Efficient greedy learning of Gaussian mixtures

Jakob Verbeek, Nikos Vlassis, Ben Krose

Efficient Greedy Learning
of Gaussian Mixtures

J.J. Verbeek, N. Vlassis and B. Kröse
Computer Science Institute
Faculty of Science
University of Amsterdam
The Netherlands

We present a deterministic greedy method to learn a mixture of Gaussians which runs in $O(nk^2)$ time. The key element is that we build the mixture component-wise. By allocating a new component close to optimal in the existing (close to optimal) learned mixture, we hope to be able to reach a solution close to optimal for the new mixture. Each component of the mixture is characterized by a fixed number of parameters. Then, instead of solving directly a optimization problem involving the parameters of all components, we replace the problem by a sequence of optimization problems involving only the parameters of the new component. We include experimental results obtained on image segmentation and reconstruction tasks as well as results of extensive tests on artificially generated data sets. In these experiments the learning method compares favorably to the standard EM with random initialization as well as to another existing greedy approach to learning Gaussian mixtures.
Keywords: Unsupervised learning, Finite mixture models, Gaussian mixtures, EM algorithm

*Intelligent
Autonomous
Systems*

Contents

Intelligent Autonomous Systems
Computer Science Institute
Faculty of Science
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
The Netherlands

tel: +31 20 525 7461
fax: +31 20 525 7490
http://www.science.uva.nl/research/ias/

Corresponding author:

J.J. Verbeek
tel: +31 (20) 525 7515
jverbeek@science.uva.nl
http://carol.science.uva.nl/~jverbeek/

1   Introduction

This paper concerns the learning (fitting the parameters of) a mixture of Gaussian distributions [13]. Mixture models form an expressive class of models for density estimation. Applications in a wide range of fields have emerged in the past decades. They are used for density estimation in 'unsupervised' problems, for clustering purposes, for estimating class-conditional densities in supervised learning settings, in situations where the data is partially supervised and in situations where some observations have 'missing values'. A recent development in mixture modeling with Gaussians is their use for local linear dimension reduction. This application was given solid foundation in the Mixtures of Probabilistic Principal Component Analyzers (MPPCA) model [18].

The most widely used algorithm to learn mixture models is the Expectation-Maximization (EM) algorithm [6]. For a given finite data set $\mathbf{X}_n$ of n observations and an initial mixture f_0, the algorithm provides a means to generate a sequence of mixture models $\{f_i\}$ with increasing log-likelihood on $\mathbf{X}_n$. The EM algorithm is known to converge to a locally optimal solution. However, convergence to a globally optimal solution is not guaranteed. The log-likelihood of the given data set under the found mixture distribution is highly dependent on the initial mixture f_0.

The standard method to overcome the high dependence on initialization is to start the EM algorithm for several random initializations and use the mixture yielding maximum likelihood on the data. The result is a nondeterministic algorithm. The nondeterminism of the algorithm may be unfavorable, especially in the process of developing large systems that include the learning of a mixture model as a module. In order to effectively evaluate performance of other modules it is convenient to rule out accidental aberrant behavior of the complete system due to 'unlucky' initializations of the mixture learning module.

In this paper we present a deterministic method to learn mixtures of k Gaussians. The method is based on previous work [22]. Here, we solve the main drawback of [22]: the time complexity is reduced from $O(k^2n^2)$ to $O(k^2n)$. Furthermore, the new method extends naturally to learning MPPCA models. The novell approach peforms on average significantly better than the approach in [22]. Both algorithms use roughly the same approach: the mixture is build-up by starting with a one-component mixture and adding new components one after the other. The new algorithm replaces the 'static' component insertion step of [21] with a procedure that inserts a component that is selected among a set of 'candidate' components dependent on the existing mixture.

The paper is organized as follows: In Section 2 we recapitulate the definition and EM-learning of Gaussian mixtures. Section 3 forms the core of the paper and describes our new greedy approach to Gaussian mixture learning. Then, in Section 4, we present experimental results on three tasks: modeling of artificially generated data drawn from several types of Gaussian mixtures, image reconstruction and texture segmentation. The second task involves the MPPCA model. The experiments compare the results of the new algorithm with those of the standard EM and the approach of [22]. Section 5 ends the paper with conclusions and a discussion.

2   Gaussian Mixtures and the EM Algorithm

A Gaussian mixture is defined as a convex combination of Gaussian densities. A Gaussian density in a d-dimensional space, parameterized by its mean $\mathbf{m} \in \mathbb{R}^d$ and $d \times d$ covariance matrix $\mathbf{C}$ is defined as:

$$\phi(\mathbf{x}; \theta) = (2\pi)^{-d/2} \det(\mathbf{C})^{-1/2} \exp\left(-(\mathbf{x} - \mathbf{m})^\top \mathbf{C}^{-1}(\mathbf{x} - \mathbf{m})/2\right), \qquad (1)$$

where θ denotes the parameters $\mathbf{m}$ and $\mathbf{C}$. A mixture of k Gaussians is then defined as:

$$f_k(\mathbf{x}) = \sum_{i=1}^{k} \pi_i \phi(\mathbf{x}; \theta_i), \quad \text{with} \quad \sum_{j=1}^{k} \pi_j = 1 \quad \text{and for} \quad j \in \{1, \ldots, k\} : \pi_j \geq 0. \tag{2}$$

The π_i are called the mixing weights and $\phi(\mathbf{x}; \theta_i)$ the components of the mixture.

The well known EM algorithm [6] enables us to update the parameters of a given k-component mixture with respect to a data set $\mathbf{X}_n = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ with all $\mathbf{x}_i \in \mathbb{R}^d$, such that the likelihood of $\mathbf{X}_n$ is never smaller under the new mixture. The updates for a mixture of Gaussians can be accomplished by iterative application of the following equations for all components $j \in \{1, \ldots, k\}$:

$$P(j \mid \mathbf{x}_i) := \frac{\pi_j \phi(\mathbf{x}_i; \theta_j)}{f_k(\mathbf{x}_i)}, \tag{3}$$

$$\pi_j := \frac{1}{n} \sum_{i=1}^{n} P(j \mid \mathbf{x}_i), \tag{4}$$

$$\mathbf{m}_j := \frac{\sum_{i=1}^{n} P(j \mid \mathbf{x}_i) \mathbf{x}_i}{n \pi_j}, \tag{5}$$

$$\mathbf{C}_j := \frac{\sum_{i=1}^{n} P(j \mid \mathbf{x}_i)(\mathbf{x}_i - \mathbf{m}_j)(\mathbf{x}_i - \mathbf{m}_j)^\top}{n \pi_j}. \tag{6}$$

As already mentioned in the previous section, the EM algorithm is not guaranteed to lead us to the best solution, where 'best' means the solution yielding maximal likelihood on $\mathbf{X}_n$ among all maxima of the likelihood.[1] The *good* thing is that *if* we are close to the global optimum of the parameter space, then it is very likely that by using EM we obtain the globally optimal solution.

3 Greedy Learning of Gaussian Mixtures

In this section we present our greedy method for learning mixtures of Gaussians. The basic idea is simple: Instead of starting with a (random) configuration of all components and improve upon this configuration with EM, we *build* the mixture component-wise. We start with the optimal one-compponent mixture, whose parameters are trivially computed. Then we start repeating two steps until a stopping criterion is met: (i) insert a new component and (ii) apply EM until convergence. The stopping criterion can implement the choice for a pre-specified number of components or it can be any model complexity selection criterion.

3.1 Motivation

Our 'greedy' approach to learning mixtures of Gaussians can be based on the following assumption:

Assumption 1 *The global optimum for a k-component mixture is reachable from the configuration obtained by optimally inserting a new component in the globally optimal configuration of the $(k-1)$-component mixture.*

With θ' is 'reachable' from θ we mean that if we start EM in θ then it will converge to θ'. Given that this assumption holds and that the solution for the one-component mixture is trivial to compute, the problem of finding a near optimal start configuration (so that EM will converge

---

[1] We implicitly assume throughout that the likelihood is bouded, by restricting the parameter space, and hence the maximum likelihood estimator is known to exist [12].

to the optimal solution) can be replaced with a sequence of $k-1$ optimal component insertion problems. The optimal component insertion problem involves a factor k fewer parameters than the original problem of finding a near optimal start configuration, we therefore expect it to be an easier problem. We recently applied a similar strategy to the k-means clustering problem [11], for which encouraging results were obtained.

Two recent theoretical results provide complementary motivation. Li has been proven [10] that for an *arbitrary* probability density function f there exists a sequence $\{f_i\}$ of finite mixtures such that $f_k(\mathbf{x}) = \sum_{i=1}^{k} \pi_i \phi(\mathbf{x}; \theta_i)$ achieves Kullback-Leibler (KL) divergence[2] $D(f \parallel f_k) \leq D(f \parallel g_P) + c/k$ for every $g_P = \int \phi(\mathbf{x}; \theta) P(d\theta)$. Hence, the difference in KL divergence achievable by k-component mixtures and the KL divergence achievable by any (possibly non-finite) mixture from the same family of components tends to zero with speed c/k (where c is a constant not dependent on k but only on the component family). Furthermore, it is shown that this bound is achievable by employing a greedy approach as discussed above.

Note that this result does not give direct support for our assumption but does tell us that we can 'quickly' approximate any density by the greedy procedure. Therefore, we might expect the results of the greedy procedure as compared to the standard (randomly initialized) EM approach to differ more when fitting mixtures with many components.

The sequence of mixtures generated by the greedy learning method can conveniently be used to guide a model selection process in the case of an unknown number of components. Recently a result of 'almost' concavity of the log-likelihood of a data set under the maximum-likelihood k-component mixture, as function of the number of components k was presented [3]. The result states that the first order Taylor approximation of the log-likelihood of the maximum likelihood models as function of k is concave under very general conditions. Hence, if we use a penalized log-likelihood model selection criterion based on a penalty term which is concave or linear in k then the penalized log-likelihood is almost concave. This implies that if the 'almost' concave turns out to be concave then there is only one peak in the penalized log-likelihood and hence this peak can be relatively easily identified, e.g. with our greedy construction method.

3.2   A General Scheme for Greedy Construction of Gaussian Mixtures

Let $\mathcal{L}(\mathbf{X}_n, f_k) = \sum_{i=1}^{n} \log f_k(\mathbf{x}_i)$ (we will just write $\mathcal{L}_k$ if no confusion arises) denote the log-likelihood of the data set $\mathbf{X}_n$ under the k-component mixture f_k. The greedy learning procedure outlined above can be summarized as follows:

1. Compute the optimal (yielding maximal log-likelihood) one-component mixture f_1. Set k:=1.

2. Perform a search to find the optimal new component $\phi(\mathbf{x}; \theta^*)$ and corresponding mixing weight α^*, where

$$\{\theta^*, \alpha^*\} = \arg\max_{\{\theta, \alpha\}} \sum_{i=1}^{n} \log\left[(1 - \alpha)f_k(\mathbf{x}_i) + \alpha\phi(\mathbf{x}_i; \theta)\right] \tag{7}$$

   with f_k fixed.

3. Set $f_{k+1}(\mathbf{x}) := (1 - \alpha^*)f_k(\mathbf{x}) + \alpha^*\phi(\mathbf{x}; \theta^*)$ and $k := k + 1$;

4. Update f_k using EM until convergence.

5. If a stopping criterion is met then quit, else go to step 2.

---

[2]The KL divergence is defined as: $D(f \parallel g) = \int_{\Omega} f(x) \log\left(f(x)/g(x)\right) \mathrm{d}x$ where Ω is the domain of the densities f and g, see [4] for details.

The stopping criterion in step 5 can be used to force the algorithm to find a mixture of a pre-specified number of components. Of course, step 5 may also implement any kind of model selection criterion. Note that step 4 may also be implemented by other algorithms than EM.

One may note similarity to the Vertex Direction Method (VDM) [2, 12, 23] to learn Gaussian Mixtures. Indeed, VDM may be regarded as a specific choice for the search of step 2, as we explain in the discussion in Section 5. In the rest of this section we are concerned with the search in step 2.

3.3 Searching for New Components

Suppose we have obtained a k-component mixture f_k, the quest is to find the component characterized by equation (7). It is easily shown that if we fix f_k and ϕ then $\mathcal{L}_{k+1}$ is concave as function of α only, allowing efficient optimization. For example, in [22] a second order Taylor approximation was used. The concavity follows trivially by noting that the second derivative is everywhere non-positive [2]. However, $\mathcal{L}_{k+1}$ as function of θ can have multiple maxima. Hence, we have to perform a global search among the new components in order to identify the optimum.

In our new algorithm, the global search for the optimal new component is achieved by starting $m \geq 1$ 'partial' EM searches. By a 'partial' EM search we mean that we fix f_k and optimize over θ and α only. The use of partial EM searches is dictated by the general scheme above, for which Li's results [10] hold. One may wonder why we would use such partial searches, since we might as well optimize over all parameters of the resulting $(k + 1)$-component mixture. The answer is that (i) if we would update all parameters then the number of computations needed in each search would be $O(nk)$ instead of $O(n)$ if we perform partial EM updates and (ii) we would not adhere to the general scheme anymore.

Each partial search starts with a different initial configuration. After these multiple partial searches we end up with m 'candidate' new components together with their mixing weights. We pick that candidate component $\phi(\mathbf{x}; \hat{\theta})$ that maximizes the likelihood when mixed into the previous mixture by a factor $\hat{\alpha}$ as in (7). Then, in step 3 of the general algorithm, instead of inserting the global maximum component $\phi(\mathbf{x}; \theta^*)$ with a factor α^* we insert $\phi(\mathbf{x}; \hat{\theta})$ with a factor $\hat{\alpha}$.

An implementation of VDM for mixtures of univariate Gaussians has been published in [7]. There, the search for new components was implemented by uniformly gridding the data space. Note that this approach scales exponentially with the dimensionality of the data space. In Section 3.3.1 we discuss the step 2 as proposed in [22] and its drawbacks. Section 3.3.2 presents our proposal for an alternative search procedure.

3.3.1 Every Data Point Generates a Candidate

In [22] it is proposed to use n candidate components. Every data point is the mean of a corresponding candidate. All candidates have the same covariance matrix $\sigma^2 \mathbf{I}$, where σ is taken proportional to the value that minimizes the mean integrated squared error of a non-parametric density estimator:

$$\sigma = \beta \Big[\frac{4}{(d+2)n} \Big]^{1/(d+4)}, \tag{8}$$

where β is set to half of the largest singular value of the covariance matrix of $\mathbf{X}_n$. For each candidate component the mixing weight α is set to the mixing weight maximizing the second order Taylor approximation around $1/2$ of the log-likelihood as function of α (for fixed component parameters θ and a fixed mixture f_k). The candidate yielding highest log-likelihood when inserted in the existing mixture f_k is selected. The selected component is then updated using partial EM steps before it is actually inserted into f_k to give f_{k+1}. Note that for every component

insertion that is performed all point-candidates are considered. There are two main drawbacks to the above mentioned method:

1. Using n candidates in each step results in a time complexity $O(n^2)$ for the search which is unacceptable in many applications. $O(n^2)$ computations are needed since the likelihood of every data point under every candidate component has to be evaluated.

2. By using fixed small (referring to the size of the determinant of the covariance matrix) candidate components the method sometimes keeps inserting small components in high density areas while in low density areas the density is modeled poorly. We observed this experimentally. Larger components that give greater improvement of the mixture are not among the candidates, nor are they found by the EM procedure following the component insertion.

Both better and faster performance might be achieved by using a smaller set of candidates that are appropriate for the current mixture.

    We conducted some experiments where instead of using all data points as means for the new components we used a smaller set given by the centroids of the nodes in a kd-tree build for the data. Originally kd-trees [1] were designed to speed-up the execution of nearest neighbor queries, range queries and related problems. A kd-tree defines a recursive binary partitioning of a k-dimensional data set, where the root node contains all data. Sproull [16] proposed to partition the data at each node by cutting with a hyper-plane perpendicular to the direction with greatest variance of the data present in that node, i.e. the direction of the first Principal Component [9]. One may view the resulting procedure as a 'nested' Principal Component Analysis. If we fully expand the tree, the leaves of the tree are given by the individual data points. We can regard each node in the tree as a bucket containing a portion of the data. Every layer of the tree gives a partitioning of the data. It turns out that these partitions provide quite reasonable *clusterings* of the data in terms of mean squared distance from the data to their closest cluster center (the cluster centers are given by the bucket means), see also [11]. In our experiment we built the tree for just several layers and used the centroids of the buckets of the leaves of the tree as means for the candidate components. This gave drastic speed-up while affecting performance only slightly.

    However, this method introduces the question of how many layers of the tree to generate. Second, still for every insertion the same initial candidates, namely all bucket centroids, are considered. Third, this approach does not provide a way to safeguard for the aforementioned second drawback.

3.3.2   A new search procedure using local kd-trees

Based on our experience with the methods above, we propose a new search procedure to look for (globally) optimal new mixture components. Two observations motivate the new search method:

1. The size (i.e. the determinant of the covariance matrix) of the inserted component (after the partial EM steps) is generally smaller than the size of the existing components of the mixture.

2. It seems (we do not have a proof) that the smaller (the determinant of the covariance of) the new component gets, the larger risk we run to end up in a local maximum when applying EM to find the globally optimal new component.

In our search strategy we account for both observations by (i) setting the size of the initial candidate configurations to a value related to and in general smaller than the scale of the components

in the existing mixture and (ii) increasing the number of candidate insertion configurations linearly in k.

For each insertion problem, our method constructs one local kd-tree for each existing mixture component, which is expanded up to two layers. Both the means and the covariance matrices of the candidate components are then initialized according to all six 'buckets' of the kd-tree. Below we discuss the procedure in detail.

Construction of Candidate Components   Based on the posterior distributions, we partition the data set $\mathbf{X}_n$ in k disjoint subsets $A_i = \{\mathbf{x} \in \mathbf{X}_n : P(i \mid \mathbf{x}) = \max_j \{P(j \mid \mathbf{x})\}\}$ with $1 \leq i \leq k$. If one is using EM for step 4, then the posteriors $P(i \mid \mathbf{x})$ are available directly since we already computed them for the EM updates of the k-component mixture. For each set A_i we construct the first six nodes (two residing in the first layer plus four residing in the second layer) of a local kd-tree $T(A_i)$ based on the data $\mathbf{x} \in A_i$ only. Then, for each of the $6k$ nodes we initialize a new component with the mean and covariance of the data present in that node. The initial mixing weights for candidates generated from A_i are set to $\pi_i/2$. The reader may have noticed that using only two layers of the local kd-trees is somewhat arbitrary. However, experiments indicated that using more layers does not improve results significantly while it slows down the algorithm.

The initial candidates can be replaced easily by candidates that yield higher likelihood when mixed into the existing mixture. To obtain these better candidates we apply an EM algorithm again, but now to update only the candidate components and their mixing weights as to maximize $\mathcal{L}_{k+1}$, while keeping the existing mixture f_k fixed. Each iteration of these partial updates takes $O(nk)$ computations, since we have to evaluate the likelihood of each datum under each of the $6k$ candidate components. In the following section we discuss how we can reduce the amount of computations needed by a factor k resulting in only $O(n)$ computations to perform one iteration of the partial updates.

We stop the partial updates if the change in log-likelihood of the resulting $(k+1)$-component mixtures drops below some threshold or if some maximal number of iterations is reached. After these partial updates we set the new component $\phi(\mathbf{x}; \theta_{k+1})$ as the candidate that maximizes the log-likelihood when mixed into the existing mixture.

As an example we included Figure 1, which depicts the evolution of a solution for artificially generated data as used in the experiments described in Section 4.1. The concepts 'eccentricity' and 'separation' are as in Section 4.1. We generated 400 2-dimensional data points from a 4-component mixture, with separation $c = 1$ and maximum eccentricity $e = 15$. On the left the mixtures $f_1, \ldots, f_4$ are depicted by their mean and an ellipse which has the eigenvectors of the covariance matrix as axes and radii of twice the squareroot of the corresponding eigenvalue. On the right we plot the candidate new components after the partial EM steps.

3.3.3   Speeding up the Partial EM Searches

In order to achieve $O(n)$ time complexity for the $6k$ partial EM searches initiated at the $6k$ initial candidates, we make the simplifying assumption that the support (non-zero density points in $\mathbf{X}_n$) for a candidate component $\phi(\mathbf{x}, \theta)$ constructed based on a node of a local kd-tree $T(A_i)$ (see previous section) is located inside A_i. This simplifying assumption allows us to base the partial updates for a candidate from A_i purely on the data in A_i, see the EM update equations in section 2.

We want to maximize the (log-)likelihood $\mathcal{L}_{k+1}$ of the data in the resulting mixtures if we insert the candidates in the existing mixture. Of course, this is equivalent to maximizing $\mathcal{L}_{k+1} - \mathcal{L}_k$. Using the above assumption, the change in log-likelihood of $\mathbf{X}_n$ under the resulting
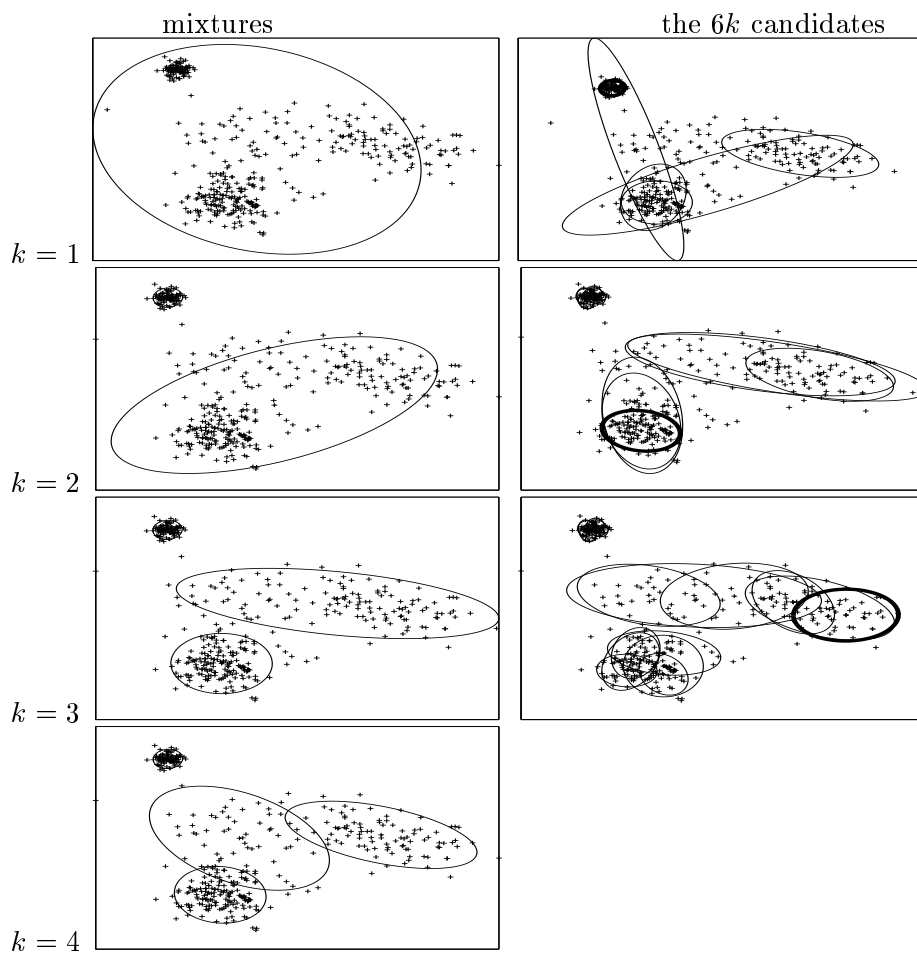
Figure 1: A trace of the construction of a 4-component mixture distribution. Left are the mixtures $f_1, \ldots, f_4$, right are the candidate new components in each component allocation step. Thick lines indicate the selected candidate.

$6k$ new $(k+1)-$component mixtures is conveniently computed as follows:

$$\mathcal{L}_{k+1} - \mathcal{L}_k = \sum_{\mathbf{x} \in \mathbf{X}_n} \log \frac{f_{k+1}(\mathbf{x})}{f_k(\mathbf{x})} = \sum_{\mathbf{x} \in \mathbf{X}_n} \log \frac{(1-\alpha)f_k(\mathbf{x}) + \alpha\phi(\mathbf{x};\theta)}{f_k(\mathbf{x})} = \tag{9}$$

$$\sum_{\mathbf{x} \in A_i} \log \frac{(1-\alpha)f_k(\mathbf{x}) + \alpha\phi(\mathbf{x};\theta)}{f_k(\mathbf{x})} + \sum_{\mathbf{x} \notin A_i} \log \frac{(1-\alpha)f_k(\mathbf{x}) + \alpha\phi(\mathbf{x};\theta)}{f_k(\mathbf{x})} \tag{10}$$

$$= \sum_{\mathbf{x} \in A_i} \Big[\log(1-\alpha) - \log \frac{(1-\alpha)f_k(\mathbf{x})}{f_k(\mathbf{x}) + \alpha\phi(\mathbf{x};\theta)} \Big] + \sum_{\mathbf{x} \notin A_i} \log(1-\alpha) \tag{11}$$

$$= n\log(1-\alpha) - \sum_{\mathbf{x} \in A_i} \log(1 - P(k+1 \mid \mathbf{x})) \tag{12}$$

The first two equalities follow from the definitions, the third is obtained by splitting the sum for data inside and data outside A_i. The left term of (11) is obtained from the left term of (10) by (i) multiplying both parts of the fraction with $(1-\alpha)$ then (ii) separating a $\log(1-\alpha)$ term and (iii) changing the sign of the log to swap the fraction. The right term of (11) equals the right term of (10) by our assumption that $\phi(\mathbf{x};\theta) = 0$ for $\mathbf{x} \notin A_i$. The last equation follows trivially now, given that the posterior $P(k+1 \mid \mathbf{x})$ is defined as before. It is clear that to evaluate (12) for all $6k$ candidates costs only $O(n)$ operations, since every $\mathbf{x}$ is used only for six candidates.

Let r_i denote the fraction of the data present in A_i, thus $\sum_{i=1}^{k} r_i = 1$. The partial EM updates of each of the six candidates based on A_i only take $6r_i n \in O(r_i n)$ computations. Hence, in total we need $O(\sum_{i=1}^{k} r_i n) = O(n)$ computations. The same analysis holds for the computation of the change in log-likelihood.

Note that this speed-up is only possible because we generate the new components based on the local kd-trees. Hence, this is another advantage of the new method over the one in [22]. If we would constrain the latter also to consider only $6k$ candidates, then still the computational cost of a single insertion step would be $O(nk)$. In the last paragraph of Section 4.1 we describe an experiment where we compare the performance of the latter method (using n candidates) with that of the new method. The new method clearly outperforms the latter. Since using fewer candidates will not increase performance we felt it is not needed to compare the new method with the latter using only $6k$ candidates.

3.3.4   Total Run Time

As mentioned in the introduction, the total run time of the algorithm is $O(k^2 n)$. This is due to the updates of the mixtures f_i, which cost $O(ni)$ computations each if we use EM for these updates. Therefore, summing over all mixtures we get $O(\sum_{i=1}^{k} ni) = O(nk^2)$. This is a factor k slower than the standard EM procedure. The run times of our experiments confirm this analysis, the new method performed on average about $k/2$ times slower than standard EM.

Note that if we do not use the aforementioned speed-up, the run time would remain $O(k^2 n)$ (since in that case every component allocation step would cost $O(kn)$, just as the preceding EM updates for the current mixture). Therefore, the speed-up is not 'essential' but allows in practice for significant faster performance. Note that if a faster alternative is used instead of EM to update the mixtures f_i in step 4 of the general scheme, then the benefit of using the speed-up becomes greater.

4   Experimental Demonstration

In this section we present results of three experiments. All experiments compare the results as obtained with standard EM and the greedy approach presented here.
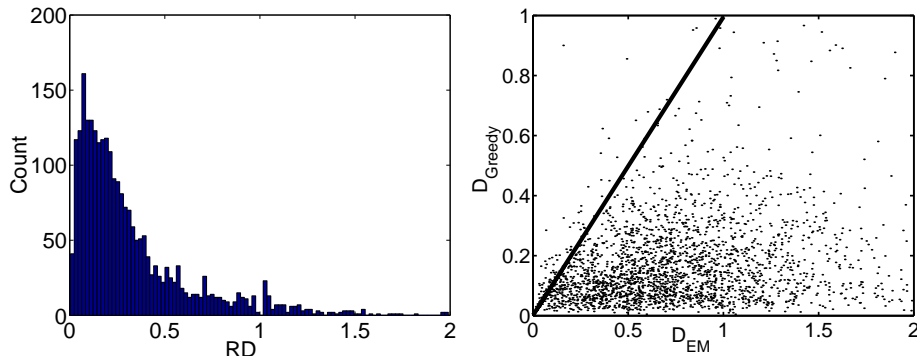
Figure 2: Histogram of RD (left) and D_{Greedy} as function of D_{EM} (right) the solid line identifies $RD = 1$.

4.1   Artificial Data Drawn from Gaussian Mixtures

In this experiment we generated data sets of 400 points in a $\mathbb{R}^d$ space with $d \in \{2, 3, 4, 5\}$. The data was drawn from a Gaussian mixture of $k \in \{4, 6, 8, 10\}$ components. The separation of the components was chosen $c \in \{1, 2, 3, 4\}$. A separation of c means:

$$\forall_{i \neq j} : \quad \| \mu_i - \mu_j \|^2 \geq c \max_{\{i,j\}} \{\text{trace}(\mathbf{C}_i), \text{trace}(\mathbf{C}_j)\}. \tag{13}$$

For each mixture configuration we generated 50 data sets. We allowed a maximum eccentricity (i.e. largest singularvalue of the covariance matrix over the smallest) of 15 for each component. Also, for each generated mixture, we generated a test set of 1000 points not presented to the mixture learning algorithms.

In the comparison below, we compare the log-likelihood of the test sets under the mixtures provided by both methods f_{EM} and f_{Greedy} with the log-likelihood under the generating mixture f. Let $D_{EM} = \mathcal{L}(\mathbf{X}_n, f) - \mathcal{L}(\mathbf{X}_n, f_{EM})$, and similarly for D_{Greedy}. The differences D_{EM} and D_{Greedy} provide empirical estimates of the KL divergence to the generating mixture, where the integral is replaced by a sum over the observed data. Let $RD = D_{Greedy}/D_{EM}$ denote the 'Relative Difference' in KL divergence.

Overall Results   For 637 of the 3200 (19.9%) experiments $RD \in (0.98, 1.02)$. For 132 of the 3200 (4.13%) experiments $RD \in [1.02, 2)$. For 17 (0.53%) experiments $RD \geq 2$ with a maximum of 5.69. The remaining 75.44% of the experiments resulted in $RD < .98$ The distribution of RD over all experiments with RD outside $(0.98, 1.02)$ (this involves 80.1% of all experiments) is visualised in the left plot of Figure 2 by a histogram of RD. In the right plot of Figure 2 we plot D_{Greedy} as function of D_{EM} for all experiments with $D_{Greedy} < 1$ and $D_{EM} < 2$ (which includes 97.37% of all experiments). Observe that if we would make histograms of D_{Greedy} for ranges of D_{EM} that they would be shaped again like the overall histogram. This indicates that the mean of D_{Greedy} in such a range does not correspond to the peak of the histogram. One may take this into account when interpreting the means presented in the tables of Figure 4.

We also compared the results of the new algorithm to using EM for multiple random initializations. We recorded for every data set the time spend by our new algorithm and allowed as many restarts of EM as possible within the time spent by our algorithm. We then kept the mixture maximizing likelihood on the data and compared the resulting approximated KL divergence D_{Mult} (defined as before) with D_{Greedy}. Figure 3 provides similar plots as in Figure 2. In the histogram, we plotted for reasons of exposition only results for experiments that yield $D_{Greedy}/D_{Mult} < 2$ (this only excludes 1.25 % of all experiments) and $D_{Greedy}/D_{Mult} \notin (0.98, 1.02)$ (this excludes 35.6% of all experiments). Typically the number of
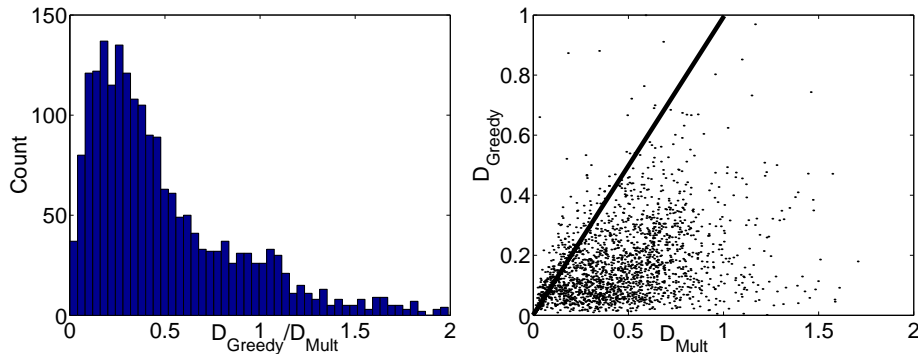
Figure 3: Histogram of D_{Greedy}/D_{Mult} (left) and D_{Greedy} as function of D_{Mult} (right) the solid line identifies $D_{Greedy} = D_{Mult}$.

$d = 2$	$c = 1$	2	3	4	$d = 3$	$c = 1$	2	3	4
$k = 4$	0.96	0.59	0.54	0.48	$k = 4$	0.79	0.55	0.59	0.64
6	0.74	0.34	0.29	0.19	6	0.61	0.43	0.29	0.29
8	0.64	0.32	0.18	0.14	8	0.61	0.30	0.19	0.30
10	0.56	0.27	0.21	0.29	10	0.61	0.27	0.21	0.30

| $d = 4$	$c = 1$	2	3	4	$d = 5$	$c = 1$	2	3	4
$k = 4$	0.77	0.60	0.74	0.74	$k = 4$	0.87	0.69	0.82	0.57
6	0.60	0.47	0.33	0.43	6	0.59	0.46	0.48	0.46
8	0.53	0.38	0.26	0.31	8	0.51	0.42	0.40	0.32
10	0.61	0.35	0.34	0.33	10	0.57	0.44	0.49	0.49

Figure 4: Tables giving averages (over 50 experiments each) of RD for different experimental settings.

runs is between one and ten, but of course this depends on the number of mixture components used, see Section 3.3.4.

We conclude that the new method still outperforms EM significantly, even if multiple restarts are used. The median of improvement in the KL divergence histogram is approximately around 0.25, while it was around 0.10 when using only one EM run. Also we observe that the histogram is more peeked when only one EM run is used. The percentage of the experiments for which the (approximated) KL divergences are close (their ratio is within the $(0.98, 1.02)$ interval) is larger when using multiple EM runs: 19.9 % for one run and 35.6 % for multiple runs. All three effects are as expected.

Results split out   Below we provide tables summarizing average RD obtained for the different types of generating mixtures. We conclude that the new algorithm gives greater improvement as the separation and the number of components in the generating mixture increase.

Comparison to using all data as candidates   In our experiments described above, we also compared the performance of the new method to the performance of the method proposed in [22]. Here, we consider again D_{Greedy} and now we compare it with the approximated KL divergence D_{fixed} when using the fixed candidate set as proposed in [22]. For 2062 (i.e. 64.44%) experiments the fraction D_{Greedy}/D_{fixed} was within $[0.95, 1.05]$. Of the remaining 1138 experiments, for 1122
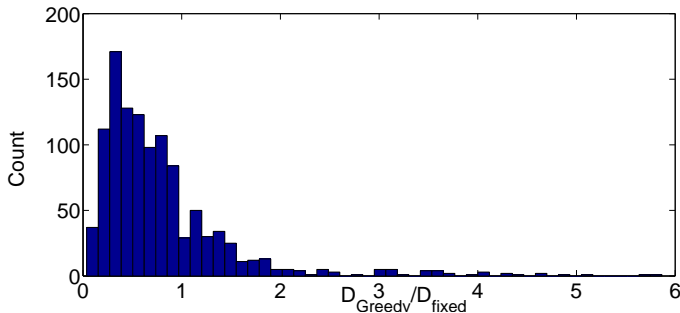
Figure 5: Comparing KL divergences for the 'old' (D_{fixed}) and the 'new' (D_{Greedy}) component insertion methods.

(98.59%) the fraction was smaller than 6. Figure 5 shows a histogram of the fraction for those 1122 experiments. Observe that *if* the new method performs differently, it generally allows for a significant reduction of the KL divergence.

4.2   Image reconstruction using MPPCA

Here, we compare image reconstruction results when using our greedy and standard EM method to learn Mixtures of Probabilistic Principal Component Analyzers (MPPCA). The MPPCA model is used to obtain low dimensional descriptions of the images from which we can reconstruct the images. [3]

In [18] MPPCA is introduced as a generative probabilistic model. The model is a constrained Gaussian mixture, the $d \times d$ covariance matrices are restricted to the form $\mathbf{C}_i = \sigma_i^2 \mathbf{I} + \mathbf{W}_i \mathbf{W}_i^\top$ for component i, where $\mathbf{W}_i$ is a $d \times q$ matrix. The matrices $\mathbf{W}_i$ span local principal subspaces. Our greedy approach for mixture learning extends naturally to the MPPCA model. For the candidates initialize $\mathbf{W}$ with the first q eigenvectors of the covariance matrix $\mathbf{C}_{full}$ of the data in the node of the kd-tree and initialize σ^2 with the mean of the smallest $d - q$ eigenvalues of $\mathbf{C}_{full}$.[4]

Once a MPPCA model is fitted to the data (64×64 pixel gray-value images in our case) the data can be stored and reconstructed using the subspace with maximum posterior probability. This is achieved by encoding the index of the used subspace and encoding the q 'local' coefficients identifying the projection of the image onto that subspace. As an error measure we use the total (summed over all pixels) absolute difference in intensity between pixels in the original images and the reconstructed images. In the table in Figure 6 we show results of the experiment compared to results obtained when using the MPPCA for five different random parameter initializations. The 698 images we used were obtained from the Isomap [17] webpage (see http://isomap.stanford.edu). Several typical images and some of the means of the mixture components are shown in Figure 7. For the reconstruction experiment, we first reduced the dimensionality of the data space to speed-up the experiment. It is well known that n points in $\mathbb{R}^m$ with $m > n$ are embedded in an at most $n - 1$ linear subspace of $\mathbb{R}^m$. Therefore, dimensionality reduction of the data from $64^2 = 4096$ to 697 dimensions is trivial. Further linear dimensionality reduction from 697 to 59 dimensions was applied by meansof Principal Component Analysis keeping 95.1% of the total variance. In this 59 dimensional subspace we learned MPPCA models consisting of 15 components and with $q = 3$.

---

[3]Spatial relations among the different local linear models are not used here. However, this is a subject of current research in our group. See for example [20] where several local linear models are joined to form a global low dimensional coordinate system.

[4]In fact, the columns of $\mathbf{W}$ must be scaled by $\sqrt{\lambda - \sigma^2}$, where λ is the eigenvalue corresponding to the eigenvector in a column, see [18] for details.

method	mean error	std. dev.	log-likelihood
Greedy	257.04	82.60	77.50
Standard	310.93	100.20	74.33
Standard	308.86	99.41	72.92
Standard	353.24	135.91	74.09
Standard	299.44	73.35	74.21
Standard	319.86	97.02	73.55

Figure 6: Average per-image reconstruction errors when compressing the 698 images with a single MPPCA model.
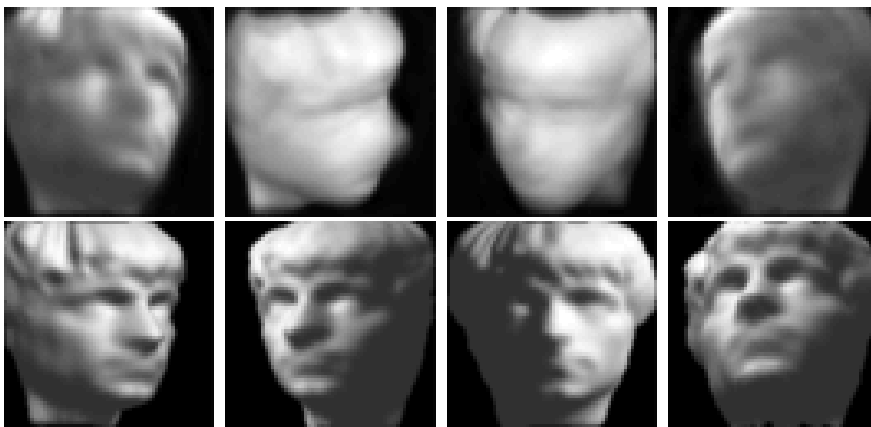
Figure 7: The means of some mixture components (upper row) and several images (bottom row).
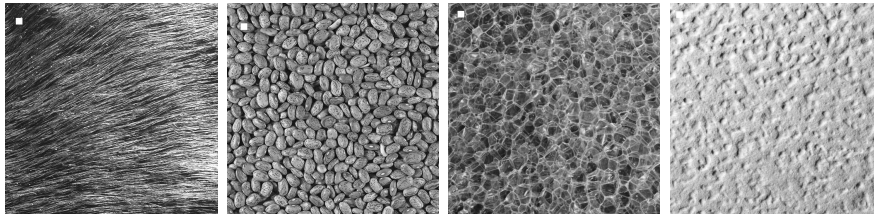
Figure 8: Several Brodatz textures, the white squares indicate the patch size.

k	2	3	4	5	6
Greedy	0.25	0.46	0.67	0.78	0.88
Standard	0.47	0.77	1.04	1.28	1.34
Uniform	1	1.59	2	2.32	2.58

Figure 9: Average conditional entropy for different values of k for the standard EM and the proposed greedy method to learn Gaussian mixtures compared with the conditional entropy for clusters distributing uniform over textures.
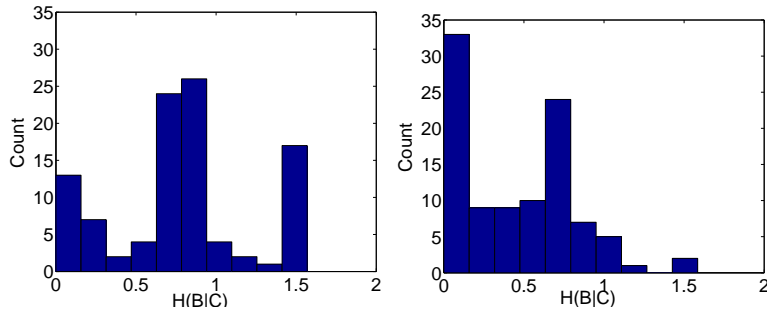
4.3   Texture Segmentation

In this experiment the task is to cluster 16×16 gray-valued images. The images are patches of 512×512 images of 37 Brodatz textures. A small selection of these textures is provided in Figure 8. The idea is that patches from the same texture display strong similarity while patches from different textures are assumed to be dissimilar. The clustering is obtained using a mixture of Gaussians fitted on the patches. The experiment described below compares results obtained when learning the mixture using our new method and the standard EM method.

The number of textures involved in each experiment is denoted by $k \in \{2, 3, 4, 5, 6\}$. For each value of k we constructed 100 data sets by randomly extracting 500 patches from k randomly selected textures. The patches are represented as 256 dimensional vectors, which we projected linearly to a lower dimensional subspace by means of PCA as to retain 80% of the total variance. In this lower dimensional space (typically somewhere between 10 and 70 dimensions were retained) we learn a k component Gaussian mixture. The clustering is now obtained by assigning each patch to the maximum posterior mixture component. Our experiment is based on the experiments described in [15], there the MPPCA model is used to learn restricted Gaussian mixtures.

To evaluate a given clustering we used the conditional entropy $H(B|C)$ that measures the entropy of the texture labels conditioned on the cluster labels. In Appendix A we discuss this measure in more detail. In Figure 9 we provide a table with the averages of $H(B \mid C)$ over 100 experiments for each value of k. Figure 10 illustrates the difference between the greedy and the standard EM method in more detail for $k = 3$. The left plot shows 3 modes:

1. Close to zero: Good segmentation, every component captures a texture. The conditional entropy is close to zero since there is almost no uncertainty about the texture class if we know the maximum posteriori mixture component.

2. Close to 2/3 bit: One texture is separated and the two others are confused. In 1/3 of the cases we have very low entropy (the separated texture) and in 2/3 of the cases we have entropy close to 1 bit (the two confused textures). Since two textures are confused we need on average approximately one bit to indicate which one is used. Taking the average over all cases we arrive at approximately 2/3 bit.

Figure 10: Histograms of $H(B \mid C)$ for $k = 3$ comparing the standard EM (left) and new greedy method (right) to learn the Gaussian mixture.

3. Close to 3/2 bits: Very poor segmentation, all textures are confused. Note that $H(B \mid C) \approx 1.59$ bits for clusters that distribute uniformly over the textures.

In the right plot, showing performance of the greedy method, we see that the third mode has almost disappeared illustrating the superior performance of the mixture learned with the greedy method.

5 Discussion and Conclusions

Discussion   Both VDM [2, 12, 23] and the proposed method are instantiations of the more general greedy scheme given in section 3.2 (although VDM skips step 4 of the scheme). VDM makes use of the directional derivative $D_{f_k}(\phi)$ of the log-likelihood for the current mixture f_k, where $D_{f_k}(\phi) = \lim_{\alpha \to 0}[\mathcal{L}(\mathbf{X}_n, (1 - \alpha)f_k + \alpha\phi) - \mathcal{L}(\mathbf{X}_n, f_k)]/\alpha$. VDM proceeds by picking the ϕ^* that maximizes $D_{f_k}(\phi)$ and inserting this in the mixture with a factor α^* such that $\alpha^* = \arg\max_\alpha\{\mathcal{L}(\mathbf{X}_n, (1 - \alpha)f_k + \alpha\phi^*\}$. Using the directional derivative at the current mixture can be seen as an instantiation of step 2 of the general scheme. The optimization over both ϕ and α is replaced by (i) an optimization over $D_{f_k}(\phi)$ (typically implemented by gridding the parameter space) and then (ii) an optimization over α. Note that by 'moving in the direction of maximum $D_{f_k}(\phi)$' does not guarantee that we move in the direction of maximum improvement of log-likelihood if we optimize over α subsequently. See [13] for an overview of VDM and other mixture learning methods.

The approximation result of Li [10], which applies to the more general scheme provided in Section 3.2, is encouraging for the use of greedy methods to learn finite mixtures. However, the result does not guarantee that each element f_k of the sequence of mixtures $\{f_i\}$ produced by the greedy algorithm is close to the k-component maximum likelihood mixture.

Recently, several other new methods to learn mixtures (of Gaussians) were proposed among which we mention [5, 8, 19]. The first tries to overcome the difficulties of learning mixtures of Gaussians in high dimensional spaces. By projecting the data to a lower dimensional subspace (which is chosen uniformly randomly!), finding the means in that space and then projecting them back, the problems of high dimensionality are reduced. The last two methods try to overcome the dependence of EM on the initial configuration as does our method. In [19] split and merge operations are applied to local optima solutions found by applying EM. The split and merge operations constitute jumps in the parameter space that allow the algorithm to jump from a local optimum to a better region in the parameter space. By then reapplying EM a better (hopefully global) optimum is found. An important benefit of our new method over [19] is that the new algorithm produces a sequence of mixtures that can be used to perform model complexity selection as the mixtures are learned. For example a kurtosis-based selection

criterion, like the one in [21], can be used here. In [8] it is proposed to start with a large number k_{max} of mixture components and to successively annihilate components with small mixing weights. This approach can be characterized as 'pruning' a given mixture, where our approach can be characterized as 'growing' a mixture. There, also a sequence of mixtures of different number of components is generated and it is exploited by integrating the process of model selection and fitting of the mixtures. Note however that

1. In general we may not know how to set k_{max} if we do not know what the 'correct' number k of components is.

2. Suppose that the 'correct' number of components can be identified (it may be simply known or inferred using model selection techniques). For the pruning approach, the considered mixtures f_i consist of $k \leq i \leq k_{max}$ components, so the EM updates cost *at least* $O(kn)$ since $k \leq i$. For our growing approach all considered mixtures consist of *at most* k components such that the EM updates of those mixtures cost *at most* $O(kn)$.

Also, Bayesian methods are used to learn Gaussian mixtures. For example in [14], a reversible jump Markov Chain Monte Carlo (MCMC) method is proposed. There, the MCMC is allowed to jump between parameter spaces of different dimensionality (i.e. parameter spaces for mixtures consisting of differing number of components). However, the experimental results reported in [14] indicate that such sampling methods are rather slow as compared to constructive maximum likelihood algorithms. It is reported that about 160 'sweeps' per second are performed on a SUN Sparc 4 workstation. The experiments involve 200.000 sweeps, resulting in about 20 minutes run time. Although it is remarked that the 200.000 sweeps are not needed for reliable results, it contrasts sharply with the 2.8 seconds and 5.7 seconds run time (allowing respectively about 480 and 960 sweeps) of the standard EM and our greedy EM in a similar experimental setting executed also on a SUN Sparc 4 workstation.

Conclusions   We proposed a greedy method to learn mixtures of Gaussians that has run time $O(nk^2)$. As compared to the standard EM algorithm we observe:

1. The proposed algorithm is deterministic.

2. Experiments on three different problems (density estimation, image reconstruction and texture segmentation) show superior performance of the new method while run time is increased only by a factor linear in k.

As compared to the method proposed in [22] we note:

1. The $O(n^2 k^2)$ time complexity has been reduced by a factor n.

2. The somewhat arbitrary choice for spherical candidate components with fixed variance and their bandwith has been replaced by a search for candidate components that depends on the current mixture.

3. Experiments suggest that if the methods yield different performance, then the new method generally outperforms the old one.

4. The new component insertion method extends naturally to the MPPCA model.

Software implementing our new algorithm in MATLAB is available by email from the first author (jverbeek@science.uva.nl).

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] D. Böhning. A review of reliable maximum likelihood algorithms for semiparametric mixture models. *J. Statist. Plann. Inference*, 47:5–28, 1995.

[3] I. V. Cadez and P. Smyth. On model selection and concavity for finite mixture models. In *Proc. of Int. Symp. on Information Theory (ISIT)*, available at `http://www.ics.uci.edu/ icadez/publications.html`, 2000.

[4] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[5] S. Dasgupta. Learning mixtures of Gaussians. In *Proc. IEEE Symp. on Foundations of Computer Science*, New York, October 1999.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Statist. Soc. B*, 39:1–38, 1977.

[7] R. DerSimonian. Maximum likelihood estimation of a mixing distribution. *J. Roy. Statist. Soc. C*, 35:302–309, 1986.

[8] M.A.T. Figueiredo and A.K. Jain. Unsupervised learning of finite mixture models. *to appear in IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[9] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[10] J. Q. Li and A. R. Barron. Mixture density estimation. In *Advances in Neural Information Processing Systems 12*. The MIT Press, 2000.

[11] A. Likas, N. Vlassis, and J.J. Verbeek. The global k-means clustering algorithm. Technical report, Computer Science Institute, University of Amsterdam, The Netherlands, February 2001. IAS-UVA-01-02.

[12] B. G. Lindsay. The geometry of mixture likelihoods: a general theory. *Ann. Statist.*, 11(1):86–94, 1983.

[13] G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, New York, 2000.

[14] S. Richardson and P. J. Green. On Bayesian analysis of mixtures with an unknown number of components. *J. Roy. Statist. Soc. B*, 59(4):731–792, 1997.

[15] D. de Ridder, J. Kittler, and R.P.W. Duin. Probabilistic pca and ica subspace mixture models for image segmentation. In M. Mirmehdi and B. Thomas, editors, *British Machine Vision Conference*, pages 112–121, 2000.

[16] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[17] J.B. Tenenbaum, V. de Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[18] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482, 1999.

[19] N. Ueda, R. Nakano, Z. Ghahramani, and G. E. Hinton. SMEM algorithm for mixture models. *Neural Computation*, 12:2109–2128, 2000.

[20] J. J. Verbeek, N. Vlassis, and B. Kröse. A soft k-segments algorithm for principal curves. In *Proc. Int. Conf. on Artificial Neural Networks*, pages 450–456, Vienna, Austria, August 2001.

[21] N. Vlassis and A. Likas. A kurtosis-based dynamic approach to Gaussian mixture modeling. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 29:393–399, 1999.

[22] N. Vlassis and A. Likas. A greedy EM algorithm for Gaussian mixture learning. Technical report, Computer Science Institute, University of Amsterdam, The Netherlands, September 2000. IAS-UVA-00-08, to appear in Neural Processing Letters.

[23] C.F. Wu. Some algorithmic aspects of the theory of optimal designs. *Annals of Statistics*, 6:1286–1301, 1978.

A   Conditional entropy for cluster evaluation

To evaluate a given clustering we consider the $k \times k$ confusion matrix A, where A_{ij} denotes how many patches of texture j are assigned to cluster i. Let B be a random variable which ranges over the k textures and C a random variable ranging over the k clusters. As a measure of how informative a given clustering is we compute the conditional entropy $H(B \mid C)$. Note that this quantity is directly related to the mutual information between the two variables $I(B; C) = H(B) - H(B \mid C)$. Hence the lower $H(B \mid C)$ the higher the mutual information is and thus the better the clustering is. Recall that the entropy of a discrete random variable X is a measure of uncertainty in X and is defined as

$$H(X) = \sum_x p(X = x) \log p(X = x), \tag{14}$$

and the conditional entropy is defined as

$$H(X \mid Y) = E_Y H(X \mid Y = y) = \sum_{x,y} p(Y = y, X = x) \log p(X = x \mid Y = y). \tag{15}$$

If we set the logartihm to base 2, the (conditional) entropy measures how many bits are needed on average per outcome to encode a string of outcomes of the variable if we use an optimal code scheme. See [4] for an excellent discussion of these concepts.

The $500k$ images in each experiment provide $500k$ joint realizations of the variables B and C resulting in the matrix A. The matrix A gives empirical estimates of

$$p(B = b \mid C = c) \approx A_{cb} / \sum_{b'} A_{cb'} \tag{16}$$

and

$$p(C = c) \approx \sum_{b'} A_{cb'} / \sum_{b'c'} A_{c'b'}. \tag{17}$$

We compute $H(B \mid C)$ as follows:

$$H(B \mid C) = -\sum_c p(C = c) \sum_b p(B = b \mid C = c) \log_2 p(B = b \mid C = c) \tag{18}$$
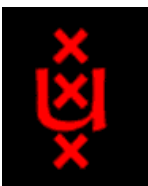
Acknowledgements

Intelligent
Autonomous
Systems

IAS reports

This report is in the series of IAS technical reports. The series editor is Stephan ten Hagen (`stephanh@science.uva.nl`). Within this series the following titles appeared:

[1] N. Vlassis, A. Likas, and B. Kröse. *Multivariate Gaussian mixture modeling with unknown number of components*. Technical Report IAS-UVA-00-04, Intelligent Autonomous Systems Group, University of Amsterdam, April 2000.

[2] N. Vlassis and A. Likas. *An EM-VDM algorithm for Gaussian mixtures with unknown number of components*. Technical Report IAS-UVA-00-05, Intelligent Autonomous Systems Group, University of Amsterdam, May 2000.

[3] N. Vlassis . *A greedy-EM algorithm for Gaussian mixture learning*. Technical Report IAS-UVA-00-08, Intelligent Autonomous Systems Group, University of Amsterdam, September 2000.

[4] J.J. Verbeek, N. Vlassis and B.J.A Kröse. *A k-segments algorithm for finding principal curves*. Technical Report IAS-UVA-00-10, Intelligent Autonomous Systems Group, University of Amsterdam, December 2000.

You may order copies of the IAS technical reports from the corresponding author or the series editor. Most of the reports can also be found on the web pages of the IAS group (see the inside front page).

**Intelligent
Autonomous
Systems**