Gossip-based greedy Gaussian mixture learning

Nikos Vlassis[1] Yiannis Sfakianakis[1] Wojtek Kowalczyk[2]

[1]Informatics Institute, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{vlassis,jsfakian}@science.uva.nl

[2]Department of Computer Science, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
wojtek@cs.vu.nl

Abstract. It has been recently demonstrated that the classical EM algorithm for learning Gaussian mixture models can be successfully implemented in a decentralized manner by resorting to gossip-based randomized distributed protocols. In this paper we describe a gossip-based implementation of an alternative algorithm for learning Gaussian mixtures in which components are added to the mixture one after another. Our new Greedy Gossip-based Gaussian mixture learning algorithm uses gossip-based parallel search, starting from multiple initial guesses, for finding good components to add to the mixture in each component allocation step. It can be executed on massive networks of small computing devices, converging to a solution exponentially faster than its centralized version, while reaching the same quality of generated models.
Keywords: Data mining, Algorithms and Complexity, Computer and Sensor Networks, Information Retrieval.

1 Introduction

Gaussian mixture models constitute a rich family of probability distributions, with many applications in statistics, pattern recognition, machine learning, and data mining [1]. Such models postulate that the observed data are generated by a two-level process that first samples components, and then draws data from the corresponding Gaussian distributions. Gaussian mixture models have been used, e.g., for clustering large datasets [2], for dimension reduction [3], and for classification [4].

Learning the parameters of a Gaussian mixture from a given dataset is often carried out by maximum likelihood and the EM algorithm [5]. The EM algorithm is an iterative optimization technique that starts with an initial estimate of the mixture parameters, and in each step produces a new parameter estimate that increases the likelihood function. However, EM is a local optimization algorithm and therefore is likely to get trapped in a local maximum of the likelihood function. Several initialization methods have been proposed for tackling this problem [6].

A way to resolve the sensitivity of EM to initialization and to improve its convergence performance is to use a greedy learning approach [7–9]. In the greedy approach, components are added to the mixture one after the other until a desired number of components. The main idea is to replace the original k-component mixture problem by a sequence of 2-component mixture problems that are easier to solve. As it was shown in [8, 9], the greedy approach can produce much better results than the standard EM algorithm (with random restarts) with little extra overhead. A similar approach has been proposed in [10] in which components of the mixture are split and merge in order to avoid local maxima of EM.

Recently, a decentralized implementation of the EM algorithm for Gaussian mixture learning, called Newscast EM, was proposed for data that are distributed over a the nodes of a network [11]. This method relies on a gossip-based randomized protocol that implements the M-step of the EM algorithm in a parallel-distributed fashion: each node starts with a local estimate of the mixture parameters, and then pairs of nodes repeatedly exchange their parameter estimates and combine them by weighted averaging. In such a gossip-based M-step, nodes learn the correct estimates exponentially fast, in a number of cycles that is logarithmic in the network size.

In this paper we show how similar gossip-based protocols can be used for the greedy learning of Gaussian mixture models. In particular, we derive a gossip-based distributed implementation of the greedy learning algorithm of [9]. The derived algorithm essentially resolves the sensitivity to initialization of the algorithm of [11]. Preliminary results indicate that the proposed algorithm can achieve comparable results to its centralized counterpart, but much faster.

2 Gaussian mixtures and the EM algorithm

For random vector $x \in \mathbb{R}^d$, a k-component Gaussian mixture model is given by the convex combination

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

of k Gaussian densities

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

parameterized by their means m_s and covariance matrices C_s, while the mixing weights π_s satisfy $\sum_s \pi_s = 1$, and define a 'prior' distribution over the components. For a given dataset $\{x_1, \ldots, x_n\}$ of independent and identically distributed samples from $p(x)$, the learning problem is to estimate the parameter vector $\theta = \{\pi_s, m_s, C_s\}_{s=1}^k$ of the k components that maximizes the log-likelihood function (assuming that the latter is bounded from above)

$$\mathcal{L} = \sum_{i=1}^{n} \log p(x_i) = \sum_{i=1}^{n} \log \sum_{s=1}^{k} \pi_s p(x_i|s). \tag{3}$$

Maximization of the data log-likelihood $\mathcal{L}$ can be carried out by the EM algorithm, which is an iterative optimization algorithm that maximizes in each step a lower bound of $\mathcal{L}$ [5, 12]. This bound $\mathcal{F}$ is a function of the current mixture parameters θ and a set of n 'responsibility' distributions $q_i(s)$, one for each point x_i. This lower bound, analogous to the variational free energy in statistical physics, equals

$$\mathcal{F} = \sum_{i=1}^{n} \sum_{s=1}^{k} q_i(s) \big[\log \pi_s + \log p(x_i|s) - \log q_i(s) \big]. \tag{4}$$

The standard EM algorithm starts with an initial estimate of the parameter vector θ (e.g., random or computed by a clustering method like k-means), and then it alternates between two steps. In the E-step, the energy $\mathcal{F}$ is maximized over the responsibilities q_i giving $q_i(s) = p(s|x_i)$, i.e., the Bayes posteriors given the parameters found in the previous step. In the M-step, the energy $\mathcal{F}$ is maximized over the parameters θ keeping the $q_i(s)$ fixed, giving the following equations:

$$\pi_s = \frac{1}{n} \sum_{i=1}^{n} q_i(s), \tag{5}$$

$$m_s = \frac{1}{n\pi_s} \sum_{i=1}^{n} q_i(s) x_i, \tag{6}$$

$$C_s = \frac{1}{n\pi_s} \sum_{i=1}^{n} q_i(s) x_i x_i^\top - m_s m_s^\top. \tag{7}$$

The E- and M-steps are repeated until $\mathcal{L}$ does not improve significantly between two consecutive iterations.

3 Greedy Gaussian mixture learning

One the limitations of the standard EM algorithm is that it is very sensitive to the initialization of the parameter vector θ. For various initialization choices EM can easily get trapped in local maxima of the log-likelihood function. An alternative approach which avoids the initialization of θ is to start with a single-component mixture (which is trivial to find) and then keep on adding components to the mixture one after the other [7–9]. In particular, each $(k+1)$-component mixture $p_{k+1}(x)$ is recursively defined as the convex combination of a k-component mixture $p_k(x)$ and a new component $f(x; \theta_{k+1})$, i.e.,

$$p_{k+1}(x) = (1 - a_{k+1}) p_k(x) + a_{k+1} f(x; \theta_{k+1}), \tag{8}$$

with mixing weight $a_{k+1} \in (0, 1)$. Assuming that $p_k(x)$ has already been learned, learning the parameters θ_{k+1} of the new component and the mixing weight a_{k+1} can be done by maximizing the log-likelihood of $p_{k+1}(x)$

$$\mathcal{L}_{k+1} = \sum_{i=1}^{n} \log[(1 - a_{k+1}) p_k(x_i) + a_{k+1} f(x_i; \theta_{k+1})] \tag{9}$$

with $p_k(x)$ kept fixed. As shown in [7], if optimal parameters $[a^*_{k+1}, \theta^*_{k+1}] = \arg\max_{[a_{k+1},\theta_{k+1}]} \mathcal{L}_{k+1}$ are computed for every k, then the 'greedy' k-component mixture $p_k(x) = (1 - a^*_k)p_{k-1}(x) + a^*_k f(x; \theta^*_k)$, can be almost as good as the maximum likelihood mixture $p^*(x)$ in the following sense:

$$\sum_{i=1}^{n} \log p_k(x_i) \geq \sum_{i=1}^{n} \log p^*(x_i) - \frac{c}{k}, \tag{10}$$

where k is the number of components of $p_k(x)$, and c is a constant independent of k.

Although the theoretical results of [7] justify the greedy approach for mixture learning, in practice it is difficult to compute the optimal parameters $[a^*_{k+1}, \theta^*_{k+1}]$ that maximize $\mathcal{L}_{k+1}$ in each component allocation step. Since $\mathcal{L}_{k+1}$ cannot be analytically maximized, an option is to perform a global search over the space of $[a_{k+1}, \theta_{k+1}]$ starting from some initial (random) estimates. This is the approach taken in [9]: a number of candidate components are generated from each one of the k components of $p_k(x)$ by randomization, and then a 'partial' EM algorithm is executed that searches locally for a vector $[\tilde{a}_{k+1}, \tilde{\theta}_{k+1}]$ with high $\mathcal{L}_{k+1}$, and which hopefully is not too far from $[a^*_{k+1}, \theta^*_{k+1}]$. In particular, assuming a fixed k-component mixture $p_k(x)$, component allocation is done as follows:

1. Data are first assigned to their 'nearest' component according to posterior probability. That is, each point x_i is assigned to component $\arg\max_s p(s|x_i)$.
2. For each component s, let A_s be the set of points assigned to component s. Repeat until m candidates (e.g., $m = 10$) are created from s:
   (a) Select two points x_s, x'_s uniformly at random from A_s.
   (b) Cluster all A_s points in two subsets, according to their nearest Euclidean distance from x_s and x'_s.
   (c) Create a candidate component f_s from each subset, having $\theta_{k+1} = [m_{k+1}, C_{k+1}]$ the mean and the covariance of the points in the subset.
   (d) Set $a_{k+1} = 0.5$ and update $[a_{k+1}, \theta_{k+1}]$ with partial EM steps, as explained below.
3. Among all mk updated parameter vectors, select the vector $[\tilde{a}_{k+1}, \tilde{\theta}_{k+1}]$ with the highest $\mathcal{L}_{k+1}$.

In the 2d step above, the parameters $[a_{k+1}, \theta_{k+1}]$ are updated by an EM algorithm that optimizes a lower bound of $\mathcal{L}_{k+1}$. The idea is to treat $p_{k+1}(x)$ as a two-component mixture, composed of the new component $f(x; \theta_{k+1})$ and the fixed mixture $p_k(x)$, and lower bound $\mathcal{L}_{k+1}$ by setting to zero the responsibility $q_i(f_s)$ of candidate components f_s for points not in A_s. This has the effect that the partial EM steps can be carried out fast, with total cost $O(mn)$. Maximizing this bound over the responsibilities $q_i(f_s)$ gives

$$q_i(f_s) = \frac{a_{k+1} f(x_i; \theta_{k+1})}{(1 - a_{k+1})p_k(x_i) + a_{k+1} f(x_i; \theta_{k+1})}, \tag{11}$$

which is the Bayes posterior for the two-component mixture $p_{k+1}(x)$. Similarly, maximizing the bound over the parameters a_{k+1} and $\theta_{k+1} = [m_{k+1}, C_{k+1}]$ gives:

$$a_{k+1} = \frac{1}{n} \sum_{i \in A_s} q_i(f_s), \tag{12}$$

$$m_{k+1} = \frac{1}{n a_{k+1}} \sum_{i \in A_s} q_i(f_s) x_i, \tag{13}$$

$$C_{k+1} = \frac{1}{n a_{k+1}} \sum_{i \in A_s} q_i(f_s) x_i x_i^\top - m_{k+1} m_{k+1}^\top. \tag{14}$$

When the best vector $[\tilde{a}_{k+1}, \tilde{\theta}_{k+1}]$ has been found in step 3 above, the new $(k+1)$-component mixture is formed as

$$p_{k+1}(x) = (1 - \tilde{a}_{k+1}) p_k(x) + \tilde{a}_{k+1} f(x; \tilde{\theta}_{k+1}), \tag{15}$$

and is subsequently updated with standard EM (all its $k+1$ components are updated). Upon convergence of EM, a new component is added to the mixture, and so on until some criterion on the number of components is satisfied, e.g., one based on MDL [7]. The above greedy algorithm is experimentally shown to outperform the standard EM with random restarts [9].

4 Gossip-based Gaussian mixture learning

A recent development in Gaussian mixture modeling is the use of gossip-based protocols for distributed learning [13, 14, 11, 15]. Such protocols apply in the case where the data x_i are not centrally available but are distributed over the nodes of a network. The main idea is to decompose the M-step of the EM algorithm into a number of cycles: each node maintains a local estimate of the model parameters, and in every cycle it contacts some other node at random, and the two nodes update their model estimates by weighted averaging. As shown in [11], under such a protocol the local estimates of the individual nodes converge exponentially fast to the correct solution in each M-step of the algorithm. In some applications the gossip approach may be more advantageous then other distributed implementations of EM that resort on global broadcasting [16] or routing trees [17].

The above distributed EM implementation relies on a gossip-based protocol that computes the average μ of a set of numbers $v_1, \ldots, v_n$ that are stored in the nodes of a network (one value per node). Each node i initially sets $\mu_i = v_i$ as its local estimate of μ, and then it runs the following protocol for a number of cycles:

1. Contact a node j that is chosen uniformly at random from $1, \ldots, n$.
2. Nodes i and j update their estimates by $\mu_i' = \mu_j' = (\mu_i + \mu_j)/2$.

It turns out that under this protocol each node learns the correct average very fast, in a number of cycles that is logarithmic in the sample size:

Theorem 1 (Kowalczyk and Vlassis, 2005). *With probability at least* $1-\delta$, *after* $\lceil 0.581(\log n + 2\log\sigma + 2\log\frac{1}{\varepsilon} + \log\frac{1}{\delta})\rceil$ *cycles holds* $\max_i |\mu_i - \mu| \leq \varepsilon$, *for any* $\varepsilon > 0$ *and data variance* σ^2.

Using this gossip-based protocol, a distributed implementation of the standard EM algorithm is possible by noting that the M-step (5)–(7) involves the computation of a number of averages. The idea is that each node i maintains a local estimate $\theta_i = \{\pi_{is}, m_{is}, \tilde{C}_{is}\}$ of the parameters of the mixture, where $C_{is} = \tilde{C}_{is} - m_{is}m_{is}^\top$, and the following protocol is executed identically and in parallel for each node i:

1. **Initialization.** Set $q_i(s)$ to some random number in $(0,1)$ and then normalize all $q_i(s)$ to sum to 1 over all s.
2. **M-step.** Initialize i's local estimates for each component s by $\pi_{is} = q_i(s)$, $m_{is} = x_i$, $\tilde{C}_{is} = x_i x_i^\top$. Then, for a fixed number of cycles, contact node j and update both i and j local estimates for each component s by:

$$\pi'_{is} = \pi'_{js} = \frac{\pi_{is} + \pi_{js}}{2}, \tag{16}$$

$$m'_{is} = m'_{js} = \frac{\pi_{is}m_{is} + \pi_{js}m_{js}}{\pi_{is} + \pi_{js}}, \tag{17}$$

$$\tilde{C}'_{is} = \tilde{C}'_{js} = \frac{\pi_{is}\tilde{C}_{is} + \pi_{js}\tilde{C}_{js}}{\pi_{is} + \pi_{js}}. \tag{18}$$

3. **E-step.** Compute $q_i(s) = p(s|x_i)$ for each s, using the M-step estimates π_{is}, m_{is}, and $C_{is} = \tilde{C}_{is} - m_{is}m_{is}^\top$.
4. **Loop.** Go to 2, unless a stopping criterion is satisfied.

As reported in [11], the above gossip-based learning algorithm produces results that are essentially identical to those obtained by the standard EM algorithm, but much faster. Resorting to Theorem 1, one can see that each node can implement the M-step in $O(\log n)$ time, whereas if the data were to be transferred to and processed by a central server, the runtime would have been $O(n)$. The communication complexity of each gossip M-step (total number of messages sent over the network) is $O(n \log n)$ since each node contacts $O(\log n)$ other nodes (one per gossip cycle).

5 Gossip-based greedy Gaussian mixture learning

A disadvantage of the gossip-based EM algorithm above is that the initialization (step 1) is random. In this section we derive a gossip-based greedy Gaussian mixture learning algorithm in which components are added sequentially to the mixture. This requires implementing a (randomized) function that computes candidate components to add to a mixture. We first note that if this function depends only on the parameters of the mixture, and since each node at the end of each M-step has converged to the same mixture parameters, then clearly each node can compute the same set of candidate components by using the same random number generator and same seed. On the other hand, if this function depends also on the data as in [8, 9], then it can also be implemented by gossip-based protocols as we show below. The resulting algorithm will alternate between full mixture updating using the gossip-based EM algorithm described in the previous section, and component allocation using the gossip-based approach described next.

We show here how each one of the component allocation steps 1–3 of Section 3 can be implemented in a gossip-based manner. We assume that a k-component mixture has already been learned by the gossip-based EM algorithm above, and therefore all nodes know the same set of parameters π_s, m_s, and C_s, for $s = 1, \dots, k$, that fully characterize $p_k(x)$. Hence each node i can evaluate for instance $p_k(x_i)$ directly, but not $p_k(x_j)$ for $j \neq i$.

1. First all data points should be partitioned to sets A_s according to their 'nearest' component in terms of posterior. For each node i we can directly evaluate $\arg\max_s p(s|x_i)$, so each i knows in which set A_s it belongs, but it does not know the assignment of a node $j \neq i$.
2. Each node $i \in A_s$ creates a local cache which initially contains only i, and which eventually will contain all nodes $j \in A_s$ (a subnetwork). To achieve this, each node i contacts random nodes from $1, \dots, n$ until it finds a node $j \in A_s$ (note that node j knows whether it is in A_s). The probability of locating such a node j within ρ steps by sampling uniformly at random from all n nodes is approximately $1 - (1 - \pi_s)^\rho$, from which we can bound the number of steps needed so that all nodes know with high probability at least one more node in their partition sets. When all nodes have two entries in their local caches, each node i runs a protocol in which it repeatedly (for a fixed number of steps) contacts a node j uniformly at random from i's local cache and merges j's local cache with its cache. The result is a local cache for each node $i \in A_s$ that contains (almost) all nodes $j \in A_s$.

   (a) After a subnetwork has been formed for every component s, and each node $i \in A_s$ knows almost all other nodes $j \in A_s$, selecting two random points x_s, x'_s from A_s corresponds to selecting two random nodes from each subnetwork. For this, each node $i \in A_s$ chooses two random numbers in $(0, 1)$, and it repeatedly runs the following protocol (the number of steps can be easily bounded):
   - Select a node j from i's cache.
   - Both nodes i and j compute and maintain the max and the min of their numbers, together with the corresponding maximizing/minimizing x-points (which are also propagated by the same protocol).

   At the end of this protocol (which has super-exponential convergence), all nodes in A_s know the (same) two points $x_s \in A_s$ and $x'_s \in A_s$.

   (b) Each node $i \in A_s$ computes its Euclidean distance to both x_s and x'_s, so it knows in which corresponding subset it belongs, and it sets a bit $b_i \in \{0, 1\}$ accordingly.

(c) A candidate vector $[a_f, \theta_f]$, with $\theta_f = [m_f, C_f]$, is implicitly created by setting $q_i(f) = b_i$ (or $q_i(f) = \bar{b}_i$ for the other subset), for each node i in the initialization of the partial EM (see next step).

(d) Compute a good $[a_f, \theta_f]$ with gossip-based partial EM steps. Each node i maintains a local estimate $[a_{if}, \theta_{if}]$ of $[a_f, \theta_f]$ and executes the following protocol:

i. **Initialization.** Set $q_i(f) = b_i$.

ii. **M-step.** Initialize i's local estimates by $a_{if} = q_i(f)$, $m_{if} = x_i$, $\tilde{C}_{if} = x_i x_i^\top$. Then, for a fixed number of cycles, contact node j from i's cache, and update both i and j local estimates by:

$$a'_{if} = a'_{jf} = \frac{a_{if} + a_{jf}}{2}, \tag{19}$$

$$m'_{if} = m'_{jf} = \frac{a_{if} m_{if} + a_{jf} m_{jf}}{a_{if} + a_{jf}}, \tag{20}$$

$$\tilde{C}'_{if} = \tilde{C}'_{jf} = \frac{a_{if} \tilde{C}_{if} + a_{jf} \tilde{C}_{jf}}{a_{if} + a_{jf}}. \tag{21}$$

iii. **E-step.** Using the M-step estimates a_{if}, m_{if}, and $C_{if} = \tilde{C}_{if} - m_{if} m_{if}^\top$, compute the posterior

$$q_i(s) = \frac{a_{if} f(x_i; \theta_{if})}{(1 - a_{if}) p_k(x_i) + a_{if} f(x_i; \theta_{if})}. \tag{22}$$

Note that node i can compute the quantity $p_k(x_i)$, as we explained above.

iv. **Loop.** Go to step ii, unless a stopping criterion is satisfied.

3. Note that, contrary to the centralized greedy algorithm of Section 3, here all mk candidate components can be updated in parallel. That is, steps (a)–(d) above can be replicated m times using the same gossip-based protocols. After steps 1 and 2 above, each node $i \in A_s$ knows all m candidate candidates that have been generated and EM-updated from points in A_s. The remaining task is to have all nodes agree which candidate to add to $p_k(x)$. Evaluating $\mathcal{L}_{k+1}$ from (9) for all of them would be expensive, as it would require that all nodes exchange all mk candidates. For practical purposes, one can consider an approximation of $\mathcal{L}_{k+1} = \sum_{i=1}^n \log[(1 - a_f) p_k(x_i) + a_f f(x_i; \theta_f)]$ in which the contribution of a new component f_s to $\mathcal{L}_{k+1}$ is zero for data points outside A_s. This gives:

$$\begin{aligned}\tilde{\mathcal{L}}_{k+1} = \sum_{i \in A_s} &\log[(1 - a_f) p_k(x_i) + a_f f(x_i; \theta_f)] \\ &+ \sum_{i \notin A_s} \log[(1 - a_f) p_k(x_i)] \\ &= \mathcal{L}_k + n \log(1 - a_f) + \sum_{i \in A_s} \log\Big(\frac{a_f f(x_i; \theta_f)}{(1 - a_f) p_k(x_i)} + 1\Big). \end{aligned} \tag{23}$$

Note that the term $\mathcal{L}_k$ is independent of $[a_f, \theta_f]$, while the third term is a sum that involves only points from each local subset A_s. Hence the latter can be efficiently computed with the gossip-based averaging protocol of Section 4 where each node i contacts only nodes j from its local cache. The quantity $\tilde{\mathcal{L}}_{k+1}$ is evaluated by each node $i \in A_s$ for all m candidates f_s, and the best candidate is kept. Then all nodes run the max-propagation protocol of step 2(a) above, in order to compute the component with the highest $\tilde{\mathcal{L}}_{k+1}$ among all mk components. Note that the complexity of the above operations is $O(\log n)$ as implied by Theorem 1, as compared to the $O(n)$ complexity of the centralized greedy EM of Section 3.

6 Results

In Fig. 1(left) we demonstrate the performance of the gossip-based averaging protocol as described in [11], for typical averaging tasks involving zero-mean unit-variance data. We plot the variance reduction rate (mean and one standard deviation for 50 runs) as a function of the number of cycles, for $n = 10^5$.

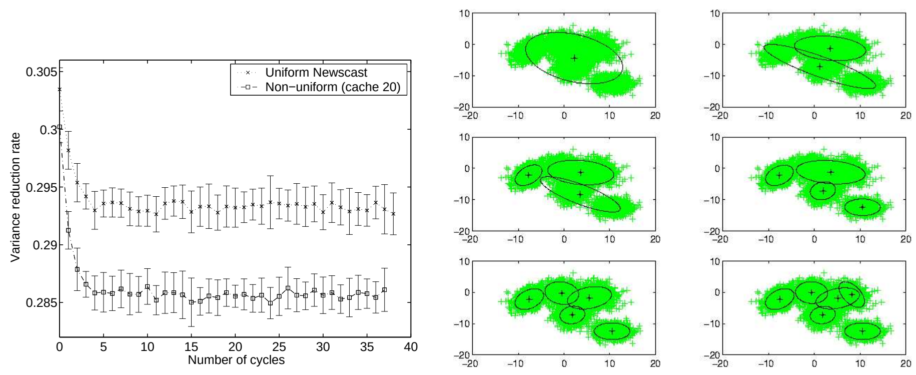

Fig. 1. (Left) Variance reduction rate of gossip-based averaging for $n = 10^5$ data. (Right) A typical run of the proposed greedy gossip-based algorithm for a 6-component mixture with 10^4 points.

We also ran a preliminary set of experiments comparing the performance of the proposed greedy gossip-based mixture learning algorithm with the algorithm of [9]. We used synthetic datasets consisting of 10^4 points drawn from Gaussian mixtures in which we varied the number of components (5, 10), the dimensionality (1, 2, 5), and the degree of separation c of the components (from 1 to 8; separation c means that for each i and j holds $||m_i - m_j|| \geq c\sqrt{\max\{\text{trace}(C_i), \text{trace}(C_j)\}}$ [18]). The results are summarized in Table 1 where we see that the two methods are virtually identical in terms of log-likelihood of a test set. The gossip-based algorithm however is much faster than the centralized one, as we explained above. A typical run for a mixture of 6 components in shown in Fig. 1(right).

Components	Dimension	Separation	Proposed algorithm	Greedy EM
5	1	3	-3.3280	-3.3280
5	1	8	-3.4294	-3.4294
10	1	3	-3.9905	-3.9905
10	1	8	-3.9374	-3.9374
5	2	1	-5.0590	-5.0494
5	2	4	-5.3331	-5.3322
10	2	1	-5.7179	-5.7132
10	2	4	-5.8104	-5.8080
5	5	1	-10.1581	-10.1367
5	5	4	-10.4246	-10.3940
10	5	1	-10.8107	-10.7939
10	5	4	-11.2110	-11.1913

Table 1. Proposed algorithm vs. centralized greedy mixture learning, for various mixture configurations.

7 Conclusions

We proposed a decentralized implementation of greedy Gaussian mixture learning using gossip-based protocols. The proposed algorithm applies in cases where a (large) set of data are distributed over the nodes of a network, and where point-to-point communication between two nodes is available. Compared to the gossip-based EM algorithm of [11], the current algorithm does not require (random) initialization of the mixture, but it grows the mixture sequentially by adding components one after the other. Compared to the algorithm of [9] the algorithm is exponentially faster for data that are already distributed over a network, without compromising solution quality.

Acknowledgments

We would like to thank A. Likas, J.J. Verbeek, and S. Voulgaris for useful discussions. The first author is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414.

References

1. McLachlan, G.J., Peel, D.: Finite Mixture Models. John Wiley & Sons (2000)

2. Moore, A.W.: Very fast EM-based mixture model clustering using multiresolution kd-trees. In: Advances in Neural Information Processing Systems 11, The MIT Press (1999)
3. Tipping, M.E., Bishop, C.M.: Mixtures of probabilistic principal component analysers. Neural Computation **11**(2) (1999) 443–482
4. Titsias, M., Likas, A.: Shared kernel models for class conditional density estimation. IEEE Trans. Neural Networks **12**(5) (2001) 987–997
5. Dempster, A.P., Laird, N.M., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Statist. Soc. B **39** (1977) 1–38
6. Meila, M., Heckerman, D.: An experimental comparison of several clustering and initialization methods. In: Proc. 14th Ann. Conf. on Uncertainty in Artificial Intelligence, San Francisco, CA (1998) 386–395
7. Li, J.Q., Barron, A.R.: Mixture density estimation. In: Advances in Neural Information Processing Systems 12, The MIT Press (2000)
8. Vlassis, N., Likas, A.: A greedy EM algorithm for Gaussian mixture learning. Neural Processing Letters **15**(1) (2002) 77–87
9. Verbeek, J.J., Vlassis, N., Kröse, B.: Efficient greedy learning of Gaussian mixture models. Neural Computation **15**(2) (2003) 469–485
10. Ueda, N., Nakano, R., Ghahramani, Z., Hinton, G.E.: SMEM algorithm for mixture models. Neural Computation **12** (2000) 2109–2128
11. Kowalczyk, W., Vlassis, N.: Newscast EM. In Saul, L.K., Weiss, Y., Bottou, L., eds.: Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA (2005)
12. Neal, R.M., Hinton, G.E.: A view of the EM algorithm that justifies incremental, sparse, and other variants. In Jordan, M.I., ed.: Learning in graphical models. Kluwer Academic Publishers (1998) 355–368
13. Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized rumour spreading. In: Proc. 41th IEEE Symp. on Foundations of Computer Science, Redondo Beach, CA (2000)
14. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. 44th IEEE Symp. on Foundations of Computer Science, Cambridge, MA (2003)
15. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Gossip algorithms: Design, analysis and applications. In: Proc. IEEE Infocom, Miami, FL (2005)
16. Forman, G., Zhang, B.: Distributed data clustering can be efficient and exact. ACM SIGKDD Explorations **2**(2) (2000) 34–38
17. Nowak, R.D.: Distributed EM algorithms for density estimation and clustering in sensor networks. IEEE Trans. on Signal Processing **51**(8) (2003) 2245–2253
18. Dasgupta, S.: Learning mixtures of Gaussians. In: Proc. IEEE Symp. on Foundations of Computer Science, New York (1999)