an open problem).