Data Min Knowl Disc
DOI 10.1007/s10618-005-0033-3

Accelerated EM-based clustering of large data sets

Jakob J. Verbeek · Jan R. J. Nunnink · Nikos Vlassis

Received: 29 July 2004 / Accepted: 14 November 2005

Abstract Motivated by the poor performance (linear complexity) of the EM algorithm in clustering large data sets, and inspired by the successful accelerated versions of related algorithms like k-means, we derive an accelerated variant of the EM algorithm for Gaussian mixtures that: (1) offers speedups that are at least linear in the number of data points, (2) ensures convergence by strictly increasing a lower bound on the data log-likelihood in each learning step, and (3) allows ample freedom in the design of other accelerated variants. We also derive a similar accelerated algorithm for greedy mixture learning, where very satisfactory results are obtained. The core idea is to define a lower bound on the data log-likelihood based on a grouping of data points. The bound is maximized by computing in turn (i) optimal assignments of groups of data points to the mixture components, and (ii) optimal re-estimation of the model parameters based on average sufficient statistics computed over groups of data points. The proposed method naturally generalizes to mixtures of other members of the exponential family. Experimental results show the potential of the proposed method over other state-of-the-art acceleration techniques.

Keywords Gaussian mixtures · EM algorithm · Free energy · kd-trees · Large data sets

1. Introduction

Mixture models provide a rigorous framework for density estimation and clustering, with many applications in machine learning and data mining (McLachlan and Peel, 2000). The

J. J. Verbeek (✉) · J. R. J. Nunnink · N. Vlassis
Informatics Institute, Faculty of Science, University of Amsterdam,
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
e-mail: j.j.verbeek@uva.nl

J. R. J. Nunnink
e-mail: j.r.j.nunnink@uva.nl

N. Vlassis
e-mail: n.vlassis@uva.nl

EM algorithm (Dempster et al., 1977) and the k-means algorithm (Gersho and Gray, 1992) are among the most popular learning algorithms for mixture models. However, both algorithms scale rather poorly for large data sets: each update step requires a complete sweep over all data points, which limits their applicability for large data sets.

Several authors (Omohundro, 1989; Moore, 1999; Moore and Pelleg, 1999; Kanungo et al., 2002) have proposed speedups of these algorithms in which the data are first grouped and statistics of these groups are cached, and then learning iterates only through these statistics instead of the data themselves. In k-means this can be done in an exact way by using geometrical reasoning and a recursive partitioning scheme that allows groups of data to be assigned in bulk to specific components (Moore and Pelleg, 1999; Kanungo et al., 2002). However, in EM the assignment of data points to components is soft, therefore such speedup schemes necessarily involve an approximation (Moore, 1999).

In this paper we propose a variant of the EM algorithm for Gaussian mixtures that offers similar speedups without compromising stability. As in Moore (1999), we first partition the data and cache some statistics in each partition cell. A generalized view of the EM algorithm (Neal and Hinton, 1998) can be used to compute optimal assignments of cells to mixture components (E-step), which are further used to update the mixture parameters (M-step). Both steps have cost that is independent of the size of the data set, and is linear in the number of partition cells. We derive an accelerated EM algorithm that strictly increases in each step a lower bound on the data log-likelihood—independent of the chosen partitioning—ensuring convergence.

To our knowledge, the proposed EM algorithm is the first provably convergent EM clustering algorithm for large data sets. An important feature of our algorithm is that it allows arbitrary data partitioning schemes, without compromising convergence, where related techniques rely on the use of fine partitions and lack convergence guarantees.

In the following, we first briefly review in Section 2 the framework of Gaussian mixtures and the EM algorithm, in Section 3 we describe the main idea of our accelerated EM algorithm, and in Section 4 we discuss several data partitioning schedules. In Section 5 we show how the same principle can be applied to the 'greedy' learning of Gaussian mixtures. We compare with similar work in Section 6, and in Section 7 we show comparative experimental results. We conclude and discuss possible future work in Section 8.

2. Gaussian mixtures and the EM algorithm

A k-component Gaussian mixture for a random vector x in $\mathbb{R}^d$ is defined as the convex combination

$$p(x) = \sum_{s=1}^{k} p(x|s)p(s) \tag{1}$$

of k Gaussian densities $p(x|s)$ which are in turn defined as

$$p(x|s) = (2\pi)^{-d/2}|C_s|^{-1/2}\exp[-(x-m_s)^\top C_s^{-1}(x-m_s)/2], \tag{2}$$

each parameterized by its mean m_s and covariance matrix C_s. The components of the mixture are indexed by the random variable s that takes values from 1 to k, and *p(s)* defines a discrete prior distribution over the components. Given a set $\{x_1, \ldots, x_n\}$ of independent and identically distributed samples from $p(x)$, the learning task is to estimate the parameter vector $\theta = \{p(s), m_s, C_s\}_{s=1}^{k}$ of the k components that maximizes the log-likelihood function $\mathcal{L}(\theta) = \sum_{i=1}^{n} \log p(x_i; \theta)$. Throughout we assume that the likelihood function is bounded from above (e.g., by placing lower bounds on the eigenvalues of the components covariance matrices)in which case the maximum likelihood estimate is known to exist (Lindsay, 1983).

Maximization of the data log-likelihood $\mathcal{L}(\theta)$ can be carried out by the EM algorithm (Dempster et al., 1977). In this work we consider a generalization of EM in which we iteratively maximize a *lower bound* of the data log-likelihood (Neal and Hinton, 1998). In our case, this bound $\mathcal{F}(\theta, Q)$ is a function of the current mixture parameters θ and a factorized distribution $Q = \prod_{i=1}^{n} q_i(s)$, where each $q_i(s)$ corresponds to a data point x_i and defines an arbitrary discrete distribution over s. For a particular realization of s we will refer to $q_i(s)$ as the 'responsibility' of component s for the point x_i. This lower bound, analogous to the (negative) free energy in statistical physics, can be expressed by the following two equivalent decompositions:

$$\mathcal{F}(\theta, Q) = \sum_{i=1}^{n} [\log p(x_i; \theta) - \mathcal{D}(q_i(s) \| p(s|x_i; \theta))] \tag{3}$$

$$= \sum_{i=1}^{n} \sum_{s=1}^{k} q_i(s) [\log p(x_i, s; \theta) - \log q_i(s)], \tag{4}$$

where $\mathcal{D}(\cdot||\cdot)$ denotes Kullback-Leibler divergence between two distributions, and $p(s|x_i)$ is the Bayes posterior over components of a data point x_i. The dependence of p on θ will be throughout assumed, and we will often omit θ.

Since the Kullback-Leibler divergence between two distributions is nonnegative, the decomposition (3) defines indeed a lower bound on the log-likelihood. Moreover, the closer the responsibilities $q_i(s)$ are to the posteriors $p(s|x_i)$, the tighter the bound. In the classical derivation of EM (Dempster et al., 1977), each E-step of the algorithm sets $q_i(s) = p(s|x_i)$ in which case, and for the current value θ^t of the parameter vector, holds $\mathcal{F}(\theta^t, Q) = \mathcal{L}(\theta^t)$.

However, as pointed out in Neal and Hinton (1998), other (suboptimal) responsibilities $q_i(s)$ can also be used in the E-step of the algorithm, provided that $\mathcal{F}$ increases (or at least does not decrease).[1] This also leads to a convergent algorithm that increases in each step a lower bound on the data log-likelihood $\mathcal{L}$. Moreover, it can be shown that (local) maxima of $\mathcal{F}$ are also (local) maxima of $\mathcal{L}$.

For particular values of the responsibilities $q_i(s)$, we can solve for the unknown parameters of the mixture by using (4). It is easy to see that maximizing $\mathcal{F}$ for the unknown parameters of a component s yields the following solutions:

$$p(s) = \frac{\sum_i q_i(s)}{n}, \tag{5}$$

$$m_s = \frac{\sum_i q_i(s) x_i}{np(s)}, \tag{6}$$

$$C_s = \frac{\sum_i q_i(s) x_i x_i^\top}{np(s)} - m_s m_s^\top, \tag{7}$$

where the sums run over all data points ($i = 1, \ldots, n$), which is costly for large n.

[1] This is why we use the more general term 'responsibility' for the distributions q rather than e.g. 'cluster posterior probability'.

3. Locally shared responsibilities

As mentioned above, in the E-step of the EM algorithm we are allowed to assign any responsibilities $q_i(s)$ to the data as long as this increases $\mathcal{F}$. The key idea in our algorithm is to *assign equal responsibilities to groups of data points that are nearby in the input space.* In this manner, we do not need to optimize over n distributions q_i but only over one distribution per group of data points. It turns out that for this optimization only a few averaged sufficient statistics need to be available per group of data points.

Consider a partition $\mathcal{P}$ of the data space into a collection of non-overlapping cells $\{A_1, \dots, A_m\}$, such that each point in the data set belongs to a single cell.[2] To all points in a cell $A \in \mathcal{P}$ we assign the same distribution $q_A(s)$ which we can compute in an optimal way as we show next. Note from (4) that the objective function $\mathcal{F}$ can be written as a sum of local parts $\mathcal{F} = \sum_{A \in \mathcal{P}} \mathcal{F}_A$, one per cell. If we impose $q_i(s) = q_A(s)$ for all data points $x_i \in A$, then the part of $\mathcal{F}$ corresponding to a cell A reads

$$\mathcal{F}_A = n_A \sum_{s=1}^{k} q_A(s) \left[\log \frac{p(s)}{q_A(s)} + \frac{1}{n_A} \sum_{x_i \in A} \log p(x_i|s) \right], \tag{8}$$

where n_A denotes the number of points in A. If we set the derivatives of $\mathcal{F}_A$ w.r.t. $q_A(s)$ to zero we find the optimal distribution $q_A(s)$ that (globally) maximizes $\mathcal{F}_A$:

$$q_A(s) \propto p(s) \exp\langle \log p(x|s) \rangle_A, \tag{9}$$

where $\langle \cdot \rangle_A$ denotes average over all points in A. Such an optimal distribution can be separately computed for each cell $A \in \mathcal{P}$, and only requires computing the average joint log-likelihood of the points in A.

3.1. Speedup using cached statistics

We now show that it is possible to efficiently compute (i) the optimal $q_A(s)$ for each cell A in the E-step and (ii) the new values of the unknown mixture parameters in the M-step, if some statistics of the points in each cell A are cached in advance. The averaging operation in (9) can be written (we ignore the additive constant $-\frac{d}{2}\log(2\pi)$ which translates into a multiplicative constant in (9)

$$\begin{aligned} \langle \log p(x|s) \rangle_A &= -\frac{1}{2} \left[\log |C_s| + m_s^\top C_s^{-1} m_s + \left\langle x^\top C_s^{-1} x \right\rangle_A - 2 m_s^\top C_s^{-1} \langle x \rangle_A \right] \\ &= -\frac{1}{2} \left[\log |C_s| + m_s^\top C_s^{-1} m_s + \mathrm{Trace}\left\{ C_s^{-1} \langle x x^\top \rangle_A \right\} - 2 m_s^\top C_s^{-1} \langle x \rangle_A \right], \end{aligned} \tag{10}$$

from which we see that the mean $\langle x \rangle_A$ and the average outer product $\langle x x^\top \rangle_A$ of the points in A are averaged sufficient statistics for computing the optimal responsibilities $q_A(s)$ in (9). The same statistics can also be used for updating the mixture parameters θ. If we set the

[2] As we discuss in Section 6, our approach straightforwardly generalizes to the case of overlapping cells.

derivatives of $\mathcal{F}$ w.r.t. θ to zero we obtain the update equations

$$p(s) = \frac{\sum_A n_A q_A(s)}{n}, \tag{11}$$

$$m_s = \frac{\sum_A n_A q_A(s) \langle x \rangle_A}{np(s)}, \tag{12}$$

$$C_s = \frac{\sum_A n_A q_A(s) \langle x x^\top \rangle_A}{np(s)} - m_s m_s^\top, \tag{13}$$

in direct analogy to the update Eqs. (5)–(7), with the advantage that the linear complexity in the number of data points has been replaced by a linear complexity in the number of cells of the partition.

Note that the proposed EM algorithm interacts with the data only through the cached statistics of groups of data. Moreover, whatever partition we choose, our algorithm strictly increases in each step a lower bound on the data log-likelihood. In the limit, if we partition all data points into separate cells, the algorithm is guaranteed to converge to a (local) maximum of the data log-likelihood. In effect, our accelerated EM algorithm is a 'maximization-maximization' or 'coordinate ascent' algorithm: both E- and M-steps involve a maximization of the free energy $\mathcal{F}$ (over Q and θ respectively).

Finally, note that the result presented here for Gaussian mixtures generally applies to mixtures of members of the exponential family. This general applicability of the result follows from the facts that for members of the exponential family (i) the expected log-likelihood over a cell of data is given by a linear function of the average sufficient statistics of the cell, and (ii) the maximum likelihood parameter is uniquely determined by the averaged sufficient statistics.

3.2. Further speedup using diagonal covariance matrices

If the covariance matrices are constrained to be diagonal, the amount of sufficient statistics and computation drops considerably since correlations between variables do not need to be estimated. Rather than caching the average outer products $\langle xx^\top \rangle$, in this case we only need to cache the diagonal of this matrix, a vector denoted by $\langle x^2 \rangle$. Analogously we write m_s^2 for the diagonal of $m_s m_s^\top$. Obviously, $\langle x^2 \rangle$ and m_s^2 can be computed without forming the matrices $\langle xx^\top \rangle$ and $m_s m_s^\top$.

In the M-step only the update for the covariance matrix changes; we now set the diagonal of the covariance matrix to the diagonal of the update (13), which can be written as $\sum_A n_A q_A(s) \langle x^2 \rangle_A / (np(s)) - m_s^2$. Also in the computation of the average log-likelihood $\langle \log p(x|s) \rangle$ for a cell, computational savings are obtained. Notably, the trace term can be replaced by the inner product of $\langle x^2 \rangle_A$ and the diagonal of the inverse covariance matrix.[3]

Concluding, by constraining the covariance matrices to be diagonal the amount of computation needed in both the E-step and the M-step (as well as the space needed to store the mixture parameters and the averaged sufficient statistics) becomes linear in the data dimensionality rather than quadratic when using full covariance matrices. This saving is important when fitting Gaussian mixtures to data in a high dimensional spaces, as is e.g. the case when employing Generative Topographic Mapping (Bishop et al., 1998) for data

[3] The inverse covariance matrix is found in linear time since it is diagonal.

visualization. It isapplication dependent whether the independence assumption between the variables within each cluster, as implemented by restricting the covariance matrices to be diagonal, is realistic. In some applications it may be worthwhile to settle for a more restrictive model with diagonal covariance matrices so as to obtain computational savings and reduced storage requirements.

4. Choosing a partition

The analysis presented in the previous sections applies to any partition, as long as averaged sufficient statistics of the data have been stored in the corresponding cells. As we showed above, for any partition we obtain a convergent algorithm that strictly increases in each step a lower bound on the data log-likelihood. Moreover, by refining a given partition, the energy $\mathcal{F}$ cannot decrease, and in the limit (when each data point is in a separate cell) the normal EM algorithm is obtained, and after the E-step $\mathcal{F} = \mathcal{L}$. Clearly, various trade-offs can be made between the computational cost and the approximation quality.

A convenient structure for storing statistics in a way that permits the use of different partitions in the course of the algorithm is a kd-tree (Bentley, 1975; Moore, 1999). This is a binary tree in which the root contains all data points, and each node is recursively split by a hyperplane that cuts through the data points contained in the node. Typically, axis-aligned hyperplanes are used for splitting nodes. In our experiments we used hyperplanes that cut along the bisector of the first principal component of the points in the node, leading to irregularly shaped cells (Sproull, 1991). Hyperplanes based on the principal component allow the kd-tree to capture the clustered data structure at a higher level in the tree. Note that the usual performance deterioration of kd-trees in high dimensional spaces does not apply here directly, since they are not used for search in this work. As in Moore (1999), we store in each node of the kd-tree the average sufficient statistics of all data points under this node. Building the kd-tree and storing statistics in its nodes has cost $O(n \log n)$, but this needs to be done only once at the beginning of the algorithm.

The outer nodes of a given expansion of the kd-tree form a partition $\mathcal{P}$ of the data set. Further expanding the tree means refining a current partition. In our implementations, as heuristic to guide the tree expansion, we employ a best-first search strategy in which we expand the node that leads to maximal increase in $\mathcal{F}$. Note that computing the change in $\mathcal{F}$ involves only a node and its children so it can be done in time linear in the number of outer nodes.

We also need a criterion when to stop expanding the tree, and one could use, among others, bounds on the variation of the data posteriors inside a node like in Moore (1999), a bound on the size of the partition (number of outer nodes at any step), or sampled approximations of the difference between log-likelihood and $\mathcal{F}$. Another possibility, which we adopted in our experiments, is to control the tree expansion based on the performance of the algorithm, that is, we refine a partition only if this (significantly) improves the value of $\mathcal{F}$.

Below we show in pseudocode the proposed accelerated EM algorithm as described in Section 3, together with the partition schedule outlined above. Input is a d-dimensional data set of n points x_i, and output is a Gaussian mixture with k components and parameters $\{p(s), m_s, C_s\}_{s=1}^k$.

1. Build a kd-tree on the data set $\{x_i\}$, and store in each node A the required data statistics $\langle x \rangle_A$ and $\langle xx^\top \rangle_A$ of the points x that are contained in A. The complexity of this step is $O(n \log n)$.

2. Choose an initial configuration of the mixture using k components (several initialization techniques from the literature can be used, e.g., random or using k-means). Choose an initial partition $\mathcal{P}_0$ of the data by expanding the tree to some depth (in our experiments we used depth 2).
3. E-step: For each cell A in the current partition $\mathcal{P}_t$ (that is: each node in the fringe of the expanded tree) compute $q_A(s)$ for each component s of the mixture using (9) and (10). Note that this step requires only the averaged sufficient statistics already cached in the tree, and has complexity $O(k|\mathcal{P}_t|)$ where $|\mathcal{P}_t|$ the size of the current fringe (typically $|\mathcal{P}_t| \ll n$).
4. M-step: For each mixture component k update its parameters using (11)–(13). Set $\mathcal{F}_{\text{old}} = \mathcal{F}$, and compute the new energy $\mathcal{F}$ from (8). Note that this step also uses the average sufficient statistics of the data in each cell, and has also complexity $O(k|\mathcal{P}_t|)$.
5. If $|\frac{\mathcal{F}}{\mathcal{F}_{\text{old}}} - 1| <$ 1e-5 then set $\mathcal{F}_t = \mathcal{F}$, else go back to step 3.
6. Expand one-level-down a single node on the fringe of the tree to create the new partition $\mathcal{P}_{t+1}$: among all nodes in $\mathcal{P}_t$ choose the node that leads to maximal increase of $\mathcal{F}_t$. This step has complexity $O(k|\mathcal{P}_t|)$.
7. If $|\frac{\mathcal{F}_t}{\mathcal{F}_{t-1}} - 1| <$ 1e-5 then stop, else set $t = t + 1$ and go back to step 3.

5. Greedy mixture learning

A recent approach to mixture learning involves building a mixture in a ‘greedy’ manner (Li and Barron, 2000; Sand and Moore, 2000; Vlassis and Likas, 2002; Verbeek et al., 2003)). The idea is to start with a single component (which is trivial to find), and then alternate between adding a new component to the mixture and updating the complete mixture. In particular, given a k-component Gaussian mixture $p_k(x)$ that has converged, the greedy method seeks a new component $\phi(x)$ with mean m_ϕ and covariance C_ϕ, and a mixing weight $a \in (0, 1)$ that maximizes the log-likelihood $\mathcal{L}_{k+1} = \sum_{i=1}^n \log p_{k+1}(x_i)$ of the *two*- component mixture

$$p_{k+1}(x) = (1 - a)p_k(x) + a\phi(x; m_\phi, C_\phi), \tag{14}$$

where $p_k(x)$ is kept fixed. The advantages of greedy mixture learning are: (1) initializing the mixture is trivial, (2) local maxima of $\mathcal{L}$ are easier to escape, and (3) model selection becomes more manageable. Relations of such greedy methods to other machine learning techniques like boosting can be found in Zhang (2002). The greedy approach is somewhat similar to a deterministic annealing (Rose, 1998) approach where components are ‘added’ at phase transitions. However, only the greedy approach is guaranteed to iteratively increase the data log-likelihood under the mixture.

In Verbeek et al. (2003), the search for a good component to add to $p_k(x)$ involves first splitting the data according to their ‘nearest’ (with highest posterior) component, then randomly generating a number of candidate components from the points in each subset, and finally maximizing $\mathcal{L}_{k+1}$ using only the data from the corresponding subset. The same principle can also be applied in the case of pre-partitioned data sets. In particular, in component allocation we divide all cells $A \in \mathcal{P}$ into k disjoint subsets $\mathcal{P}_s$ $(s = 1, \ldots k)$ according to their ‘nearest’ (with highest responsibility) component: $\mathcal{P}_s = \{A \in \mathcal{P} : s = \arg\max_{s'} q_A(s')\}$. Then we generate a component $\phi(x; m_\phi, C_\phi)$ from the data contained in a random subset of cells $\mathcal{S} \subset \mathcal{P}_s$ as

$$m_\phi = \frac{1}{n_\mathcal{S}} \sum_{A \in \mathcal{S}} n_A \langle x \rangle_A, \qquad C_\phi = \frac{1}{n_\mathcal{S}} \sum_{A \in \mathcal{S}} n_A \langle x x^\top \rangle_A - m m^\top, \tag{15}$$

where n_S is the total number of points in $\mathcal{S}$. (Note that both m_ϕ and C_ϕ can be calculated without requiring the data points themselves). Subsequently we update (a, m_ϕ, C_ϕ) in (14) by maximizing a lower bound of $\mathcal{L}_{k+1}$ using only the cells in $\mathcal{P}_s$. (Cells outside $\mathcal{P}_s$ will not contribute significantly to the bound). Let r_A be the responsibility of the new component $\phi(x; m_\phi, C_\phi)$ for any cell $A \in \mathcal{P}_s$, and $1 - r_A$ the responsibility of the old mixture p_k. The free energy (8) for cell A under the two-component mixture (14) then reads

$$\mathcal{F}_A^{k+1} = n_A r_A \left[\log \frac{a}{r_A} + \langle \log \phi(x) \rangle_A \right] + n_A (1 - r_A) \left[\log \frac{1-a}{1-r_A} + \langle \log p_k(x) \rangle_A \right]. \tag{16}$$

Since $\mathcal{F}_A^k$ is a lower bound on $\sum_{x \in A} \log p_k(x)$ which we have already computed from (8), we can replace the latter in (16) to get the bound

$$\mathcal{F}_A^{k+1} \geq n_A r_A \left[\log \frac{a}{r_A} + \langle \log \phi(x) \rangle_A \right] + n_A (1 - r_A) \left[\log \frac{1-a}{1-r_A} + \frac{\mathcal{F}_A^k}{n_A} \right]. \tag{17}$$

In the E-step we compute the optimal r_A for each cell $A \in \mathcal{P}_s$ by setting the derivative of (17) w.r.t. r_A to zero. This gives:

$$r_A = \frac{a \exp\langle \log \phi(x) \rangle_A}{(1-a) \exp\left(\mathcal{F}_A^k / n_A\right) + a \exp\langle \log \phi(x) \rangle_A}, \tag{18}$$

where $\langle \log \phi(x) \rangle_A$ can be computed fast using (10). Similarly, in the M-step we maximize $\mathcal{F}_{\mathcal{P}}^{k+1} = \sum_{A \in \mathcal{P}} \mathcal{F}_A^{k+1}$ using the r_A found in (18). As in Verbeek et al. (2003), we set the responsibility of the new component for all cells outside $\mathcal{P}_s$ to zero, in which case it is not difficult to see that we get the following update equations:

$$a = \frac{\sum_{A \in \mathcal{P}_s} n_A r_A}{n}, \tag{19}$$

$$m_\phi = \frac{\sum_{A \in \mathcal{P}_s} n_A r_A \langle x \rangle_A}{na}, \tag{20}$$

$$C_\phi = \frac{\sum_{A \in \mathcal{P}_s} n_A r_A \langle x x^\top \rangle_A}{na} - m_\phi m_\phi^\top. \tag{21}$$

Note that the sums run over cells in $\mathcal{P}_s \subset \mathcal{P}$. We refer to Nunnink (2003) for more details.

6. Related work

The idea of using a kd-tree structure for accelerating the EM algorithm for learning mixtures from large data sets was first proposed in Moore (1999). In that work in each EM step every node in the kd-tree is assigned responsibility distribution equal to the Bayes posterior of the centroid of the data points stored in the node, i.e., $q_A = p(s|\langle x \rangle_A)$, cf. (9). If there is little variation in the posteriors within a node, which is achieved by having relatively fine partitions, the approximation $q_A = p(s|\langle x \rangle_A)$ will only slightly affect the update in the M-step and therefore this will probably increase the data log-likelihood. However, this is not guaranteed. Also, in Moore (1999) a different tree expansion is computed in each EM step, while as stopping criterion for tree expansion bounds are used on the variation of theposterior probabilities of the data inside a node of the kd-tree (a nontrivial operation that in principle requires solving a quadratic programming problem).

The main advantage of our method compared to Moore (1999) is that our algorithm strictly increases in each step a lower bound of the data log-likelihood by computing the optimal responsibility distribution for each node, thus ensuring stability. Moreover, this optimal distribution is independent of the size, shape, or other properties of the node, allowing us to use even rough partitions. As mentioned above and as demonstrated in theexperiments below, by gradually refining the partition while running the algorithm we can get close to the optima of the log-likelihood in relatively few steps.

Our approach to compute optimal shared responsibilities may also be used in other related accelerated EM algorithms to furnish them with a guarantee to iteratively improve the free-energy bound on the data log-likelihood. In Bradley et al. (1998) an algorithm is proposed that learns the mixture while reading data from disk. During learning some data records are stored in memory while other groups of data records are only stored by their average sufficient statistics, which frees memory space for new records to be readfrom disk. In Bradley et al. (1998) a group A of data points are associated to the mixture components by computing $p(s|\langle x\rangle_A)$, as in Moore (1999).

In McCallum et al. (2000) an approach more similar to that presented here was introduced. The main difference with the method proposed here is that in the former the data is divided in *overlapping* subsets. The data within each subset contributes to the update of one fixed mixture component associated with that subset, and to the other components only in the form of the expected value of the data points in the subset, similar to Moore's suboptimal responsibilities. Our approach of computing optimal shared posteriors can be straightforwardly extended to a collection $\mathcal{P}$ of overlapping subsets ofdata points as follows.[4] For each data point x_i and subset A we introduce an association variable β_{iA}, such that $\sum_A \beta_{iA} = 1$, and $\beta_{iA} = 0$ if x_i is not in subset A. For example, the β_{iA} could be set uniform over all subsets A to which x_i belongs. We can now define a slightly modified lower bound on the datalog-likelihood:

$$\mathcal{F}' = \sum_{i=1}^{n} \left[\log p(x_i) - \sum_{A\in\mathcal{P}} \beta_{iA} \mathcal{D}(q_A(s) \| p(s|x_i)) \right] \tag{22}$$

$$= \sum_{A\in\mathcal{P}} \left[\sum_{i=1}^{n} \beta_{iA} \right] \mathcal{H}(q_A) + \sum_{s=1}^{k} q_A(s) \left[\sum_{i=1}^{n} \beta_{iA} \log p(x_i, s) \right]. \tag{23}$$

The derivations of optimal assignments and parameter re-estimation equations are completely analogous to those presented in Section 3.[5]

A different way to speedup the EM algorithm for mixture learning from large data sets was proposed in Thiesson et al. (2001). The authors propose to divide the data set into several random disjoint subsets of (about) equal size and then to process the data per subset. For each subset, first for all points x_i in the subset the optimal responsibilities $q_i(s)$ are computed as the posteriors $p(s|x_i)$ (E-step), then the parameter estimates are updated by taking into account the newly computed posteriors for the data in the current subset. In this manner, when

4 We only require that each data point is contained in at least one subset.

5 In principle it is also possible, and straightforward, to optimize over the association variables β_{iA}.

computing the responsibilities for the next subset, we already take into account information from the previous subset through the updated parameter estimates. However, the reported speedups that were obtained by this approach are modest; the authors do not report speedups greater than 2.3 (i.e. the time needed until convergence for the standard EM algorithm is 2.3 times greater than the time required by their accelerated algorithm). in the experiments below, we report similar findings.

An approach based on random projections to learn Gaussian mixtures in high dimensional spaces is presented in Dasgupta (1999). To alleviate the large sample requirements for accurate parameter estimation in high dimensional spaces, the data is first randomly projected to a low dimensional linear subspace. The data are clustered in the low dimensional space, and then the result is used to find the mixture in the high dimensional space. Although a thorough theoretical analysis is presented, in comparison to our method the above approach does not directly resolve the computational burden associated with a large number of datapoints, and moreover the approach is limited to learning of Gaussian mixtures.

7. Experiments

We carried out synthetic experiments to evaluate the proposed accelerated EM algorithm for learning a k-component Gaussian mixture, using both the non-greedy and the greedy variant. We compare against the standard EM algorithm, the greedy EM algorithm proposed in Verbeek et al. (2003), and the accelerated EM algorithms described in Moore (1999) and Thiesson et al. (2001).

In our experiments we used synthetic data sets sampled from a randomly chosen k-component Gaussian mixture in d dimensions with a component separation[6] of c. For each data set we built a kd-tree and stored in its nodes the data statistics as explained above.

7.1. EM vs. accelerated EM

In the first experiment we compared the accelerated EM algorithm described in Section 3 and Section 4 with the standard EM algorithm. The training set consisted of 10,000 points drawn from a 10-component 3-separated Gaussian mixture in two dimensions, and we also sampled a test set of 1000 points from the same mixture. We evaluate the algorithms based on the log-likelihood the learned mixture assigns to the test set in order to measure the ability to identify the generating mixture rather than the training data drawn from that mixture. We applied both algorithms to a mixture initialized with k-means.

We started the accelerated EM algorithm with an initial expansion of the tree to depth two. We kept this partition fixed and ran the algorithm until convergence. Convergence was measured in terms of relative increase in $\mathcal{F}$ (we used threshold 10^{-5}). We then refined the partition by expanding the node of the tree that led to maximal increase in $\mathcal{F}$. Then we ran the algorithm again until convergence, refined the partition by expanding best-first a single node of the tree, and so on. We terminated the algorithm if $\mathcal{F}$ hardly improved between two successive partitions. Note that refining a particular partition and computing the new responsibilities can be viewed as applying an E-step which justifies the use of the relative improvement of $\mathcal{F}$ as a convergence measure.

[6] Following Dasgupta (1999), a Gaussian mixture is c separated if for each pair (i, j) of component densities $\|m_i - m_j\| \geq c\sqrt{d \max\{\lambda_{\max}(C_i), \lambda_{\max}(C_j)\}}$, where $\lambda_{\max}(C)$ denotes the maximum eigenvalue of C.

In Fig. 1 we show the speedup and the (negative) log-likelihood obtained by the two algorithms vs. the true log-likelihood of the test set, averaged over 20 trials. Speed was measured using the total number of basic floating point operations needed for convergence.

With respect to run-time, the results show that: (1) the speedup of the accelerated EM compared to the standard EM is at least linear in the number of data points, as expected from the analysis, (2) the number of dimensions and components have a negative effect on the speedup, and (3) the amount of separation has a positive effect. The lower speedup in high dimensions can be ascribed to the use of a kd-tree, while many components or smaller separation lead to more diverse responsibilities, thus making coarse partitions perform worse. In general the accelerated EM requires more iterations to converge than the standard EM, but it is still faster than the latter since the iterations themselves are executed much faster.

With respect to solution quality, we note that both the standard EM and its accelerated counterpart reach solutions that are on the average suboptimal with respect to the true model. This is due to the k-means initialization. However, the relative difference in log-likelihood between the standard EM and our algorithm is small, even for mixtures with many components or high dimensionality.

7.2. Greedy EM vs. accelerated greedy EM

In a second experiment we compared the greedy EM algorithm described in Verbeek et al. (2003) with its ‘accelerated greedy’ counterpart of Section 5. We started the accelerated greedy EM algorithm with a partition of size four and expanded the tree one node at a time, best first, as in the first experiment. We used a default data set of 10,000 points and a test set of 500 points drawn from a 5-component 2-separated mixture in two dimensions. In Fig. 2 we show the results averaged over 20 trials. The accelerated greedy algorithm is always faster, with a speedup that is linear in the size of the data set. Moreover, thisspeedup comes almost ‘for free’: the log-likelihoods of both algorithms are practically equal to that of the generating mixture.

7.3. Incremental EM vs. accelerated EM

In this experiment we compared our accelerated EM algorithm with the incremental algorithm of Thiesson et al. (2001). The training set was sampled from a 10-component 3-separated Gaussian mixture density in two dimensions. Both algorithms were initialized using k-means. The accelerated algorithm was used as described above. After testing multiple block sizes, we found that using the incremental algorithm the largest speedup was obtained for a block size of 1/25 of the total data set size.

The goal of this experiment was to point out the main difference between the two algorithms in relation to speedup and performance. The measure used to point out the difference in speed is the comparison between the total time which the two algorithms need to convergence and the time the standard EM algorithm needs to converge. The measure to depict the quality of both algorithms is the log-likelihood of a test set under the learned mixtures. In Fig. 3, we show the obtained speedups and the negative log-likelihoods.

The difference in speedup is clear and can be explained by an obvious difference between the algorithms. The incremental algorithm randomly divides the data set in equally sized blocks, and performs partial EM-steps on each block iteratively. Each partial EM-step takes far less time than performing one full EM-step on the entire data set. The final speedup produced byThiesson is a result of the fact that the sum of the time needed to perform all partial EM-steps is smaller than the time needed to perform a single EM-step on the

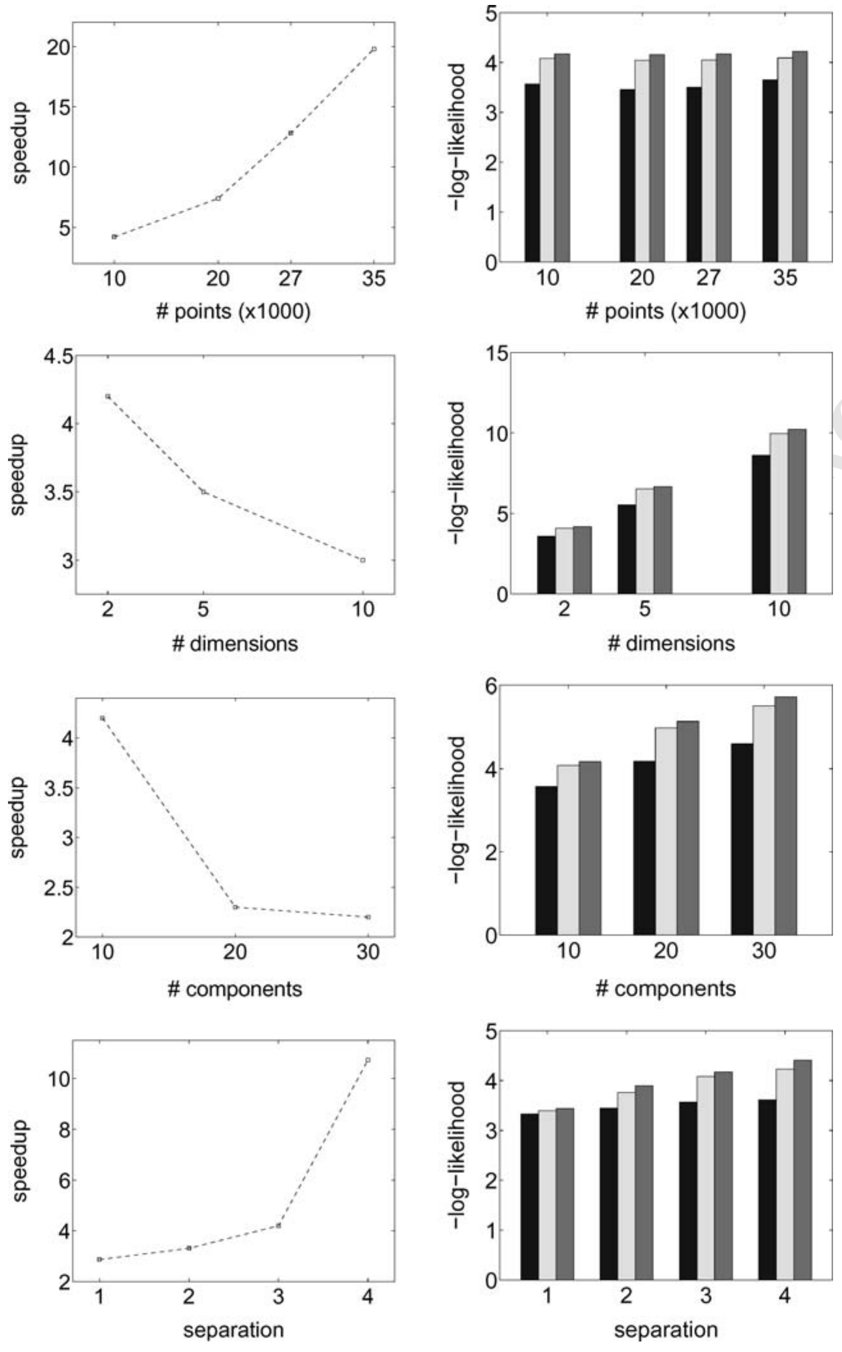

Fig. 1 Mixture learning with the standard EM vs. the accelerated EM. The graphs show the speedup factor and the bar charts show the negative log-likelihood: generating mixture (black), standard EM (light), accelerated EM (dark)

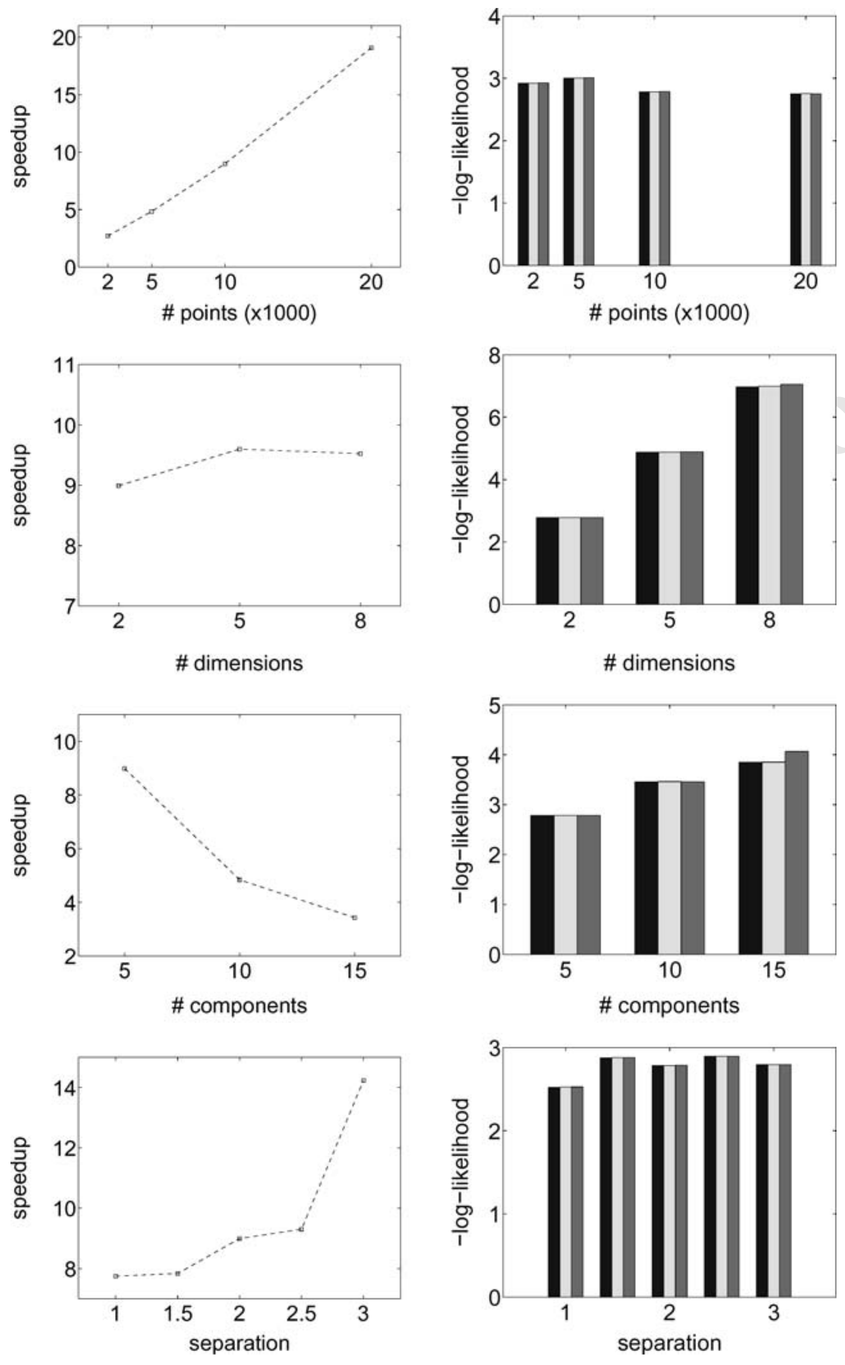

Fig. 2 Greedy mixture learning vs. the accelerated greedy EM. The graphs show the speedup factor and the bar charts show the negative log-likelihood: generating mixture (black), greedy EM (light), accelerated greedy EM (dark)

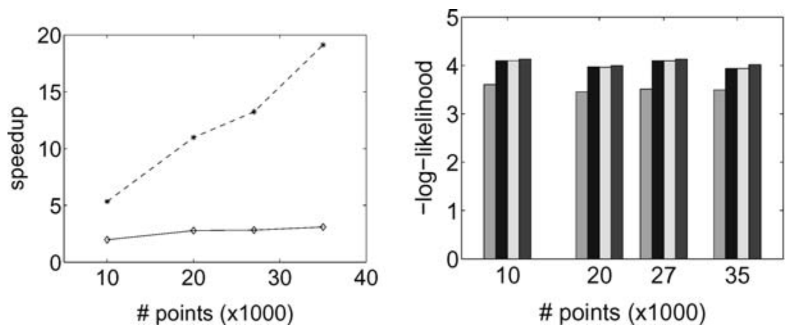

Fig. 3 Accelerated EM vs. incremental EM. The graph shows the speedup factor: accelerated EM (dotted line) and incremental EM (solid line). The bar chart shows the negative log-likelihood: generating mixture (mid grey), standard EM (black), incremental EM (light grey) and accelerated EM (dark grey)

entire data set. Optimal speedup is obtained by finding the optimal trade-off between doing partialEM-steps on small blocks and doing partial EM-steps on a small amount of blocks. But as the algorithm still performs these partial EM-steps using all the data-points in a single block, it eventually, after having processed each block, performs EM on all data points in the data set. Therefore, the time complexity is still linear with respect to the amount of data points, as it is with normal EM.

With respect to the solution quality, the performance of the incremental EM algorithm is comparable to that of the standard EM algorithm as it does take all data points into account just as the standard EM algorithm does. The relative difference in log-likelihood between the incremental EM and our algorithm is small.

7.4. Very fast EM vs. accelerated EM

In this experiment we compared the accelerated EM algorithm with the 'Very fast EM' algorithm proposed in Moore (1999). A training set of 100.000 data points was sampled from a 3-component 2-separated Gaussian mixture. Both algorithms were initialized using k-means. The initial binary trees were constructed by doing a full expansion until depth 6, thus resulting in a total tree-size of 127 nodes. The goal of this experiment was to point out the main difference between the two algorithms in relation to how they expanded the kd-tree.

With Moore's algorithm, expansion is done from the root node downwards. The criterion to expand a node is based on finding for each component s the minimum and maximum of $p(x|s)$ that can be attained within the bounding box of the node. The node is expanded if there is a component s for which the difference between the minimum and the maximum of $p(x|s)$ within the bounding box is bigger than a certain threshold τ, times the prior probability of that component. The higher τ is set, the smaller the expansion of the tree.

Finding the minimum and maximum of $p(x|s)$ in the bounding box can be formulated as finding the minimum and maximum of $(x - m_s)^\top C_s(x - m_s)$ such that x is within the bounding box of the node. The minimum and maximum can thus be found by solving a quadratic program. To make our comparison independent of the computationally costly operation of solving the quadratic program we did not compare the algorithms directly in terms of running time. Instead, we used the amount of expansion of the tree as a performance measure. The expansion is a good performance measure as it depicts the amount of nodes

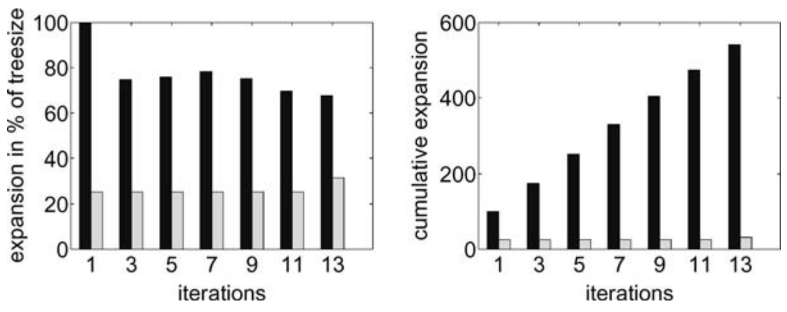

Fig. 4 Accelerated EM vs. very fast EM ($t = 0.5$). Left panel shows the percentage of tree-expansion, right panel shows the cumulative expansions of nodes as a percentage of the tree size. Results for very fast EM are in black, those for accelerated EM in grey

taken into account inevery EM step, so a cumulative plot of the amount of expanded leaf nodes is a proper representation of the quantity of operations performed before convergence.[7]

No significant difference in the log-likelihood assigned to the test was found between the mixtures learned using the two different algorithms. However, the amount of nodes that were expanded before convergence, and thus the required amount of computation, is considerably larger for Moore's algorithm. This is explained by the fact that Moore's algorithm expands in each EM step the tree from the root down until it finds nodes that fail to meet the expansion criterion. Therefore the cumulative amount of expansion operations is much larger as can be seen in Fig. 4. We conclude from these results that even without the additive computational cost of solving the quadratic programs, the amount of computation required by the very fast EM algorithm is much larger than the amount required by our algorithm.

8. Conclusions and future work

We presented an accelerated EM algorithm that can be used to speed up large data set applications of EM to learning mixtures of Gaussians. Our algorithm strictly maximizes in each learning step a lower bound on data log-likelihood, and that bound becomes tighter if we use finer partitions. The algorithm finds mixture configurations near local optima of the log-likelihood surface which are comparable with those found by the standard EM algorithm, but considerably faster.

Moreover, we have a convergent algorithm that maximizes a lower bound on data log-likelihood regardless of the particular partition of the data. This allows us to use rough partitions where the true posteriors differ heavily in each part, which might be needed when working on large data sets and only limited computational resources are available. In practice, we can start the algorithm with a rough partition for which the EM iterations are performed very fast, refine the partition when the algorithm converges, run EM again, and so on, thus balancing the computational cost with the quality of the solution.

Comparing our work with Moore (1999), we conclude that with less computational effort we can use the optimal shared responsibilities instead of the posterior at the node centroid. Furthermore, we obtained a provably convergent algorithm and have the freedom to use

[7] Recall that both algorithms have a running time that is linear in the number of nodes that is processed in each step.

arbitrary partitions of the data. Comparing with Verbeek et al. (2003), it appears that without compromising quality we can achieve speedups in greedy mixture learning that are at least linear in the number of data points.

In the experiments reported on in this paper we used kd-trees to find a hierarchical partitioning of the data set. For high-dimensional data, other tree structures (such as ball-trees (Omohundro, 1989) and anchor hierarchies (Moore, 2000)) have been reported to yield greater speedups of nearest neighbor queries and related tasks in high-dimensional data sets. More research is needed to determine the most efficient data structures to be used within our acceleration scheme. Advantages and disadvantages of overlapping vs. non-overlapping partition schemes need to be identified as well. Similarly, there might be more efficient strategies to refine the partitions in the course of the EM algorithm.

Finally, it would be interesting to see how the proposed framework performs on other mixture models like, for instance, the Generative Topographic Mapping (Bishop et al., 1998) or in supervised mixture modelling (Titsias and Likas, 2001). As future work we would like to consider the application of the proposed framework to the learning of non-Gaussian mixtures, e.g. mixtures for discrete data, using AD-trees and related techniques (Moore and Lee, 1998).

Acknowledgments We would like to thank the reviewers for their useful comments which helped to improve this manuscript. We are indebted to Tijn Schmits for part of the experimental work. JJV is supported by the Technology Foundation STW (project AIF 4997) applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

References

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Comm ACM 18(9):509–517

Bishop CM, Svensén M, Williams CKI (1998) GTM: The generative topographic mapping. Neur Comput 10:215–234

Bradley PS, Fayyad UM, Reina CA (1998) Scaling EM (expectation maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research

Dasgupta S (1999) Learning mixtures of Gaussians. In: Proceedings of the IEEE Symposium on Foundations of Computer Science, vol. 40. IEEE Computer Society Press, Los Alamitos, CA, USA, pp 634–644

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. J Royal Stat Soc Ser B (Methodological) 39(1):1–38

Gersho A, Gray RM (1992) Vector quantization and signal compression. Kluwer Academic Publishers, Boston

Kanungo T, Mount DM, Netanyahu N, Piatko C, Silverman R, Wu AY (2002) An efficient *k*-means clustering algorithm: Analysis and implementation. Trans Patt Anal Mach Intell 24:881–892

Li JQ, Barron AR (2000) Mixture density estimation. In: Solla SA, Leen TK, Müller K-R (eds) Advances in neural information processing systems, vol. 12. MIT Press, Cambridge, MA, USA, pp 279–285

Lindsay BG (1983) The geometry of mixture likelihoods: A general theory. Ann Stat 11(1):86–94

McCallum A, Nigam K, Ungar L (2000) Efficient clustering of high-dimensional data sets with application to reference matching. In: Ramakrishnan R, Stolfo S (eds) Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, vol. 6. ACM Press, New-York, NY, USA

McLachlan GJ, Peel D (2000) Finite mixture models. John Wiley & Sons

Moore A (1999) Very fast EM-based mixture model clustering using multiresolution kd-trees. In: Kearns MJ, Solla SA, Cohn DA (eds) Advances in Neural information processing systems, vol. 11. MIT Press, Cambridge, MA, USA, pp 543–549

Moore A, Pelleg D (1999) Accelerating exact *k*-means algorithms with geometric reasoning. In: Proc 5th Int Conf Knowledge Discovery and Data Mining, pp 277–281

Moore AW (2000) The anchors hierarchy: Using the triangle inequality to survive high-dimensional data. In: Boutilier C, Goldszmidt M (eds) Proceedings of the Annual conference on uncertainty in artificial intelligence, vol. 16. Morgan Kaufmann, San Mateo, CA, USA, pp 397–405

Moore AW, Lee MS (1998) Cached sufficient statistics for efficient machine learning with large data sets. J Arti Intell Res 8:67–91

Neal RM, Hinton GE (1998) A view of the EM algorithm that justifies incremental, sparse, and other variants. In: Jordan MI (eds) Learning in graphical models. Kluwer, Boston, MA, USA, pp 355–368

Nunnink JRJ (2003) Large scale Gaussian mixture modelling using a greedy expectation-maximisation algorithm. Master’s thesis, Informatics Institute, University of Amsterdam. www.science.uva.nl/research/ias/alumni/m.sc.theses

Omohundro SM (1989) Five balltree construction algorithms. Technical Report TR-89-063, International Computer Science Institute, Berkeley

Rose K (1998) Deterministic annealing for clustering, compression, classification, regression and related optimization proble ms. IEEE Trans Inform The 86(11):2210–2239

Sand P, Moore AW (2001) Repairing faulty mixture models using density estimation. In: Brodley CE, Danyluk AP (eds) Proceedings of the international conference on machine learning, vol. 18. Morgan Kaufmann, San Mateo, CA, USA, pp 457–464

Sproull RF (1991) Refinements to nearest-neighbor searching in *k*-dimensional trees. Algorithmica 6:579–589

Thiesson B, Meek C, Heckerman D (2001). Accelerating EM for large databases. Mach Learn 45(3):279–299

Titsias M, Likas A (2001) Shared kernel models for class conditional density estimation. IEEE Trans Neur Netw 12(5):987–997

Verbeek JJ, Vlassis N, Kröse BJA (2003) Efficient greedy learning of Gaussian mixture models. Neur Comput 15(2):469–485

Vlassis N, Likas A (2002) A greedy EM algorithm for Gaussian mixture learning. Neur Proc Lett 15(1):77–87

Zhang T (2002) A general greedy approximation algorithm with applications. In: Dietterich TG, Becker S, Ghahramani Z (eds) Advances in neural information processing systems, vol. 14. MIT Press, Cambridge, MA, USA