Sequentially Fitting Mixtures Models using an Outlier Component

Michalis K. Titsias and Christopher K. I. Williams*
School of Informatics, University of Edinburgh, Edinburgh EH1 2QL, UK
c.k.i.williams@ed.ac.uk M.Titsias@sms.ed.ac.uk

Abstract

We describe a method for training mixture models by learning one component at a time and thus building the mixture model in a sequential manner. We do this by incorporating an auxiliary outlier component (a uniform density to any of the data points) into the mixture model which allows us to fit just one data cluster by "ignoring" the rest of the clusters. Once a model is fitted we remove from consideration in a probabilistic fashion all the data explained by this model and then repeat the operation. This process can be viewed as fitting a mixture model using a constrained EM algorithm. The algorithm can be used to provide a sensible initialization of the mixture components when we train a J-component mixture. We apply the algorithm to train J-component mixtures of Gaussians and multivariate multinomials and show results on real data.

1 Introduction

We address the problem of learning a mixture density model with J components

$$P(\mathbf{x}) = \sum_{j=1}^{J} \pi_j P_j(\mathbf{x}|\theta_j) \tag{1}$$

where $P_j(\mathbf{x}|\theta_j)$ is the j^{th} component having parameters θ_j and π_j the mixing coefficient. Mixture models have been widely used in statistical modelling as density estimation methods (McLachlan & Peel, 2000). Given a set of i.i.d data $X = \{\mathbf{x}^1, \ldots, \mathbf{x}^N\}$ we wish to estimate the underlying density of $\mathbf{x}$ by a mixture model of the form (1). The component densities $P_j(\mathbf{x}|\theta_j)$ can be chosen from some parametric family such as the exponential family. Most of the presentation in the rest of the paper assumes that $P_j(\mathbf{x}|\theta_j)$ can be any distribution while in out experiments we specify this to be a multivariate Gaussian in the case of continuous data and a multinomial for discrete-valued data.

*http://anc.ed.ac.uk

In this paper we describe a method for training mixture models by learning one component model at a time and thus building the mixture in a sequential manner. The key idea in doing this is that we incorporate an auxiliary outlier component (a uniform density to any possible data point) into the mixture model which allows us to fit one cluster of the data by "ignoring" the rest of the clusters. Data points that are fitted by a model are then "removed from consideration" in a probabilistic fashion so that at next stage a new model can fit to a unexplored region of the data space and so on. Intuitively the algorithm begins by considering all data as outliers and at each stage successively refines this belief by searching for clusters (structure) in all data that previously was labelled as outliers. Such a method can be useful for improving parameter initialization of EM algorithm when it is applied for training mixture models. This is because at each stage it provides a sensible way to initialize a density model to data regions that are not well explained by the already fitted models, e.g. for Gaussian mixtures this can be more effective than simultaneously initializing the centres by randomly selecting data points.

An additional feature is that the algorithm can indicate when to stop adding new components; when the outlier component fits no data (or fits only background clutter data) we have potentially reached the desirable number of components and we can stop. We show that this can be used to find the number of components in some simple clustering problems.

The structure of the remainder of the paper is as follows: In section 2 we describe the sequential algorithm for fitting a mixture model assuming any form for the component density $P_j(\mathbf{x}|\theta_j)$. In section 3 we show some experiments in real data using Gaussian and multinomial mixtures and provide also a comparison with the regular EM. For Gaussian mixtures we include also in the comparison another sequential (greedy) algorithm proposed by Vlassis and Likas (2002) and Verbeek et al. (2003). We conclude with a discussion in section 4.

2 Sequential algorithm for Mixture Models

In this section we describe the sequential algorithm for learning mixture models. Particularly, section 2.1 illustrates the idea of using an outlier component to train a single Gaussian density. Section 2.2 describes the algorithm for sequentially fitting the components of a mixture model using an outlier component, section 2.3 provides details regarding the application of the algorithm to Gaussian and multinomial mixtures and section 2.4 discusses related work.

2.1 Fitting one density model together with an outlier component

We wish to learn a density model $P(\mathbf{x}|\theta)$ together with a uniform distribution $U(\mathbf{x})$, called the outlier component, so that

$$P(\mathbf{x}) = \alpha P(\mathbf{x}|\theta) + (1-\alpha)U(\mathbf{x}). \tag{2}$$

For clarity assume at the moment that the model $P(\mathbf{x}|\theta)$ is a multivariate Gaussian with parameters $\theta = \{\mu, \Sigma\}$. Selecting first a value for α we can learn the parameters by maximizing the log likelihood $L = \sum_{n=1}^{N} \log P(\mathbf{x}^n)$ using the EM algorithm.

Notice that if $\alpha = 1$ the outlier component is neglected and the parameter estimate for the Gaussian is the maximum likelihood solution. As α decreases the Gaussian becomes more and more focused on some population of the data and as α approaches zero the Gaussian fits very few data points ending up with fitting just one data point[1].

An obvious use of this model is to robustify the Gaussian estimate by choosing a high value for α (say 0.9) which can be useful in situations where one data cluster is embedded in background clutter. However, what is less obvious is the fact that by choosing properly the value of α the robust model of equation (2) can be used for learning just one data cluster by ignoring any other clusters of the data. To illustrate this consider the data of Figure 1 which form three separate clusters. We maximize the likelihood by initializing μ selecting one data point and $\Sigma = cI$ with c equal to the maximum variance of all data dimensions. α is initialized to 0.5 and is learned infrequently by EM (every 10 iterations). Figure 1 illustrates two runs of the EM algorithm under two different parameter initializations of the mean.

If we can fit just one cluster of the data by the model described above we can then, by repeating the process, fit all the data clusters sequentially. This motivates the sequential algorithm for fitting mixture models described in next section.

2.2 Fitting mixture models sequentially

In this section we discuss how we can use an outlier component to fit a mixture model rather than a single density model. Such a mixture should have the form

$$P(\mathbf{x}) = \sum_{j=1}^{J} \pi_j P_j(\mathbf{x}|\theta_j) + (1 - \sum_{j=1}^{J} \pi_j) U(\mathbf{x}) \tag{3}$$

where generally $\sum_{j=1}^{J} \pi_j \leq 1$. We wish to train this mixture model sequentially by learning only one density model $P_j(\mathbf{x}|\theta_j)$ at each stage. An intuitive way of thinking about this is that we start by assuming that the mixing coefficients π_j, $j, \ldots, J$ are set to zero, so that the outlier component has all the probability. At j^{th} stage the mixing coefficient π_j is set free to get a positive value and the corresponding component model $P_j(\mathbf{x}|\theta_j)$ is allowed to fit some part of the data. At each stage the mixing coefficient of the outlier component always decreases which implies that the probability of what is considered as outlying data decreases sequentially.

[1] If we know the smallest distance between any two training points we can easily work out a lower bound of α that below of that value the Gaussian fits exactly one data point.

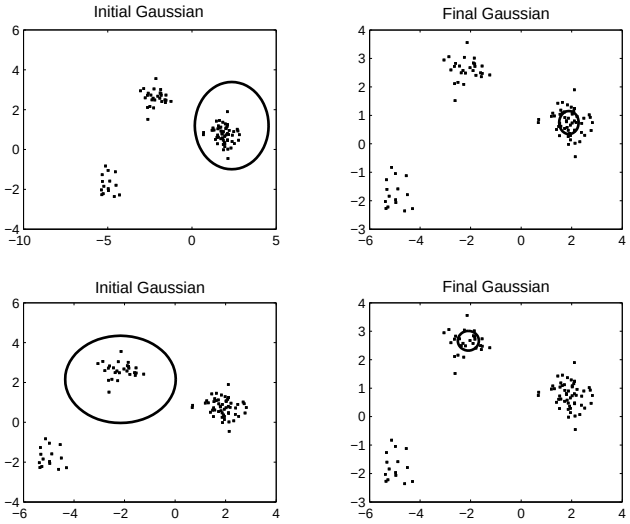

Figure 1: Illustrates fitting a Gaussian using an outlier component. Under different initializations of the mean (plots on the left) we can discover different clusters (plots on the right).

Attias (2000) has shown how mixture models can be fitted using variational Bayesian methods. It is interesting to note that if one component of the mixture at a time is updated repeatedly while keeping the other components fixed then a scheme quite similar to ours would emerge, as the as-yet-unfitted components would tend to have vague distributions not dissimilar to our uniform component[2].

We now describe our algorithm in detail, starting with training the first component $P_1(\mathbf{x}|\theta_1)$. By allowing the coefficient π_1 to obtain a positive value we have the mixture

$$P(\mathbf{x}) = \pi_1 P_1(\mathbf{x}|\theta_1) + (1 - \pi_1)U(\mathbf{x}) \tag{4}$$

which is exactly the model discussed in section 2.1 and thus learning the parameters $\{\theta_1, \pi_1\}$ can be done by maximizing the log likelihood using EM.

Suppose now that we have fitted a model $P_1(\mathbf{x}|\theta_1)$ to the data. What we wish to do next is to train the second mixture component $P_2(\mathbf{x}|\theta_2)$ by considering the mixture

$$P(\mathbf{x}) = \pi_1 P_1(\mathbf{x}|\theta_1) + \pi_2 P_2(\mathbf{x}|\theta_2) + (1 - \pi_1 - \pi_2)U(\mathbf{x}). \tag{5}$$

This case now becomes a little more complicated in the sense that we wish the second model not to fit data that are already well explained by the first

[2]We thank Steve Roberts for a helpful discussion on this point.

model. Generally the new model $P_2(\mathbf{x}|\theta_2)$ should fit a subset of the data that is reasonably different from all the data fitted by $P_1(\mathbf{x}|\theta_1)$. Such a constraint can be efficiently taken into account by applying a constrained EM algorithm where instead of the log likelihood we maximize a lower bound of the log likelihood (Neal & Hinton, 1998). Particularly, we compute the responsibilities of the uniform component for each data point $\mathbf{x}^n$ by

$$z_1^n = \frac{(1-\pi_1)U(\mathbf{x}^n)}{\pi_1 P_1(\mathbf{x}^n|\theta_1) + (1-\pi_1)U(\mathbf{x}^n)}, \tag{6}$$

which are computed by having only trained the first component and then we express a lower bound of the log likelihood of the model (5). Particularly, introducing the notation $P_2^u(\mathbf{x}) = \pi_2 P_2(\mathbf{x}|\theta_2) + (1-\pi_1-\pi_2)U(\mathbf{x})$ and using Jensen's inequality we have

$$\begin{aligned} & \sum_{n=1}^{N} \log\{\pi_1 P_1(\mathbf{x}|\theta_1) + P_2^u(\mathbf{x})\} \\ = & \sum_{n=1}^{N} \log\left\{(1-z_1^n)\frac{\pi_1 P_1(\mathbf{x}|\theta_1)}{1-z_1^n} + z_1^n \frac{P_2^u(\mathbf{x})}{z_1^n}\right\} \\ \geq & \sum_{n=1}^{N} (1-z_1^n)\log \pi_1 P_1(\mathbf{x}^n|\theta_1) + \sum_{n=1}^{N} z_1^n \log P_2^u(\mathbf{x}) \\ + & H(\{z_1^n\}) \end{aligned} \tag{7}$$

where $H(\{z_1^n\})$ denotes an entropic term independent of $\{\pi_2, \theta_2\}$. Since the parameters of the first model are fixed, maximizing the above bound simplifies to maximizing only the second term in the above sum under the constrain that π_2 should receive a value smaller or equal to $1-\pi_1$. According to the form of the above objective function maximizing with respect to $\{\theta_2, \pi_2\}$ favours solutions where the model fits data that previously was explained mainly by the outlier component. To make this more obvious note that the weights $\{z_1^n\}$ are close to zero for all data explained by $P_1(\mathbf{x}|\theta_1)$ and close to one for all data explained by the outlier component. So the objective function (7) effectively removes from consideration in a probabilistic fashion data fitted by the first model. This process of fitting the mixture components to the data can be performed sequentially. The algorithm is summarised below

1. Set $j = 0$. Initialize: $z_0^n = 1$ for all n.
2. Set $j = j + 1$. Initialize θ_j and $\pi_j = \alpha(1 - \sum_{i=1}^{j-1} \pi_i)$, where $\alpha < 1$ (we use $\alpha = 0.5$).
3. Optimize the parameters $\{\theta_j, \pi_j\}$ by running EM and maximizing:

$$\sum_{n=1}^{N} z_{j-1}^n \log\{\pi_j P_j(\mathbf{x}^n|\theta_j) + (1 - \sum_{i=1}^{j-1} \pi_i - \pi_j)U(\mathbf{x}^n)\} \tag{8}$$

   where π_j is sparsely updated by EM (every 10 iterations).

4. Update the log likelihood weights

$$z_j^n = \frac{(1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x}^n)}{\sum_{i=1}^{j} \pi_i P_i(\mathbf{x}^n|\theta_i) + (1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x}^n)}. \quad (9)$$

5. Go to step 2 or output the mixture $P_j(\mathbf{x}) = \sum_{i=1}^{j} \pi_i P_i(\mathbf{x}|\theta_i) + (1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x})$.

At each stage of the above algorithm a new model is trained (step 3) by maximizing a weighted log likelihood where the weights $\{z_j^n\}$ mask out (probabilistically) data fitted by the previously learned models. At step 4 the $\{z_j^n\}$ values are updated so that at the next stage we can fit a new model most probably to a different data subset. The sequential process can be considered as a kind of boosting algorithm for density estimation as the data points are reweighted on each iteration so that points that are well fitted by the current models become less important in later stages. We further discuss this issue in section 4.

Note that the mixing coefficient of the outlier component $1 - \sum_{i=1}^{j} \pi_i$ can only decrease at each stage and naturally the learning process stops once this coefficient becomes very close to zero. In section 3 we describe a simple stopping criterion based on this idea and we use it to find the number of clusters in some simple clustering problems.

The outcome of the sequential algorithm can be used to initialize a mixture model. In such case the outlier component is discarded and the coefficients π_j, $j = 1, \ldots, J$ are normalized to sum to one. The parameters of the resulting mixture can be refined by applying EM and maximizing the likelihood.

2.2.1 Refinement of the previous learned components

So far the models fitted to the data remain fixed for the later stages, however it might be more effective if we can refine their parameters during learning. Next we discuss the refinement procedure we use in our implementation.

The refinement can be introduced as an additional step of the sequential algorithm between steps 4 and 5 which is applied only for $j \geq 2$. After step 4 the values $\{(1 - z_j^n)\}$ are the responsibilities according to which all the components $P(\mathbf{x}|\theta_i)$, $i = 1, \ldots, j$ explain the data. Based on these responsibilities we maximize a lower bound of the log likelihood

$$\begin{aligned} F &= \sum_{n=1}^{N} (1 - z_j^n) \log \sum_{i=1}^{j} \pi_i P(\mathbf{x}^n|\theta_i) \\ &+ \sum_{n=1}^{N} z_j^n \log(1 - \sum_{i=1}^{j} \pi_i) U(\mathbf{x}^n) + H(\{z_j^n\}) \\ &= \sum_{n=1}^{N} (1 - z_j^n) \log \sum_{i=1}^{j} \pi_i P(\mathbf{x}^n|\theta_i) + const \end{aligned} \quad (10)$$

where the sum $\sum_{i=1}^{j} \pi_i$ is fixed at the value that obtains before we apply the refinement step. Note that the second term in the first line is a constant since $\sum_{i=1}^{j} \pi_i$ is invariant as well as the entropic term since depends on the $\{z_j^n\}$ values. Thus the refinement objective function is just the first term in the above sum under the constraint that $\sum_{i=1}^{j} \pi_i$ is invariant, which can be carried out using EM. Such refinement will be within the data regions that the first j components already fit. Completely unexplored data regions for which the outlier component obtains high responsibility will remain unexplored after refinement.

Assuming that we have carried out the refinement step as described above, we need to feed the changes made back into the sequential algorithm. This is simply done by updating the weights, so the refinement step is completed by updating $\{z_j^n\}$ according to the step 4 of the sequential algorithm.

It is interesting to compare the weighted log likelihood used in the refinement step (eq.(10)) with the corresponding used for learning a new Gaussian (eq.(8)). The quantity F in eq.(10) explicitly masks out (in a probabilistic way) all the data fitted by the outlier component. This allows refinement of the Gaussians without big moves in unexplored data regions. On the other hand the weighted log likelihood used to learn a new Gaussian works in the opposite way; it masks out all the data fitted by the already learned Gaussians in order to fit a new Gaussian to unexplored data regions.

2.2.2 Running time analysis

For mixture models trained by EM algorithm the time complexity depends on the number of evaluations of the component densities $P_j(\mathbf{x})$. The sequential algorithm without refinement fits one component at each stage which has complexity $O(N)$ per EM iteration. If the maximum number of iterations is M_1, the whole process for fitting J models needs $O(JNM_1)$ operations. This can be further reduced by noting that at each stage the data fitted by the current models become much less important since their weights z_{j-1}^n will take a very close to zero value. We can explicitly remove these data from the dataset by expressing the set $A_j = \{n : z_{j-1}^n > \delta\}$ with δ small (e.g. $\delta = 0.05$) and use only the data from A_j when we train the j^{th} component. Thus the j^{th} stage needs $O(N_j M_1)$ time ($N_j = |A_j|$) and the total time complexity becomes $O(JKM_1)$ where K denotes the average value of all N_js. In practice K can be much smaller than N. The algorithm with refinement steps at the j^{th} stage learns first one component and updates the current mixture (only for $j \geq 2$) which totally requires $O(JNM_1 + \sum_{i=2}^{J} iNM_2) = O(JNM_1 + J^2NM_2)$ operations where M_2 is the maximum number of EM iterations for the refinement procedure. This time can also be reduced similarly to the no-refinement case.

Note that when we train a J-component mixture the sequential algorithm is used as the initialization procedure of the mixture model. This means is not desirable to run all the EM algorithms required by the sequential algorithm till convergence. Typically we can apply few iterations M_1 and M_2 for training a new Gaussian and refining the current mixture, respectively. Especially it is desirable that M_2 be small (e.g less than 20) so that the refinement procedure

to be fast. Once the components have been initialized the regular EM is applied to refine this mixture which needs $O(JN)$ operations per EM iteration.

2.3 Parameter initialization and specifying $U(\mathbf{x})$

When we apply the sequential algorithm we have to specify several parameters. First of all the α value used in the step 2 of the algorithm is set equal to 0.5. Now, when $P_j(\mathbf{x}|\theta_j)$ is a Gaussian we initialize the mean to some training point selected from the distribution $G(n) = z_j^n / \sum_{n'=1}^{N} z_j^{n'}$. $G(n)$ at each stage favours points that are mainly explained by the outlier component. The covariance matrix is initialized to be spherical with variance equal to the maximum variance of all data dimensions.

In the case of multinomial mixtures the component densities have the form

$$P_j(\mathbf{x}|\theta_j) = \prod_{i=1}^{d} P(x_i|\theta_j^i) \tag{11}$$

where d is the number of dimensions and $P(x_i|\theta_j^i)$ are multinomial distributions. To initialize θ_j we first select a training point $\mathbf{x}^n$ from $G(n)$ and initialize the multinomial parameters for dimension i by giving a fraction γ of the probability mass to the value that corresponds to the value of x_i^n and split the rest of probability mass uniformly over the rest of the values.

One crucial point is how we define the uniform distribution $U(\mathbf{x})$. An obvious way is to find the hypercube/sphere that contains all the training data and express the uniform density in that space. However in high dimensional spaces the data often lies on lower dimensional manifolds and such a "bounding box" uniform distribution will tend to give very low probability densities to the data points in comparison with the other components. To overcome this we set $U(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} P(\mathbf{x}^n|\theta_{ML})$ where $P(\mathbf{x}|\theta_{ML})$ is a single component model (e.g. a Gaussian) using maximum likelihood parameters θ_{ML}. This can be seen as a rescaling of the α parameter in equation 2.

2.4 Related work

In addition to the standard EM algorithm, various initialization strategies have also been proposed. Figueiredo and Jain (2002) and others have demonstrated a backwards selection method, starting with many components and pruning some away using a prior that favours sparsity. The forward sequential (greedy) algorithm for Gaussian mixtures proposed by Vlassis and Likas (2002) and Verbeek et al. (2003) initializes the Gaussians one after the other by comparing at each stage a set of candidate initializations. Once a Gaussian has been initialized the current mixture model is refined by maximizing the likelihood using EM. One important difference with our method is that we use the outlier component which allows the Gaussians at each stage to fit some part of the data, while in (Vlassis & Likas, 2002; Verbeek et al., 2003) the Gaussians at each stage fit all

the data. Note also that our method does not use a set of candidate initializations when we fit a new component and for this reason it requires less running time.

3 Experiments

In this section we demonstrate the sequential algorithm for training Gaussian and multinomial mixtures. In section 3.1 we apply the algorithm to learn a J-component mixture on two real datasets and provide a comparison with the regular EM as well as the sequential algorithm of Verbeek et al. (2003). In section 3.2 we apply the sequential algorithm to find the number of components in a case with well-separated clusters.

3.1 Training a J-component mixture model

We present two experiments in real high dimensional data using Gaussian and multinomial mixtures respectively. In the first experiment we use the Brodatz textures images following an experimental setup used in Verbeek et al. (2003). The task is to cluster a set of $16 \times 16 = 256$ patches taken from 256×256 pixel Brodatz texture images. We consider the number of clusters (textures) from which patches are extracted to be $J = \{3, 5, 7, 9\}$. For a specific J we randomly choose J textures from the 37 available textures, create a set of $900J$ patches and then keep the half ($450J$) for training and the rest for testing. We repeat this experiment 50 times. Each data set was also projected from the 256 to 50 dimensions using PCA in order to speed up the experiment. For each of the 50 datasets of a certain J we train a mixture model with J components using (i) k-means initialized EM[3] ($k$means), (ii) the sequential (greedy) algorithm of Verbeek et al. (2003) (VVK)[4], (iii) the sequential algorithm with refinement (Ref) and with (iv) no refinement (NoRef) steps. For the Ref and NoRef methods the M_1 and M_2 numbers defined in section 2.2.2 are set to 50 and 20 respectively. Table 1 and 2 display the t-statistic values of the difference of the average log likelihoods for training and test data set respectively. Note that when we consider the differences in log likelihoods of the method A and B ($A - B$ in the notation in the Tables 1 and 2) and the t-statistic is larger than 2.01 the method A is significantly better than B at level 5% ($t_{0.025,49} = 2.01$). When the t-statistic is less than -2.01 the method A is significantly worse. Observe that the sequential fitting algorithm with refinement beats EM initiated with k-means and that these differences are significant for $J = 7,\ 9$. Also the method with refinements is better than the VVK method and this is significant for $J = 7, 9$. Note that using refinement always improves the results compared to the algorithm with no refinement steps. We have also run the algorithm for

[3] We used the NETLAB implementation available from `http://www.ncrg.aston.ac.uk/netlab`.

[4] The code is provided by the authors at http://carol.science.uva.nl/~vlassis/research/learning/index_en.html.

Table 1: t-statistic values for the differences of the average training set log likelihoods for the Brodatz textures. Bold face indicates that the test is significant at the 5% level.

	3	5	7	9
NoRef - $k$means	-1.03	-0.27	1.7	1.74
Ref - $k$means	0.74	1.74	**3.26**	**3.5**
VVK - $k$means	1.05	-0.26	1.27	0.48
Ref - NoRef	**2.34**	1.24	**2.63**	**3.48**
Ref - VVK	0.12	1.38	**2.49**	**2.49**
NoRef - VVK	-1.46	-0.07	0.83	1.04

Table 2: t-statistic values for the differences of the average test set log likelihoods for the Brodatz textures. Bold face indicates that the test is significant at the 5% level.

	3	5	7	9
NoRef - $k$means	-1.02	-0.23	1.66	1.52
Ref - $k$means	0.69	1.75	**3.17**	**3.25**
VVK - $k$means	1.01	-0.24	1.03	0.18
Ref - NoRef	**2.34**	1.3	**2.59**	**3.8**
Ref - VVK	0.11	1.41	**2.57**	**2.53**
NoRef - VVK	-1.43	-0.05	0.93	1.09

mixture fitting proposed by Figueiredo and Jain (2002) on this data using their code (available from `http://www.lx.it.pt/~mtf/mixturecode.zip`). However, the pruning strategy they use means that one cannot guarantee to get J components in the final model, and when fewer than J components are selected the log likelihoods are low leading to poor performance in comparison to the methods reported in the Tables.

In our experiments the regular EM with few steps of the k-means algorithm for initialization was the fastest followed by the sequential algorithm with no-refinement and the algorithm with refinement, while the method of Verbeek et al. (2003) was the slowest. For example, choosing $J = 7$ the real running time of the algorithms (averaged over 10 runs) was: 25 seconds for the $k$means method, ii) 30 seconds for the NoRef method, iii) 65 seconds for the Ref method and iv) 102 seconds for the VVK method, respectively.

The second experiment uses handwritten digits data as employed in Frey et al. (1996). The digits are quantized to 8×8 binary images. Following Meila and Heckerman (2001) we use only the digit 6 dataset (but note that different preprocessing means that our results are not directly comparable to theirs). This dataset consists of 700 training cases and 200 test cases. The implementation of the regular EM (Reg-EM) is based on initializing the multinomial parameters by picking J data points randomly and applying the parameter initialization

Table 3: Mean average log likelihoods for the test data in the digit6 dataset.

J	REF	NOREF	REG-EM
3	-28.1 ± 0.21	-27.97 ± 0.2	-28.06 ± 0.21
5	-26.8 ± 0.03	-26.79 ± 0.03	-26.8 ± 0.03
7	-26.23 ± 0.09	-26.25 ± 0.13	-26.28 ± 0.14
9	-25.93 ± 0.14	-25.90 ± 0.12	-25.95 ± 0.14
11	-25.72 ± 0.11	-25.68 ± 0.09	-25.67 ± 0.12

method described in section 2.2 (γ was chosen to be 0.75). The mean average log likelihoods over 20 random initializations and for different choices of the number of components J are displayed in Table 3. On this data there is no significant difference between the performance of the algorithms.

3.2 Finding the number of components

The sequential algorithm can indicate when to stop adding components, since in case the outlier component fits very few data we probably have reached the required number of components. We apply this to find the number of components in cases of separate clusters.

The criterion we use is very simple. Assume that we have trained the j^{th} Gaussian, we express the set $S_j = \{\mathbf{x}^n : z_j^n > 0.5\}$ which indicates the data points for which the outlier component obtains the largest responsibility. Now if S_j is a large set, we know that there are some data regions that are not explored by the Gaussian models, so we have to continue learning. However when this set contains very few data we can conclude that the Gaussians have discovered all the data regions. In our experiments we require $|S_j| > d$, where d is the data dimensionality, otherwise we stop learning. For the data of Figure 2 generated from a five-component mixture we applied the sequential algorithm with refinement steps. The stopping criterion is met when we reach five components, when, in fact $|S_5| = 0$.

Note that the above criterion can find the number of components in problems with well-separated clusters. In cases the data form highly overlapping clusters or where there is no clear separation of the data into clusters the above criterion will only roughly indicate how many components needed to represent the data.

4 Discussion

Above we have described our sequential approach for fitting mixture models, and have demonstrated that for training a J-component mixture model it can produce better performance than rival algorithms. Compared to the sequential method of Verbeek et al. (2003) our method gave better performance and it is also fastest since we do not use a set of candidate initializations when we

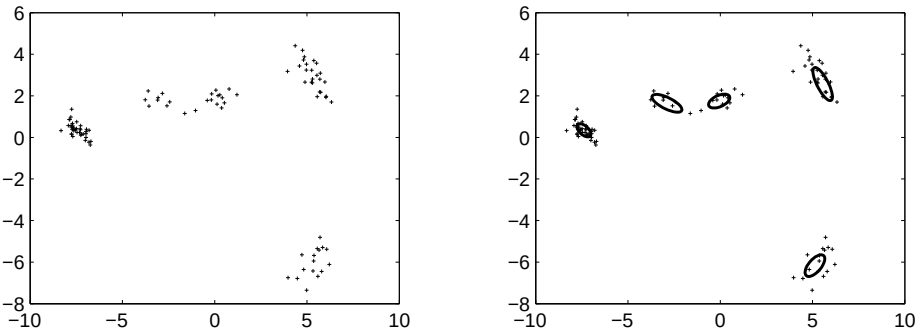

Figure 2: The plot on the left illustrates a data set generated from a 5-component Gaussian mixture. The plot on the right displays the solution found by the sequential algorithm once the stopping criterion is met.

fit a new component. Note that our method could probably be significantly improved by having a small set of candidate initializations.

In terms of model selection, Bayesian methods using the marginal likelihood as a selection criterion or approximations such as BIC penalties are most common. While our method is unlikely to be able to compete with sophisticated Bayesian methods such as reversible jump MCMC (Green, 1995) on densities whose components are not well separated, it does provide a much more rapid answer.

The sequential algorithm can be regarded as a boosting density estimation algorithm. There has been some recent work (Thollard et al., 2002; Rosset & Segal, 2003) on extending boosting from the supervised learning problem to the density estimation problem. Our sequential formulation of fitting process is reminiscent of these boosting algorithms, however one attractive feature of our scheme is that the boosting view derives from a constrained EM formulation of the problem which derives the weightings in a particular way. After we had developed our idea we learned of the work of Neal and Mackay (1998) who have shown that a sequential fitting approach to the mixture of experts architecture gave a boosting-like algorithm for supervised learning.

References

Attias, H. (2000). A variational Bayesian framework for graphical models. *Advances in Neural Information Processing Systems 12*. MIT Press.

Figueiredo, M. A. T., & Jain, A. K. (2002). Unsupervised learning of finite mixture models. *PAMI, 24(3)*, 381–396.

Frey, B., Hinton, G., & Dayan, P. (1996). Does the wake-sleep algorithm produce

good density estimators. *Advances in Neural Information Processing Systems 8*. MIT Press.

Green, P. J. (1995). Reversible Jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika, 82(4)*, 711–732.

McLachlan, B. G., & Peel, D. (2000). *Finite mixture models*. Wiley, New York.

Meila, M., & Heckerman, D. (2001). An experimental comparison of model-based clustering methods. *Machine Learning, 42*, 9–29.

Neal, R., & Hinton, G. (1998). A view of the EM algorithm that justifies incremental, sparse and other variants. In M. Jordan (Ed.), *Learning in Graphical Models*, 355–368. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Neal, R., & Mackay, D. (1998). Likelihood–based boosting. Unpublished paper available at `http://www.inference.phy.cam.ac.uk/mackay/BayesICA.html`.

Rosset, S., & Segal, E. (2003). Boosting density estimation. *Advances in Neural Information Processing Systems 15*. MIT Press.

Thollard, F., Sebban, M., & Ezequel, P. (2002). Boosting density function estimators. *13th European Conference on Machine Learning* (pp. 431–443).

Verbeek, J., Vlassis, N., & Krose, B. (2003). Efficient greedy learning of Gaussian mixture models. *Neural Computation, 15*, 469–485.

Vlassis, N., & Likas, A. (2002). A greedy EM for Gaussian mixture learning. *Neural Processing Letters, 15*, 77–87.