Large Scale Gaussian Mixture Modelling using a Greedy Expectation-Maximisation Algorithm

J.R.J. Nunnink

May 19, 2003

Contents

List of Figures

Chapter 1

Introduction

An important task in data analysis is to search and identify clusters of data items. These clusters can be used for a number of purposes, such as classification of new data or for predicting missing data. Data clustering methods can also be used as learning algorithms when applied to partially labelled data, where each found cluster corresponds to a class.

The task of clustering algorithms is to fit a model to the data. A useful and popular class of models are Mixture models which are convex combinations of basic model components [8]. Typically these components are Gaussian density functions, because often data is locally Gaussian distributed, in which case the data set can be assumed to have been generated by a hypothetical Gaussian mixture. The task is then to find the parameter of that generating mixture.

Because of their probabilistic nature, Gaussian mixtures are in principle preferred over models that cut a data set in discrete parts. In most AI applications where a new data item needs to be classified, it is more desirable to calculate the probability that this item belongs to certain clusters than to assign it to only one cluster.

The most popular algorithm for training a Gaussian mixture is the Expectation-Maximisation (EM) algorithm [5]. Among its advantages are its easy implementation, no need to set extra user-defined parameters, and guaranteed monotone increase in model quality. There are also some downsides, however, the most important of which are its high dependency on initialisation and computational complexity, which is linear with respect to the size of the data set.

Over the past years several improvements of the EM algorithm were proposed to counteract those problems. One of them is a generalisation of the EM algorithm, based on the variational free energy in statistical physics, that justifies several useful variants [11]. Also, some improvements aimed at lowering the computational cost by working with groups of data items instead of the items themselves [9] (In [1, 6, 10] this method is applied to the k-means algorithm). Finally, a greedy version of EM was proposed, which deals with the initialisation problems of EM, thus resulting in higher quality models [13, 14]. This greedy method, however, is computationally more expensive.

In this thesis I will describe an algorithm that combines these improvements, thus being able to handle large data sets, by keeping the computational cost low, while producing high quality models with every run of the algorithm.

In Chapter 2, I will explain Gaussian mixture models and the various EM algorithms. Then, in Chapter 3, I will describe a new algorithm that combines the strengths of the various EM variants. The experimental results obtained by running the new algorithm are displayed in Chapter 4. Finally in Chapter 5 are the conclusions about the performance of the new algorithm.

Chapter 2

Training Gaussian Mixtures

2.1 Gaussian Mixtures

2.1.1 Gaussian Mixture Model

Finite mixture models are models that consist of a weighted sum of basic model components. Thus a mixture model f_k with k components is defined as:

$$f_k(\boldsymbol{x}) = \sum_{s=1}^{k} \pi_s \phi_s(\boldsymbol{x}; \theta_s), \tag{2.1}$$

where ϕ_s is the s-th component parameterised by θ_s, and π_s is the mixing weight of the s-th component. The mixing weights must satisfy $\sum_{s=1}^{k} \pi_s = 1$ and $\pi_s \geq 0$.

Gaussian mixture models are those mixture models where the basic components are multivariate Gaussian density functions, defined by:

$$p(\boldsymbol{x}; \theta) = (2\pi)^{-d/2} |\boldsymbol{C}|^{-1/2} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{m})^\top \boldsymbol{C}^{-1} (\boldsymbol{x} - \boldsymbol{m})\right], \tag{2.2}$$

where $\boldsymbol{x} \in \mathrm{I\!R}^{\mathrm{d}}$, and θ represents the parameters $\boldsymbol{m}$ and $\boldsymbol{C}$, which are respectively the mean and the covariance matrix of the density function and where the $|\cdot|$ operator denotes the determinant of a matrix.

The Gaussian density function is especially useful for modelling clustered data that is normally distributed. Note that for example sensory noise is typically Gaussian distributed. Therefore such data sets can be assumed to have been generated by a Gaussian mixture, whose parameters we want to find. When generating a data set corresponding to a model, each component generates a cluster of points that is positioned and shaped according to the parameters of that component, and the relative number of points belonging to each cluster would depend on its mixing weight.

2.1.2 Model Quality

So assuming that a data set is generated by a certain Gaussian mixture, the task is to fit a model to this data, in other words to estimate the parameters of the generating mixture. For this it is necessary to have some notion of the quality of the model with respect to the data set. A standard way to do this is to look at the likelihood that the data set was actually generated by the model itself, in other words to what extent the distribution of the data corresponds to the model.

We define the log-likelihood $\mathcal{L}$, given the parameters θ and the data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, as:

$$\begin{aligned}
\mathcal{L}(\theta) &= \log p(\boldsymbol{X}) \\
&= \log \prod_{\boldsymbol{x} \in \boldsymbol{X}} \sum_{s=1}^{k} p(\boldsymbol{x}, s) \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{X}} \log \sum_{s=1}^{k} p(\boldsymbol{x}, s), \\
&= \sum_{\boldsymbol{x} \in \boldsymbol{X}} \log \sum_{s=1}^{k} p(\boldsymbol{x}|s) p(s), \qquad (2.3)
\end{aligned}$$

where $p(s)$ is the prior distribution over the components, also known as the mixing weights. Note that independence of the data points is assumed and therefore the likelihood of the data set is equal to the product of the likelihoods of the data points.

Fitting the model to a data set is now the same as maximising (2.3) with respect to the parameters θ and the mixing weights π_s. However it is a complex function so this maximisation cannot be done easily.

2.2 The Expectation-Maximisation Algorithm

2.2.1 Log-likelihood Optimisation

Optimisation of the log-likelihood could be done by a gradient descend method, so by calculating a linear or quadratic approximation of the log-likelihood function (2.3) around the point of the current parameter values, and then changing the parameters to values that increase the log-likelihood according to the approximated gradient, i.e. by taking a step up on the log-likelihood surface.

A problem for that method, however, is that the gradient gives little information about the surface around the current point and therefore one does not know how large the step should be, because taking too large steps might make you miss optimal states while taking too small steps could make convergence very slow. Another problem is that one doesn't have a notion about the reliability of the approximation, so one could be taking steps in the wrong direction and never find out.

Another method is to choose a lower bound on the log-likelihood that is easier to optimise and use that lower bound to find the optimal parameters. The Expectation-Maximisation (EM) algorithm by Dempster et al. [5] is such a method that iteratively maximises a lower bound with respect to the parameters and a probability distribution over the components.

2.2.2 Free Energy

I will now first describe a variational version of the EM algorithm as proposed by Neal and Hinton [11] and then show that the well-known regular EM algorithm is a special case of it.

Neal and Hinton propose a lower bound $\mathcal{F}$ on the log-likelihood that uses a distribution Q, which gives for every data point $\boldsymbol{x}$ a distribution $q_{\boldsymbol{x}}$ over the components s. They are also called the *responsibilities* of the components for the data points. $\mathcal{F}$ can be derived in the following way, starting with Bayes' Rule. All probabilities are assumed given the current parameters:

$$\begin{aligned}\log p(\boldsymbol{x}, s) &= \log p(s|\boldsymbol{x}) + \log p(\boldsymbol{x}) \\ \log p(\boldsymbol{x}) &= \log p(\boldsymbol{x}, s) - \log p(s|\boldsymbol{x}) + \log q_{\boldsymbol{x}}(s) - \log q_{\boldsymbol{x}}(s). \qquad (2.4)\end{aligned}$$

$E_{q_{\boldsymbol{x}}}(\cdot)$ will denote the expectation with respect to the distribution $q_{\boldsymbol{x}}$ over the components. Note that $E_{q_{\boldsymbol{x}}} \log p(\boldsymbol{x}) = \log p(\boldsymbol{x})$ because $p(\boldsymbol{x})$ is independent on $q_{\boldsymbol{x}}$. Furthermore $\mathcal{H}$ is the *entropy* of a distribution, defined by:

$$\mathcal{H}(q_{\boldsymbol{x}}) = -\sum_s q_{\boldsymbol{x}}(s) \log q_{\boldsymbol{x}}(s) = -E_{q_{\boldsymbol{x}}} \log q_{\boldsymbol{x}}(s),$$

and D_{KL} is the *Kullback-Leibler divergence*, which is a measure of the distance between two distributions and is defined by:

$$D_{KL}(q_{\boldsymbol{x}} \parallel p(s|\boldsymbol{x})) = \sum_s q_{\boldsymbol{x}}(s) \log \frac{q_{\boldsymbol{x}}(s)}{p(s|\boldsymbol{x})} = E_{q_{\boldsymbol{x}}} \log \frac{q_{\boldsymbol{x}}(s)}{p(s|\boldsymbol{x})}. \qquad (2.5)$$

Using these three definitions we can now rewrite (2.4) to:

$$\begin{aligned}\log p(\boldsymbol{x}) &= E_{q_{\boldsymbol{x}}} \log p(\boldsymbol{x}, s) - E_{q_{\boldsymbol{x}}} \log p(s|\boldsymbol{x}) + E_{q_{\boldsymbol{x}}} \log q_{\boldsymbol{x}}(s) - E_{q_{\boldsymbol{x}}} \log q_{\boldsymbol{x}}(s). \\ &= E_{q_{\boldsymbol{x}}} \log p(\boldsymbol{x}, s) + D_{KL}(q_{\boldsymbol{x}} \parallel p(s|\boldsymbol{x})) + \mathcal{H}(q_{\boldsymbol{x}}). \qquad (2.6)\end{aligned}$$

And thus we get the two equivalent definitions of $\mathcal{F}_{\boldsymbol{x}}$:

$$\begin{aligned}\mathcal{F}_{\boldsymbol{x}}(Q, \theta) &= \log p(\boldsymbol{x}; \theta) - D_{KL}(q_{\boldsymbol{x}} \parallel p(s|\boldsymbol{x})) \qquad (2.7) \\ &= E_{q_{\boldsymbol{x}}} \log p(\boldsymbol{x}, s; \theta) + \mathcal{H}(q_{\boldsymbol{x}}). \qquad (2.8)\end{aligned}$$

From now on I will refer to $\mathcal{F}$ as the *free energy* of a mixture, after the (negative) variational free energy used in statistical physics that is analogous to $\mathcal{F}$. Note that $\mathcal{F}_{\boldsymbol{x}}$ is the free energy of the mixture for one data point $\boldsymbol{x}$. To get that of the whole data set one can simply sum over all points, $\mathcal{F} = \sum_{\boldsymbol{x}} \mathcal{F}_{\boldsymbol{x}}$.

The first composition (2.7) of this definition is in fact equal to the log-likelihood minus the Kullback-Leibler divergence, and since this divergence is always non-negative it is thus immediately clear that $\mathcal{F}$ is in fact a lower bound on the log-likelihood.

2.2.3 Lower Bound Maximisation

The EM algorithm consists of two parts, an E-step that maximises $\mathcal{F}$ with respect to the responsibilities Q given the current parameters, and an M-step that maximises $\mathcal{F}$ with respect to the model parameters θ, given the responsibilities found in the E-step. Continually repeating these two steps will eventually converge the model parameters to a (local) maximum of the free energy.

For the E-step we will use the first decomposition (2.7) of $\mathcal{F}$. Since only the divergence term of that equation depends on $q_{\boldsymbol{x}}$ and since that term has negative influence on $\mathcal{F}$, we only need to minimise the Kullback-Leibler divergence between the responsibilities and the posterior distribution over the components $p(s|\boldsymbol{x})$. The divergence is always non-negative and its minimum value is 0 and this occurs when the distributions between which the divergence is calculated, $q_{\boldsymbol{x}}$ and $p(s|\boldsymbol{x})$, are exactly equal c.f. (2.5). Thus in the E-step we set $q_{\boldsymbol{x}}$ equal to the posterior $p(s|\boldsymbol{x})$, which can be calculated with:

$$p(s|\boldsymbol{x}) = \frac{\pi_s p(\boldsymbol{x};\theta_s)}{f_k(\boldsymbol{x})}, \tag{2.9}$$

where $f_k(\boldsymbol{x})$ is the likelihood of $\boldsymbol{x}$ under the entire mixture.

The M-step uses the second decomposition (2.8) of the free energy. In this decomposition the entropy term does not depend on θ, so maximising $\mathcal{F}$ with respect to the parameters means maximising the expectation, with respect to the Q found in the E-step, of the joint log-likelihood. This maximisation can be carried out by calculating the first order derivative and setting it to zero. This gives the well-known update equations:

$$\pi'_s = \frac{1}{n}\sum_{i=1}^{n} q_i(s), \tag{2.10}$$

$$\boldsymbol{m}'_s = \frac{1}{n\pi'_s}\sum_{i=1}^{n} q_i(s)\boldsymbol{x}_i, \tag{2.11}$$

$$\boldsymbol{C}'_s = \frac{1}{n\pi'_s}\sum_{i=1}^{n} q_i(s)\boldsymbol{x}_i\boldsymbol{x}_i^\top - \boldsymbol{m}'_s\boldsymbol{m}'^\top_s. \tag{2.12}$$

An important feature of the free energy is that the E and M-steps do not necessarily have to optimise it in every step. In fact, any responsibility and parameter update step that increases $\mathcal{F}$ will let it move towards a local maximum, which is the task of the algorithm. This means that the responsibilities do not have to be set equal to the posterior distribution in the E-step and that not all model components s need to have their parameters updated in every M-step.

Neal and Hinton [11] prove that any local maximum $\mathcal{F}(Q^*, \theta^*)$ of the free energy is also a local maximum $\mathcal{L}(\theta^*)$ of the log-likelihood and therefore when this algorithm converges it has in fact also found a (local) maximum of the log-likelihood. It is easy to see that for any given θ there is a Q_θ, equal to the posterior distribution, for which $\mathcal{F}(Q_\theta, \theta) = \mathcal{L}(\theta)$. In particular for any θ^* that maximises the log-likelihood there is a Q^* such that $\mathcal{F}(Q^*, \theta^*) = \mathcal{F}(Q_{\theta^*}, \theta^*) = \mathcal{L}(\theta^*)$. If there is a local maximum of the log-likelihood at θ^* then that means there is no $\theta^\dagger$ nearby for which $\mathcal{L}(\theta^\dagger) > \mathcal{L}(\theta^*)$ and since Q_θ changes continuously with θ this means that there is no $\theta^\dagger$ and $Q_{\theta^\dagger}$ nearby for which $\mathcal{F}(Q^\dagger, \theta^\dagger) > \mathcal{F}(Q^*, \theta^*)$, in other words the free energy then also has a local maximum at θ^* and Q^*.

2.2.4 Time Complexity

The main computation that has to be done by the algorithm is calculating the likelihood $p(\boldsymbol{x}|s; \theta_s)$ of a data point for a component. Each E-step requires that calculation for each of the n data points and each of the k components in order to compute all the responsibilities. Summing the responsibilities over all data points in the M-step has to be done for each of the k components. So the number of computations per iteration of EM is $O(nk)$. Note that I disregard the dependence on the number of dimensions, since it is not relevant to the discussion in this thesis.

2.2.5 EM Variants

Several variants of the free energy maximisation algorithm are possible based on the extent to which the responsibilities are updated. The *regular* EM algorithm as proposed by Dempster et al. [5] can now be seen as special case or variant. In regular EM the E-step calculates the posterior distribution over the components and uses this to update the model components in the M-step. This is the same as setting the responsibilities always equal to $p(s|\boldsymbol{x})$ in the E-step, which means the divergence term is always made 0 and $\mathcal{F}(Q, \theta)$ is always equal to $\mathcal{L}(\theta)$ after the E-step. The lower bound $\mathcal{F}$ always increases, unless it has already converged.

Another possible variant would be *incremental* EM, which doesn't update the complete responsibilities in the E-step, by calculating them for only part of all data points. It leaves the responsibilities for the other data points unchanged or even sets them to some predetermined value, a uniform distribution for example. This is justified as long as it leads to an increase of the total free energy. Note that the total free energy is the sum of the free energy of each data point, and that $\mathcal{F}_{\boldsymbol{x}}$ will stay unchanged for those points that haven't got their responsibilities updated. $\mathcal{F}_{\boldsymbol{x}}$ will increase for the data points that do have updated responsibilities, since those responsibilities are closer to the posterior distributions, resulting in a smaller divergence term in the first decomposition (2.7) of $\mathcal{F}_{\boldsymbol{x}}$. Thus the total free energy will never decrease. I will discuss an incremental variant in more detail in Section 2.3.2.

Also possible is *sparse* EM, which does a limited E-step by updating the responsibilities for each data point of only a certain number of the components and leaving the rest relatively unchanged, making sure the responsibilities still sum to 1 for each data point. Since the responsibilities $q_{\boldsymbol{x}}$ for each data point are optimised for the updated components, the overall responsibilities will not get worse, so $\mathcal{F}_{\boldsymbol{x}}$ will never decrease.

2.3 Greedy EM

One of the main problems of the EM algorithm is the simple fact that one needs to initialise the algorithm with a Gaussian mixture. Accordingly, the resulting mixture after convergence, and thus the log-likelihood, depends heavily on the choice of the initial mixture. Most often the algorithm gets stuck in the first local maximum it reaches. The usual solution for this is to run the algorithm several times using different starting mixtures. The hope is that the best local maximum will yield a log-likelihood close to the global maximum within several tries.

Another problem is that one often doesn't know how many clusters the data set contains. The number of components in the initial mixture will then be very difficult to choose.

Recently Vlassis and Likas [14] proposed a *greedy* approach, which aims at avoiding these two issues. Based on theoretical work by Li and Barron [7] they use an algorithm that builds the Gaussian mixture model componentwise starting with a 1-component mixture. After having converged a mixture f_k they add a new component and then use the EM algorithm to converge to an optimal f_{k+1}, and so on. Fig. 2.1 shows an example of the construction of a Gaussian mixture using greedy EM.

This does require more work because you have to do EM until convergence for every intermediate mixture. But this gets compensated by the fact that this algorithm is much better at finding a (near) global maximum of the log-likelihood, and therefore the algorithm has to be run only once. Also is initialisation much easier since the optimal parameters of a 1-component mixture can be found trivially. Finally there is no need to guess the number of clusters, and thus the number of components. The algorithm can be ter-

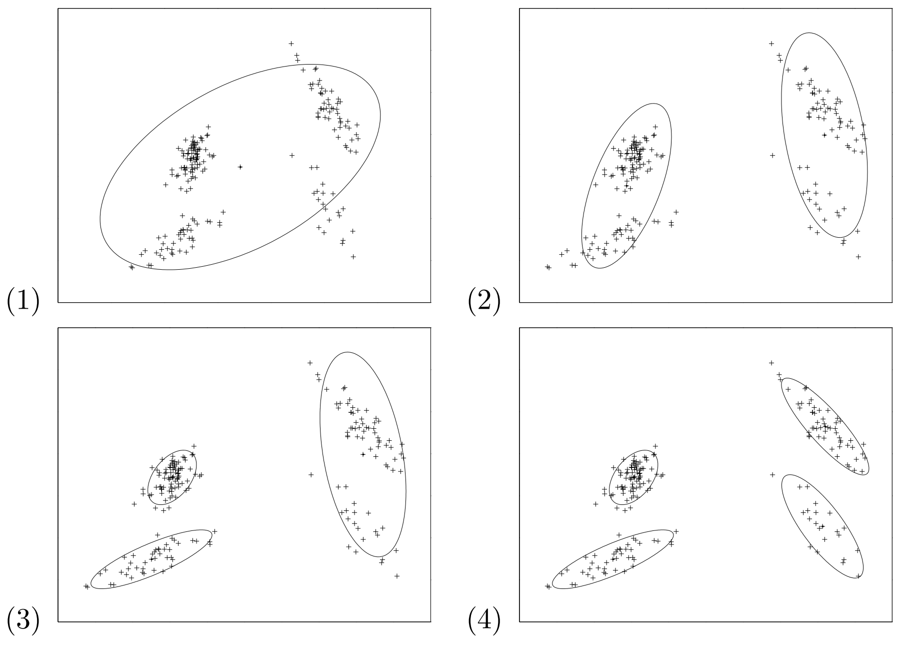


Figure 2.1: Example of the construction of a 4-component Gaussian mixture. Each ellipse corresponds with one component.

minated whenever optimally adding a new component no longer leads to an increase in free energy. Looking at the sequence of mixtures can even give us a notion of optimal number of components, by using for example cross validation.

2.3.1 Component Insertion

The important issue in greedy EM is the search for and insertion of the optimal new component. Vlassis and Likas did this by considering every data point as a candidate for insertion, giving them standard parameters and looking at the resulting log-likelihood if they were to be inserted.

There are some points of criticism on this method of searching, however. Firstly, the number of computations used is quadratic in the number of data points, since you have to compute the log-likelihood of every data point for every candidate. When working with large data sets this is unacceptable. Secondly, the choice for the candidates' parameters is not guaranteed to be good and might lead to bad modelling of certain areas of the data space.

Verbeek et al. [13] come with an improvement of the insertion procedure. They propose to look at the current mixture f_k and generate only a certain number of m candidates per component. This is done by randomly splitting the data points that 'belong' to a component, i.e. those points for which that component has highest responsibility, in two groups. Calculating the mean and covariance of these groups give the parameters for two candidates. This splitting of the 'parent' component is repeated until m candidates are generated. Then for each of these km candidates, instead of immediately comparing them, they do a number of sparse incremental EM steps first and then compare the resulting log-likelihoods. This avoids the dependency on initial candidate parameters, since they are changed to optimal values anyway. It also uses less computations than [14], because the number of candidates depends on the number of components and not on the number of data points. I will discuss sparse incremental EM in the next section, and its time complexity in Section 2.3.3.

2.3.2 Sparse Incremental EM

For the insertion step of the greedy algorithm we do sparse incremental EM steps for each candidate component. For that we treat the mixture f_{k+1} as a 2-component mixture, with one component being a new candidate ϕ and the other being the old mixture f_k:

$$f_{k+1} = \alpha\phi + (1-\alpha)f_k. \tag{2.13}$$

Furthermore when updating we keep the old mixture fixed and only update the parameters of the new component and the mixing weight α.

Since we only update the parameters of the new component, only the responsibilities of the new component are needed. These responsibilities will probably be very small for the points lying far from the new component, and since every candidate is generated from an existing parent component time is saved by updating the new component's parameters using only those points which the parent component 'owns', i.e. for which it has highest responsibility. We set the responsibilities for all other points to a predetermined distribution, one that gives responsibility 0 to the new component and 1 to the old mixture. The resulting error is relatively small and the closer the responsibilities of the new component for the points outside its area are to 0, the closer Q gets to the posterior distribution and the tighter the lower bound gets. Because $\mathcal{F}_{\boldsymbol{x}}$ is optimised for the points that belong to the parent component, $\mathcal{F}$ will never decrease. Therefore this sparse incremental version of EM is justified.

Let ϕ_{par} be the parent component of the new candidate. Now define a subset $\mathcal{A}$ of the data set $\boldsymbol{X}$ that consists of all points that give highest responsibility to the parent component:

$$\mathcal{A} = \{\boldsymbol{x} \in \boldsymbol{X} : \phi_{par} = \arg\max_s q_{\boldsymbol{x}}(s)\}.$$

The update equation for the E-step can be calculated by setting the first derivative of (2.7) with respect to the responsibilities to 0. The responsibilities, of the new component ϕ and for the points $\boldsymbol{x} \in \mathcal{A}$ only, become:

$$q_{\boldsymbol{x}} = \frac{\alpha\phi(\boldsymbol{x};\theta_{k+1})}{(1-\alpha)f_k + \alpha\phi(\boldsymbol{x};\theta_{k+1})}. \tag{2.14}$$

Maximising the parameters of the new component only in the M-step using the responsibilities found in the E-step and the fact that $\forall \boldsymbol{x} \notin \mathcal{A} : q_{\boldsymbol{x}} = 0$ gives the following update equations:

$$\alpha' = \frac{1}{n}\sum_{\boldsymbol{x}\in\mathcal{A}} q_{\boldsymbol{x}}, \tag{2.15}$$

$$\boldsymbol{m}'_{k+1} = \frac{1}{n\alpha'}\sum_{\boldsymbol{x}\in\mathcal{A}} q_{\boldsymbol{x}}\boldsymbol{x}, \tag{2.16}$$

$$\boldsymbol{C}'_{k+1} = \frac{1}{n\alpha'}\sum_{\boldsymbol{x}\in\mathcal{A}} q_{\boldsymbol{x}}\boldsymbol{x}\boldsymbol{x}^\top - \boldsymbol{m}'_{k+1}\boldsymbol{m}'^\top_{k+1}. \tag{2.17}$$

For more details see Verbeek et al. [13].

2.3.3 Time Complexity

The total number of computations needed for the km sparse incremental EM searches is $O(mn)$ because in each E-step on average n/k responsibilities have to be updated, and since the likelihoods of the data points for the old mixture f_k were already computed during previous EM steps only the likelihoods of the n/k points for the new component need to be calculated. The number of main computations in the M-step is of the same order since n/k responsibilities need to be summed for each of the km candidates.

Complete EM takes $O(nk)$ computations as I have explained in Section 2.2.4. Suppose the maximum number of components used is k. If complete EM is carried out between insertion steps then the total number of computations for the entire algorithm is $O(nk^2+nkm)$. If complete EM is not carried out between insertion steps then that number becomes $O(nkm)$. Note that this is a factor m more than regular EM, however as said previously greedy EM doesn't need to be run multiple times as regular EM does. Also m does not need to be large. There are theorems that allow us to compute values for m such that the best candidate is guaranteed with high probability to be among the, say, best 5% of all possible candidates per component [13].

2.4 Related Work

Recently other methods for improving or speeding up the regular EM algorithm were proposed. Among these were several methods that split the data set in a number of groups, analysed the data in these groups prior to running EM and then during EM only used the results of the analyses, also called *sufficient statistics*, to optimise mixture log-likelihood. These methods are actually clustering groups of points rather than clustering data points. These methods have the advantage that the number of computations no longer depend directly on the number of data points, but on the number of groups of data, which is very useful for applications that involve large data sets. Trade-offs between the quality of the result of the algorithm and the speed are also possible.

2.4.1 EM using a kd-tree

One such method I will decribe further is proposed by Moore [9]. It uses a kd-tree (see Bentley [2]) to store the sufficient statistics of the data. An kd-tree is a binary tree in which every node corresponds to a subset of the data set. The points in a node get split into two parts according to some criterion and these two parts are the points that correspond to the two children nodes, which then get split in two, and so on.

In [9] the points themselves are not stored in the tree, only some statistics about the points. For a particular node b those are the *number of points* $|b|$, *centroid* $\langle \boldsymbol{x} \rangle_b$ and '*covariance*' $\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle_b$, with the latter two defined as:

$$\langle \boldsymbol{x} \rangle_b = \frac{1}{|b|} \sum_{\boldsymbol{x} \in b} \boldsymbol{x} \tag{2.18}$$

$$\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle_b = \frac{1}{|b|} \sum_{\boldsymbol{x} \in b} \boldsymbol{x}\boldsymbol{x}^\top . \tag{2.19}$$

Note that this covariance is not the real covariance of the points in a node. To get the real covariance we would first have to subtract the centroid from all points. For this algorithm only $\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle_b$ is needed however.

The nodes are split in two through the middle along their widest dimension, with dimension width defined as the largest distance in that dimension between any two points from the node. Note that nodes are thus always split axis-aligned, leading to hyperrectangular boxes bounding all points in a node.

When performing an iteration of the EM algorithm, the parameters need to be updated using (2.10)-(2.12). It is clear that the only values needed for this are $\sum_{\boldsymbol{x} \in \boldsymbol{X}} q_{\boldsymbol{x}}(s)$, $\sum_{\boldsymbol{x} \in \boldsymbol{X}} q_{\boldsymbol{x}}(s)\boldsymbol{x}$ and $\sum_{\boldsymbol{x} \in \boldsymbol{X}} q_{\boldsymbol{x}}(s)\boldsymbol{x}\boldsymbol{x}^\top$ for each component s. We can approximate these values from a node of the tree using the following equations:

$$\sum_{\boldsymbol{x}\in b} q_{\boldsymbol{x}}(s) \approx \overline{q}(s)|b|, \tag{2.20}$$

$$\sum_{\boldsymbol{x}\in b} q_{\boldsymbol{x}}(s)\boldsymbol{x} \approx \overline{q}(s)|b|\langle \boldsymbol{x}\rangle_b, \tag{2.21}$$

$$\sum_{\boldsymbol{x}\in b} q_{\boldsymbol{x}}(s)\boldsymbol{x}\boldsymbol{x}^\top \approx \overline{q}(s)|b|\langle \boldsymbol{x}\boldsymbol{x}^\top\rangle_b, \tag{2.22}$$

where all terms on the right hand side are stored as sufficient statistics in the node and $\overline{q}$ is an approximation of the responsibility of all points in the node.

The algorithm begins by computing these values for the root node, since that node corresponds to the entire data set. To compute these values for any particular node it can also get the values from the two children nodes and sum those, since $\sum_{\boldsymbol{x}\in\boldsymbol{X}} q_{\boldsymbol{x}}(s) = \sum_b \sum_{\boldsymbol{x}\in b} q_{\boldsymbol{x}}(s)$ where b are the children nodes. Therefore, a choice needs to be made at each node whether it should be an end node, for which the approximation $\overline{q}$ is computed. If it is not an end node then its children's values are computed and summed. The deeper we go into the tree, the smaller the boxes become, and the better the approximation becomes. So we need a criterion for the number of end nodes.

Moore [9] lets the choice of when to stop going deeper into the tree, summing children nodes' values, depend on the difference between the maximal and minimal possible values for the responsibilities. In other words, when for all components the theoretically highest and lowest responsibility for any possible point in a node are 'close', then it is not necessary to go any deeper, since then the approximation $\overline{q}$ is guaranteed to be accurate. This, however, requires some complex computations. I will explain the reason for this criterion below.

Moore calculates the approximation of the responsibility in the following way:

$$\overline{q}(s) = p(s|\langle \boldsymbol{x}\rangle_b) = \frac{\pi_s p(\langle \boldsymbol{x}\rangle_b|s)}{f_k(\langle \boldsymbol{x}\rangle_b)}. \tag{2.23}$$

In other words, the responsibility of a component for the points in a node is approximated using the responsibility for the centroid of a node. Of course the responsibility of the centroid does not tell us anything about the optimal approximated responsibility for the points in a node. This means that the deeper one goes in the tree the better the approximation becomes, because the responsibility will vary less in smaller nodes and therefore the responsibility of the centroid will probably be closer to the true responsibility.

This is why the criterion mentioned above that all possible responsibilities in a node need to be 'close' together in order to stop descending the tree is very important, since only then the approximation $\overline{q}$ is guaranteed to be accurate, i.e. $\sum_{\boldsymbol{x}} q_{\boldsymbol{x}}\boldsymbol{x} \approx \overline{q}\sum_{\boldsymbol{x}} \boldsymbol{x}$.

2.4.2 Time Complexity

This algorithm consists of two parts, building the tree and doing EM iterations. Building the tree requires $O(n \log n)$ computations, because on every layer of the tree a distance has to be calculated (for the splitting criterion) for every data point. There are $O(\log n)$ layers and $O(n)$ data points. Building the tree needs to be done only once though, so this building time is not that important. Doing one EM iteration does not cost time linear in the number of data points anymore. The number of times in each E-step that the likelihood $p(\langle \boldsymbol{x} \rangle_b | s)$ has to be calculated is once for each end node and each component, so $O(kB)$ when B is the number of end nodes in an iteration.

2.5 Motivation

As I discussed above the free energy generalisation of the EM algorithm opens possibilities for a number of different, but justified, methods that can have certain trade-offs between quality and speed without breaking the conditions necessary for guaranteeing at least no decrease in model quality w.r.t. $\mathcal{F}$ by doing EM steps. Furthermore greedy methods can ensure good results with every run while using only few extra computations. Finally, methods that use cached sufficient statistics based on geometric partitioning of the data space can give significant speed-ups, avoiding time complexity linear to the size of the data set, while paying only little with respect to quality, making these methods very useful for large scale applications.

This of course raises the question if these latter two algorithms can be combined to give an algorithm that is capable of handling large data sets, while avoiding some of the normal problems of EM, by using a greedy scheme. In the next chapter I will describe such a method and show that it indeed has the expected qualities.

Chapter 3

Chunky EM

In this chapter I will describe an algorithm that uses a partitioning of the data set to speed up the EM algorithm, and in particular the greedy EM.

3.1 Partitions

Let us first describe a convenient way of denoting a partition of a data set. Since the versions of EM described in the previous chapter that use a kd-tree only add the sufficient statistics from the leaves of the tree, one could look at those leaves as disjoint subsets of the data set that together contain the entire data set.

When $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ is the data set let $\mathcal{B}$ be a *partition* of the data set:

$$\mathcal{B} = \{b_1, \ldots, b_r\}, \tag{3.1}$$

where each *box* b is given by:

$$b = \{\boldsymbol{x}_k, \ldots, \boldsymbol{x}_l\}, \tag{3.2}$$

such that

$$b_1 \cup \ldots \cup b_r = \boldsymbol{X}, \tag{3.3}$$

and

$$\forall_{i \neq j} : b_i \cap b_j = \emptyset. \tag{3.4}$$

Let *expanding* or *refining* a partition be defined as increasing the number of boxes in that partition. Analogous to the kd-tree described previously this means splitting the points in a box in two groups, resulting in two new boxes that replace the old box in the partition. Note that going deeper in a kd-tree can be seen as refining a partition.

3.1.1 Splitting Boxes

In Moore's method [9] nodes or boxes were split axis aligned, and through the middle of their widest dimension. While this is quite fast and easy to do, it also has a disadvantage. Often the data and its clusters will not be axis aligned. This can lead to poor partitions, because one easily gets largely empty boxes with only few points. The less the points in a box are uniformly distributed the worse any approximation based on sufficient statistics will be. This means partitions will be unnecessarily large before they can be used. This problem will only get worse when working on higher dimensional data spaces, since the clusters will be even less axis aligned, resulting in more emptiness in boxes.

Therefore a better splitting method is one based on the orientation of the points themselves. Calculating the covariance of all points in a box provides us with a notion of this orientation. If we then take the largest eigenvector of this covariance matrix we get the main orientation of the cluster of points in a box. Splitting the box using a hyperplane perpendicular to the middle of that eigenvector will result in better partitions since large data clusters are better split up into smaller chunks. One could look at this as performing a recursive Principal Component Analysis on the data, a technique used in combination with kd-trees to limit its decrease in performance with high dimensional spaces. [12]

One might ask why Moore didn't use this splitting method. This is because his criterion for determining whether or not to go deeper in the tree needs hyperrectangular axis aligned boxes. The algorithm I will describe does not need this, however, and therefore splitting along the eigenvector is a preferable choice.

3.1.2 Sufficient Statistics

For our algorithm we also use the sufficient statistics instead of the points themselves. Because the splitting method described above does not depend on the model, but only on the data set which never changes, it is possible to calculate all statistics before doing EM. We will store in each box b the same statistics as Moore, namely the *number of points* $|b|$, *centroid* $\langle \boldsymbol{x} \rangle_b$ and '*covariance*' $\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle_b$, with the latter two defined as:

$$\langle \boldsymbol{x} \rangle_b = \frac{1}{|b|} \sum_{\boldsymbol{x} \in b} \boldsymbol{x} \tag{3.5}$$

$$\langle \boldsymbol{x}\boldsymbol{x}^\top \rangle_b = \frac{1}{|b|} \sum_{\boldsymbol{x} \in b} \boldsymbol{x}\boldsymbol{x}^\top . \tag{3.6}$$

Note again that (3.6) does not compute the real covariance of the points in a box, for which we would need to subtract the centroid from each data point first.

3.2 Chunky EM

As above the goal is to increase the total free energy $\mathcal{F}$ of the model, which is the sum of the free energy for each data point $\boldsymbol{x}$:

$$\begin{aligned} \mathcal{F}_{\boldsymbol{x}}(Q,\theta) &= \log p(\boldsymbol{x};\theta) - D_{KL}(q_{\boldsymbol{x}} \parallel p(s|\boldsymbol{x})) & (3.7) \\ &= E_{q_{\boldsymbol{x}}} \log p(\boldsymbol{x}, s; \theta) + \mathcal{H}(q_{\boldsymbol{x}}). & (3.8) \end{aligned}$$

This is done by calculating the optimal responsibilities Q for the given parameters θ and then updating the model parameters using this Q:

$$\begin{aligned} \pi'_s &= \frac{1}{n}\sum_{i=1}^{n} q_i(s), & (3.9) \\ \boldsymbol{m}'_s &= \frac{1}{n\pi'_s}\sum_{i=1}^{n} q_i(s)\boldsymbol{x}_i, & (3.10) \\ \boldsymbol{C}'_s &= \frac{1}{n\pi'_s}\sum_{i=1}^{n} q_i(s)\boldsymbol{x}_i\boldsymbol{x}_i^{\top} - \boldsymbol{m}'_s\boldsymbol{m}'^{\top}_s. & (3.11) \end{aligned}$$

3.2.1 Shared Responsibilities

The idea of chunky EM is to assign the same responsibility to all points in a box b of a partition $\mathcal{B}$. If we call this responsibility $\overline{q}_b$, and using the fact that the total free energy is also equal to the sum of the free energy $\mathcal{F}_b$ for each box in the partition, the free energy equation for a box b can be written as:

$$\begin{aligned} \mathcal{F}_b(Q,\theta) &= \sum_{\boldsymbol{x}\in b}[\log p(\boldsymbol{x};\theta) - D_{KL}(\overline{q}_b \parallel p(s|\boldsymbol{x}))] & (3.12) \\ &= \sum_{\boldsymbol{x}\in b}[E_{\overline{q}_b} \log p(\boldsymbol{x}, s; \theta) + \mathcal{H}(\overline{q}_b)]. & (3.13) \end{aligned}$$

The update equation for the responsibilities can be found by setting the derivative of (3.12) with respect to $\overline{q}_b$ to 0:

$$\overline{q}_b(s) = \frac{\pi_s \exp\langle \log p(\boldsymbol{x}|s)\rangle_b}{\sum_{s'} \pi_{s'} \exp\langle \log p(\boldsymbol{x}|s')\rangle_b}. \quad (3.14)$$

Here $\langle\,\cdot\,\rangle_b = \frac{1}{|b|}\sum_{\boldsymbol{x}\in b}(\cdot)$ denotes the average over all points in a box b.

The update equations for the parameters can be calculated by setting the derivative of (3.13) with respect to θ to 0:

$$\pi'_s = \frac{1}{n}\sum_{b\in\mathcal{B}} \overline{q}_b(s)|b|, \tag{3.15}$$

$$\boldsymbol{m}'_s = \frac{1}{n\pi'_s}\sum_{b\in\mathcal{B}} \overline{q}_b(s)|b|\langle \boldsymbol{x}\rangle_b, \tag{3.16}$$

$$\boldsymbol{C}'_s = \frac{1}{n\pi'_s}\sum_{b\in\mathcal{B}} \overline{q}_b(s)|b|\langle \boldsymbol{x}\boldsymbol{x}^\top\rangle_b - \boldsymbol{m}'_s\boldsymbol{m}'^\top_s. \tag{3.17}$$

Note that for these updates only the stored sufficient statistics and $\overline{q}_b$ are needed.

It is clear that main computation for (3.14) is that of $\langle \log p(\boldsymbol{x}|s)\rangle_b$. If we would have to do this by computing the log-likelihood of every data point then we would gain nothing over regular EM. However, it is possible to rewrite the average:

$$\langle \log p(\boldsymbol{x}|s)\rangle_b = \frac{1}{|b|}\sum_{\boldsymbol{x}\in b}\log p(\boldsymbol{x}|s) =$$
$$-\frac{1}{2}\left[d\log 2\pi + \log|\boldsymbol{C}_s| + \frac{1}{|b|}\sum_{\boldsymbol{x}\in b}(\boldsymbol{x}-\boldsymbol{m}_s)^\top \boldsymbol{C}_s^{-1}(\boldsymbol{x}-\boldsymbol{m}_s)\right] =$$
$$-\frac{1}{2}\left[d\log 2\pi + \log|\boldsymbol{C}_s| + \boldsymbol{m}_s^\top \boldsymbol{C}_s^{-1}\boldsymbol{m}_s + \langle \boldsymbol{x}^\top \boldsymbol{C}_s^{-1}\boldsymbol{x}\rangle_b - 2\boldsymbol{m}_s^\top \boldsymbol{C}_s^{-1}\langle \boldsymbol{x}\rangle_b\right] =$$
$$-\frac{1}{2}\left[d\log 2\pi + \log|\boldsymbol{C}_s| + \boldsymbol{m}_s^\top \boldsymbol{C}_s^{-1}\boldsymbol{m}_s + \mathrm{Trace}\{\boldsymbol{C}_s^{-1}\langle \boldsymbol{x}\boldsymbol{x}^\top\rangle_b\} - 2\boldsymbol{m}_s^\top \boldsymbol{C}_s^{-1}\langle \boldsymbol{x}\rangle_b\right],$$

where Trace$\{\cdot\}$ is the sum of the diagonal elements of a matrix. With this it is possible to calculate the optimal average responsibility of a box fast using the cached sufficient statistics. From (3.14) it is also clear that when a partition is refined so much that each box contains only one data point, the equations are equal to those of regular EM, since then $\exp\langle \log p(\boldsymbol{x}|s)\rangle_b = p(\boldsymbol{x}|s)$.

3.2.2 Partition Size

The last issue that needs to be solved is the question of which partition to use during the EM steps. It is clear that the finer the partition gets the closer the shared responsibilities get to the real responsibilities, thus resulting in a better approximation and better convergence. However, a finer partition also means more computations for each EM step, because (3.14)-(3.17) have a computational cost linear in the size of the partition, which we would like to avoid.

The method we will be using for this is to start with a relatively coarse partition, converge the model using this partition, and then refine the partition and converge again, and so on. The idea behind this is that, because of the optimal shared responsibilities, a coarse partition will still guarantee no decrease of the free energy. Note that this is not true for Moore's method. Therefore it is possible to do fast EM steps with coarse partitions, pushing the model roughly towards a maximum, which will be reached using finer partitions.

Several methods are possible for choosing the amount of refinement. A simple, low cost method would be to refine each box once before applying EM. This might lead to unnecessary expansion, however, and therefore a better, but more costly, way would be to expand the boxes whose refinement will lead to the best increase of the free energy. This increase can be calculated by comparing a box' optimal shared responsibilities to those of its children boxes.

Several choices can be made for when to stop expanding. One could for example look at the increase in free energy after expansion, and stop when this increase is below a certain convergence threshold. Setting an upper limit to the partition length is also possible.

3.2.3 Time Complexity

In every E-step of the algorithm the responsibilities for each box and each component need to be calculated, requiring $O(kB)$ computations, where B is the total number of boxes comprising the partition. Note that calculating $\langle \log p(\boldsymbol{x}|s) \rangle_b$ for one box requires about as much work as calculating $p(\boldsymbol{x}; \theta_s)$ for one data point $\boldsymbol{x}$. If data partitions are implemented using a kd-tree then building the tree requires $O(n \log n)$ computations, but only needs to be done once.

3.3 Greedy Chunky EM

As discussed in the previous chapter there are several reasons for using a greedy method, namely, (i) there is no dependency on parameter initialisation, and (ii) no prior knowledge about the number of clusters is needed.

The greedy algorithm consists of two parts, converging a k-component model and inserting a new component leading to a $k+1$-component model. These two parts are repeated until adding components no longer improves the model quality or some other model complexity measure that uses for example a test set. The convergence of a k-component mixture model is done with the chunky EM described in the previous section. The insertion procedure is somewhat different from the non-chunky version and I will discuss it below.

3.3.1 Chunky Insertion

In the component insertion step, greedy EM [13] as described in the previous chapter divides the data points in groups which have highest responsibility for a particular component. The chunky version divides the boxes in the partition k disjoint subsets, where each subset is comprised of boxes that have highest average responsibility for a model component. It then also randomly splits these subsets in two parts, from which two candidate components are created. This is repeated until m candidates are generated for each of the k components. These km candidates are then trained using sparse incremental chunky EM steps, explained in the next section. The component that will be inserted into the mixture is the trained candidate that results in the highest free energy.

The initial parameters of the candidates, except for the mixing weight α which is set to half the mixing weight of the parent component, are not trivially found, because we do not want to look at all data points, but only at the boxes and the sufficient statistics stored therein. Suppose we have a partition $\mathcal{B}$, that got divided into k subsets, and that one such subset $\mathcal{A}$ got split into two subsets $\mathcal{A}_1$ and $\mathcal{A}_2$. These two subsets will both generate a candidate, which will need an initial mean and covariance. This mean and covariance must be the mean and covariance of the points in the corresponding boxes.

For example, for group $\mathcal{A}_1$ we can write:

$$\boldsymbol{m} = \frac{1}{|\mathcal{A}_1|}\sum_{\boldsymbol{x}\in\mathcal{A}_1}\boldsymbol{x} = \frac{1}{|\mathcal{A}_1|}\sum_{b\in\mathcal{A}_1}\sum_{\boldsymbol{x}\in b}\boldsymbol{x} = \frac{1}{|\mathcal{A}_1|}\sum_{b\in\mathcal{A}_1}|b|\langle\boldsymbol{x}\rangle_b, \tag{3.18}$$

where $|\mathcal{A}_1|$ is the total number of points in the boxes in $\mathcal{A}_1$, and:

$$\begin{aligned}
\boldsymbol{C} &= \frac{1}{|\mathcal{A}_1|}\sum_{\boldsymbol{x}\in\mathcal{A}_1}(\boldsymbol{x}-\boldsymbol{m})(\boldsymbol{x}-\boldsymbol{m})^\top \\
&= \frac{1}{|\mathcal{A}_1|}\sum_{b\in\mathcal{A}_1}\sum_{\boldsymbol{x}\in b}\left[\boldsymbol{x}\boldsymbol{x}^\top - \boldsymbol{x}\boldsymbol{m}^\top - \boldsymbol{m}\boldsymbol{x}^\top + \boldsymbol{m}\boldsymbol{m}^\top\right] \\
&= \frac{1}{|\mathcal{A}_1|}\sum_{b\in\mathcal{A}_1}|b|\left[\langle\boldsymbol{x}\boldsymbol{x}^\top\rangle_b - \langle\boldsymbol{x}\rangle_b\boldsymbol{m}^\top - \boldsymbol{m}\langle\boldsymbol{x}\rangle_b^\top + \boldsymbol{m}\boldsymbol{m}^\top\right] \\
&= \frac{1}{|\mathcal{A}_1|}\sum_{b\in\mathcal{A}_1}|b|\langle\boldsymbol{x}\boldsymbol{x}^\top\rangle_b - \boldsymbol{m}\boldsymbol{m}^\top. \qquad (3.19)
\end{aligned}$$

Both $\boldsymbol{m}$ and $\boldsymbol{C}$ can be calculated without requiring the data points themselves.

3.3.2 Sparse Incremental Chunky

In the insertion part of the greedy chunky algorithm we have to decide which new component to insert. Logically this decision should be made based on the increase in total free energy after the new component is fully inserted in the existing mixture. To get a notion of the increase we will perform a certain number of sparse incremental EM steps on the new mixture. At this moment we are only interested in finding the best candidate and not in training the model.

In sparse incremental EM we treat the mixture as a 2-component mixture, with the first component being the candidate component and the second component being the old mixture:

$$f_{k+1} = \alpha\phi + (1-\alpha)f_k. \tag{3.20}$$

The old mixture parameters are kept fixed and only the parameters $\boldsymbol{m}_{k+1}$ and $\boldsymbol{C}_{k+1}$ of the new component and the mixing weight α are updated.

Since we only update the parameters of the new component only the responsibilities of the new component are needed. Similar to Section 2.3.2, these responsibilities will probably be very small for the boxes lying far from the new component, and since every candidate is chosen from an existing parent component we can save time by updating the new component's parameters using only the boxes in $\mathcal{A}$ for which the parent component has highest responsibility, as described in the previous section. We set the responsibilities for all other boxes to a predetermined distribution, one that gives responsibility 0 to the new component and 1 to rest of the mixture.

So suppose we want to insert a new candidate component ϕ into a current mixture f_k, we have a partition $\mathcal{B}$ of the data set and ϕ comes from parent component ϕ_{par}. We have selected a subset $\mathcal{A}$ of $\mathcal{B}$:

$$\mathcal{A} = \{b \in \mathcal{B} : \phi_{par} = \arg\max_s \overline{q}_b(s)\}.$$

We want to maximise the free energy $\mathcal{F}$ for each box b, given by:

$$\begin{aligned}\mathcal{F}_b(Q,\theta) &= \sum_{\boldsymbol{x}\in b}[\log p(\boldsymbol{x};\theta) - D_{KL}(\overline{q}_b \parallel p(s|\boldsymbol{x}))] && (3.21)\\ &= \sum_{\boldsymbol{x}\in b}[E_{\overline{q}_b}\log p(\boldsymbol{x},s;\theta) + \mathcal{H}(\overline{q}_b)]. && (3.22)\end{aligned}$$

Because we consider f_{k+1} to be a 2-component mixture we can write $\overline{q}_b$ as the responsibility of the new component and thus $(1-\overline{q}_b)$ as the responsibility of the old mixture for a certain box b.

Inserting (3.20) into (3.22) gives:

$$\begin{aligned}\mathcal{F}_b^{k+1} &= \sum_{\boldsymbol{x}\in b}\overline{q}_b\log(\alpha\phi(\boldsymbol{x})) + (1-\overline{q}_b)\log((1-\alpha)f_k(\boldsymbol{x})) + \mathcal{H}(\overline{q}_b)\\ &= \sum_{\boldsymbol{x}\in b}\overline{q}_b[\log\phi(\boldsymbol{x}) + \log\alpha] + (1-\overline{q}_b)[\log f_k(\boldsymbol{x}) + \log(1-\alpha)] + \mathcal{H}(\overline{q}_b).\end{aligned}$$

Since $\mathcal{F}_b^k$ is a lower bound on $\sum_{\boldsymbol{x}\in b} \log f_k(\boldsymbol{x})$ we can replace the latter, giving the following formula for the free energy of a box:

$$\begin{aligned}\mathcal{F}_b^{k+1} &\geq \overline{q}_b \left[\sum_{\boldsymbol{x}\in b} \log \phi(\boldsymbol{x}) + |b| \log \alpha\right] \\ &\quad +(1-\overline{q}_b)\left[\mathcal{F}_b^k + |b|\log(1-\alpha)\right] + |b|\mathcal{H}(\overline{q}_b). \qquad (3.23)\end{aligned}$$

Expectation Step

In the E-step we maximise our lower bound on $\mathcal{F}_b^{k+1}$ by optimising $\overline{q}_b$ for each box. We can compute the optimal $\overline{q}_b$ by setting the derivative of (3.23) with respect to $\overline{q}_b$ to 0. This gives:

$$\sum_{\boldsymbol{x}\in b} \log \phi(\boldsymbol{x}) + |b|\log\alpha - \mathcal{F}_b^k - |b|\log(1-\alpha) - |b|\log\overline{q}_b + |b|\log(1-\overline{q}_b) = 0,$$

which we rewrite to:

$$\overline{q}_b = \frac{\alpha \exp \langle \log \phi(\boldsymbol{x})\rangle_b}{(1-\alpha)\exp(\mathcal{F}_b^k/|b|) + \alpha \exp \langle \log\phi(\boldsymbol{x})\rangle_b}, \qquad (3.24)$$

where $\langle \log\phi(\boldsymbol{x})\rangle_b$ is the average log-likelihood of the points in the box under the new component which we can compute fast using cached statistics (see Section 3.2.1).

Maximisation Step

In the maximisation step we optimise our bound $\mathcal{F}_\mathcal{B}^{k+1} = \sum_{b\in\mathcal{B}} \mathcal{F}_b^{k+1}$ by computing new parameters for the new component based on the $\overline{q}_b$ found with (3.24). Using (3.23) we can write:

$$\begin{aligned}\mathcal{F}_\mathcal{B}^{k+1} &\geq \sum_{b\in\mathcal{B}} \overline{q}_b \big[\sum_{\boldsymbol{x}\in b} \log\phi(\boldsymbol{x}) + |b|\log\alpha\big] \\ &\quad + \sum_{\boldsymbol{x}\in\mathcal{B}} (1-\overline{q}_b)\big[\mathcal{F}_b^k + |b|\log(1-\alpha)\big] + \sum_{b\in\mathcal{B}} |b|\mathcal{H}(\overline{q}_b) \qquad (3.25)\end{aligned}$$

We can now use our approximation to set the responsibility of the new component for all boxes outside $\mathcal{A}$ to 0, like in [13]. In other words:

$$\forall\ b \notin \mathcal{A} : \overline{q}_b = 0$$

Combining (3.25) and (3.3.2) gives:

$$\begin{aligned}\mathcal{F}_\mathcal{B}^{k+1} &\geq \sum_{b\in\mathcal{A}} \overline{q}_b \big[\sum_{\boldsymbol{x}\in b} \log\phi(\boldsymbol{x}) + |b|\log\alpha\big] \\ &\quad + \sum_{b\in\mathcal{B}} (1-\overline{q}_b)\big[\mathcal{F}_b^k + |b|\log(1-\alpha)\big] + \sum_{b\in\mathcal{A}} |b|\mathcal{H}(\overline{q}_b) \qquad (3.26)\end{aligned}$$

Setting the derivative of (3.26) with respect to α to 0 gives:

$$\sum_{b\in\mathcal{A}} \overline{q}_b |b| \frac{1}{\alpha} - \sum_{b\in\mathcal{B}} (1-\overline{q}_b)|b| \frac{1}{1-\alpha} = 0,$$

thus

$$\alpha_{k+1} = \frac{\sum_{b\in\mathcal{A}} \overline{q}_b |b|}{N}, \tag{3.27}$$

where N is the total number of data points.

Setting the derivative of (3.26) with respect to $\boldsymbol{m}_{k+1}$ to 0 leads to:

$$\sum_{b\in\mathcal{A}} \overline{q}_b \sum_{\boldsymbol{x}\in b} (-\boldsymbol{C}_{k+1}^{-1}\boldsymbol{x} + \boldsymbol{C}_{k+1}^{-1}\boldsymbol{m}_{k+1}) = 0,$$

$$\sum_{b\in\mathcal{A}} \overline{q}_b (-\sum_{\boldsymbol{x}\in b} \boldsymbol{x} + |b|\boldsymbol{m}_{k+1}) = 0,$$

and

$$\boldsymbol{m}_{k+1} = \frac{\sum_{b\in\mathcal{A}} \overline{q}_b |b| \langle \boldsymbol{x} \rangle_b}{\sum_{b\in\mathcal{A}} \overline{q}_b |b|} \tag{3.28}$$

Setting the derivative of (3.26) with respect to $\boldsymbol{C}_{k+1}$ to 0 gives:

$$\sum_{b\in\mathcal{A}} \overline{q}_b \sum_{\boldsymbol{x}\in b} (-\boldsymbol{C}_{k+1} + (\boldsymbol{x}-\boldsymbol{m}_{k+1})(\boldsymbol{x}-\boldsymbol{m}_{k+1})^\top) = 0,$$

thus

$$\begin{aligned} \boldsymbol{C}_{k+1} &= \frac{\sum_{b\in\mathcal{A}} \overline{q}_b \sum_{\boldsymbol{x}\in b} (\boldsymbol{x}-\boldsymbol{m}_{k+1})(\boldsymbol{x}-\boldsymbol{m}_{k+1})^\top}{\sum_{b\in\mathcal{A}} \overline{q}_b |b|} \\ &= \frac{\sum_{b\in\mathcal{A}} \overline{q}_b |b| \langle \boldsymbol{x}\boldsymbol{x}^\top \rangle_b}{\sum_{b\in\mathcal{A}} \overline{q}_b |b|} - \boldsymbol{m}_{k+1}\boldsymbol{m}_{k+1}^\top. \end{aligned} \tag{3.29}$$

It is clear that the parameters of the new component now only depend on the boxes in $\mathcal{A}$ and that the sparse incremental EM steps use time based on the size of $\mathcal{A}$.

3.3.3 Time Complexity

If B is the size of the partition, then sparse incremental chunky EM requires the update of on average B/k responsibilities in the E-step, so the total number of computations required for the km candidates for each EM loop is $O(mB)$. As explained above, complete chunky EM requires $O(Bk)$ computations. Suppose the maximum number of components used is k, then if complete chunky EM is done between insertion steps the total number of computations for the entire algorithm is $O(Bk^2+Bkm)$. If complete chunky EM is not done between insertion then that number becomes $O(Bkm)$. This is a factor m more than non-greedy chunky EM, however as said previously greedy chunky EM doesn't need to be run multiple times as chunky EM does. As above, building a kd-tree requires $O(n \log n)$ computations.

Chapter 4

Experimental Results

4.1 Generating Data Sets

As mentioned earlier the EM algorithm can be applied on data that is assumed to have been generated by a Gaussian mixture. Therefore, I will conduct my experiments on data that has been artificially generated from a random Gaussian mixture. For these data sets several parameters can be specified.

These are the number of data points n, the number of dimensions d, the number of components k of the generating mixture, and for each component the mixing weight π, mean $\boldsymbol{m}$, and covariance matrix $\boldsymbol{C}$. In all experiments n, d, and k are set to a specified number, while the component parameters are randomly chosen. These component parameters can be constrained by the component separation c [4], given by:

$$\forall_{i \neq j} : ||\boldsymbol{m}_i - \boldsymbol{m}_j|| \geq c\sqrt{\max\left\{\mathrm{Trace}(\boldsymbol{C}_i), \mathrm{Trace}(\boldsymbol{C}_j)\right\}}. \tag{4.1}$$

In other words, the amount of separation specifies the minimum distance between component means based on the size of their covariance matrix. See Fig. 4.1 for examples of data sets with different amounts of component separation.

4.1.1 Model Initialisation

All EM algorithms need to be initialised with a mixture model. For the greedy variants, which are initialised with a 1-component mixture, the starting model is trivially found, since the mean and covariance of the only component are simply the mean and covariance of the complete data set, and its mixing weight is of course 1. The non-greedy EM versions, however, need to be initialised with a k-component mixture, and since the end result of EM depends heavily on its initialisation, the choice for the starting mixture requires some care.

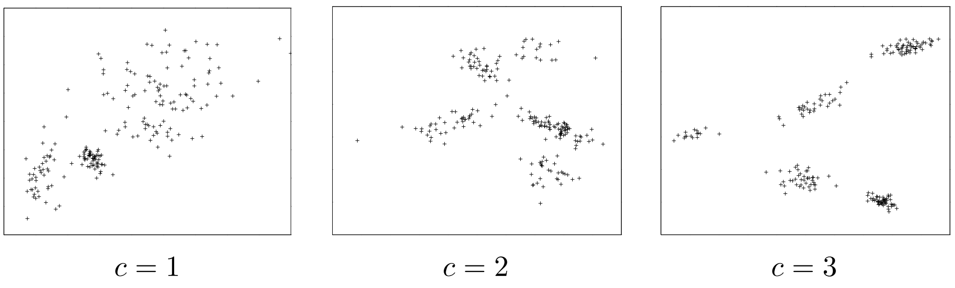


Figure 4.1: Examples of data sets generated by a 5-component Gaussian mixture with different amounts of component separation.

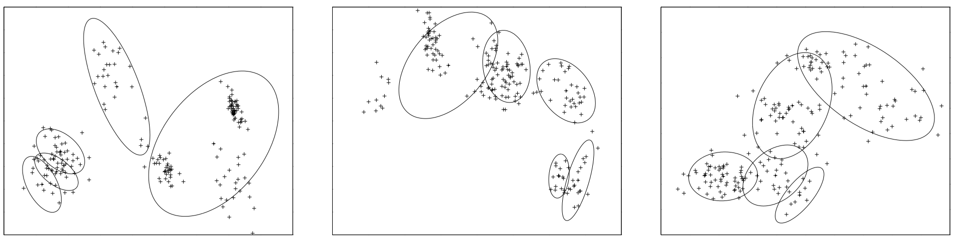


Figure 4.2: Examples of k-means initialised Gaussian mixtures.

Therefore, I will use the k-means algorithm for initialisation. First, k data points are randomly selected from the data set. Then the data set is divided in k subsets, which are also known as Voronoi Regions (VR), where each VR consists of the points that are closest to one of the selected data points. Finally, the k model components' means and covariance matrices are set to the means and covariance matrices of the points in the k VRs, and the mixing weights are set to the proportion of the data points in each VR.

Fig. 4.2 shows some examples of random model initialisations. Each ellipse corresponds to a model component, and the size and shape of the ellipse to its covariance.

4.2 Shared Responsibilities

An important part of the described improvements of the chunky EM algorithm is the computation of the *optimal* shared responsibility instead of taking the responsibility of the centroid of a box $p(s|\langle \boldsymbol{x} \rangle_b)$ as the shared responsibility [9]. The optimal shared responsibility maximises the free energy better, in other words, the divergence term in (3.12) will be smaller.

In the following experiment I will compare the quality of the two approximations in terms of partition coarseness. First, 20 data sets of 5000 points were generated from random 20-component Gaussian mixtures in 10

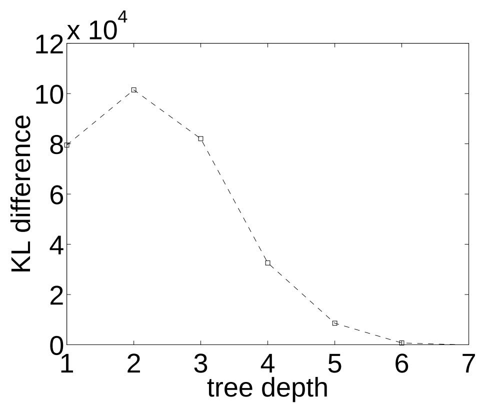


Figure 4.3: Differences in Kullback-Leibler divergence between optimal and suboptimal shared responsibilities in terms of partition size.

dimensions with a separation of 2. Then, for each data set a 20-component Gaussian mixture was initialised using k-means. Finally, the shared responsibilities were computed by using both approximations, (2.23) and (3.14), for different partition sizes.

We will compare the approximations by comparing the Kullback-Leibler divergence between the shared responsibilities and the posterior distribution over the components. Fig. 4.3 shows the divergence difference between the optimal shared responsibilities and using the responsibility of the centroid. Each unit on the horizontal axis corresponds to going one level deeper in the kd-tree, which is equivalent to refining each box in the partition once, thereby doubling the size of the partition.

The figure clearly illustrates that our approximation performs better when using coarser partitions, which is what we expected. This validates our approach of starting EM with a coarse partition and refining it during convergence.

4.3 Normal vs Chunky EM

In this section I will describe experiments to determine the influence of the number of data points, the number of dimensions, the number of components, and the amount of separation on the difference in speed and quality between regular EM and chunky EM.

The default data set in these experiments consists of 10,000 points in 2 dimensions generated from random 10-component Gaussian mixture with a separation of 3. From the same random mixture we also generated a test set of 1000 points to compare log-likelihoods. We then initialised a mixture using k-means and applied both regular EM and chunky EM to it. For each experiment we recorded the log-likelihood of the generating mixture for the test set, and for both algorithms the total number of basic floating point operations needed for convergence, and the log-likelihood of the final mixture on the test set.

Fig. 4.4 shows the averages of those values after repeating the experiment 20 times. The results show that chunky EM in general is faster than regular EM, and that this speedup is at least linear with respect to the number of data points, which the theory predicted.

Furthermore, the amount of separation has a positive effect on the speedup. This can be ascribed to the fact that when the clusters are better separated the responsibilities of the points in a cluster will differ less, making the approximation of the shared responsibilities better.

The reason for the fact that the number of dimensions has a slightly negative influence on the speedup is the decrease in performance of kd-trees with high dimensional spaces. The decrease in speedup for data sets with more clusters is probably caused by the fact that more clusters means that less points have responsibilities that are close and therefore the partitions need to be finer in order to make a good approximation.

On average the mixtures converged by chunky EM perform worse than those found by regular EM, which could be expected, since chunky EM is after all using an approximation. However, Fig. 4.4 also shows that the difference is small in comparison to the distance to the log-likelihood of the generating mixture, even when applied to high dimensional spaces with many clusters.

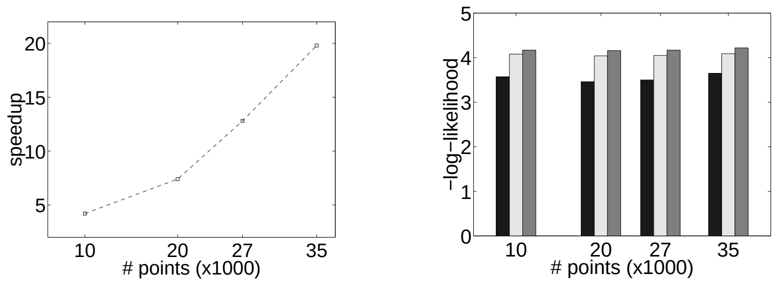
4.4 Greedy EM vs Chunky Greedy EM

In this section I will compare the greedy EM algorithm, as described in Section 2.3 with the chunky greedy EM algorithm, and look at the influence of the number of data points, the number of dimensions, the number of components, and the amount of separation on the speedup and log-likelihood differences.

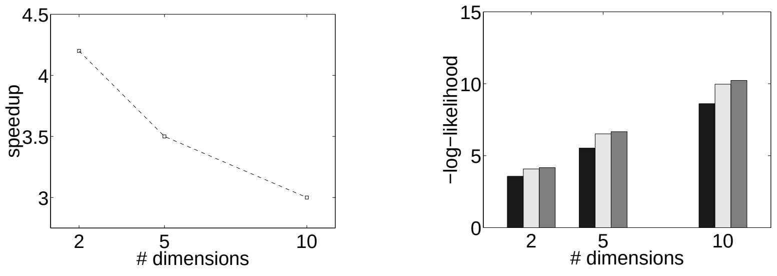
The default data set consists of 10,000 data points in 2 dimensions drawn from random 5-component 2-separated Gaussian mixtures, with a test set of 500 points drawn from the same mixture to compare log-likelihoods. Then both the chunky greedy and the non-chunky greedy EM algorithms were applied on this data set. This experiment was repeated 10 times and the averages were recorded.

The results are shown in Fig. 4.5. From the figure it is clear that the chunky version is always faster than regular greedy and that the speedup is linear with respect to the size of the data set, which the theory indicated. Furthermore, the log-likelihoods of both algorithms are practically equal to that of the generating mixture, which is what we expect from the greedy

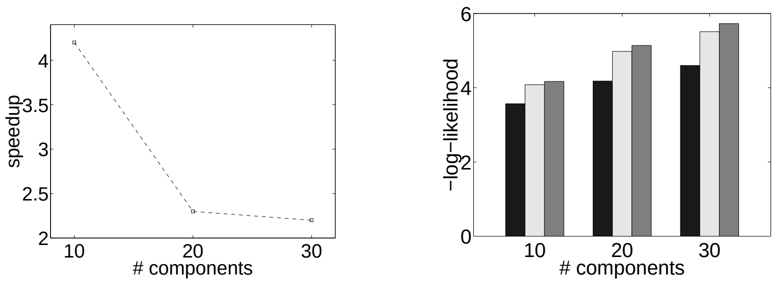
Number of data points (x1000):



Number of dimensions:



Number of components:



Amount of separation:

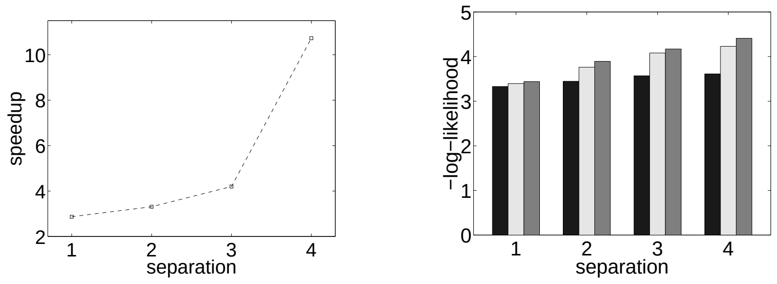
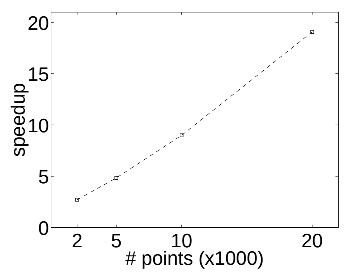


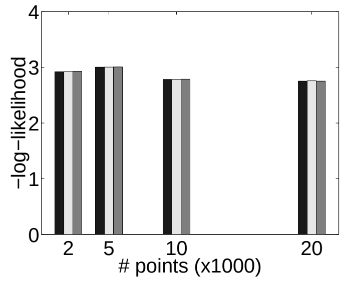
Figure 4.4: Experimental results comparing the regular and the chunky EM, showing the influence of different parameters on speedup (left) and negative log-likelihood on the test set (right) of the generating mixture (black), the converged mixture of regular EM (light), and chunky EM (dark).

methods.

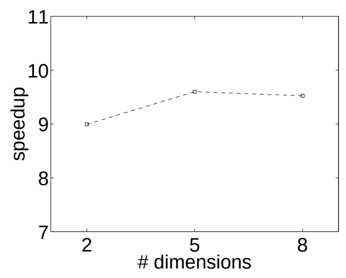
The lower speedup for high dimensional data sets can be ascribed to the use of a kd-tree. Data sets with more clusters and data sets with less component separation both cause more diverse responsibilities, thus making coarse partitions perform worse. This explains the loss of speedup for such data sets.

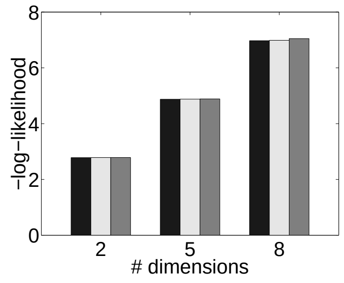
Number of data points (x1000):



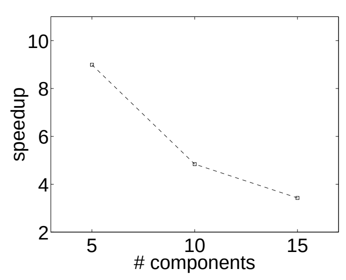


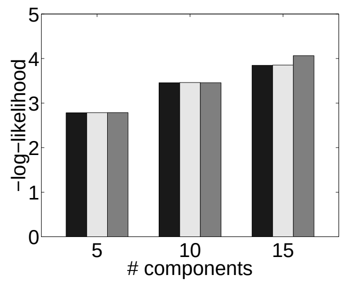
Number of dimensions:



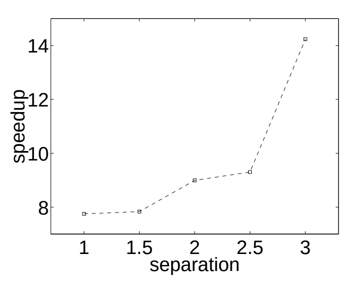


Number of components:





Amount of separation:



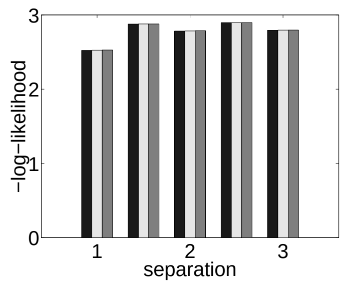


Figure 4.5: Experimental results comparing chunky greedy and non-chunky greedy EM. On the left is the speedup and on the right the negative log-likelihood on the test set, with the generating mixture (black), the converged mixture of regular greedy EM (light), and chunky greedy EM (dark).

Chapter 5

Conclusion

I have presented a variant of the Expectation-Maximisation algorithm specifically designed for dealing with large scale data sets, without paying much in terms of quality. The experiments described in the previous chapter clearly show that this algorithm is faster than the regular versions and that this speedup is linear with respect to the size of the data set. The experiments also show that there is barely any loss in quality even when working with large or complex data sets.

The theory also shows that the effect of the EM algorithm, namely the guaranteed monotone increase in model quality with respect to the free energy, still holds independent of the size of the partition. Therefore one can use the size of the partition as an easy to implement trade-off between speed and quality. This is useful when there are, for example, only limited resources available.

Future work could be done on applying this combination of geometric clustering and greedy model construction to other applications of the EM algorithm, such as, for example, the Generative Topographic Mapping [3].

Bibliography

[1] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. In *Proc. First Workshop High Performance Data Mining*, 1998.

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

[3] C. M. Bishop, M. Svensén, and C. K. I Williams. GTM: The generative topographic mapping. *Neural Computation*, 10:215–234, 1998.

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

[5] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[6] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions PAMI*, 24:881–892, 2002.

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

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

[9] A. Moore. Very fast em-based mixture model clustering using multiresolution kd-trees. In M. Kearns and D. Cohn, editors, *Advances in Neural Information Processing Systems*, pages 543–549. Morgan Kaufman, 1999.

[10] A. Moore and D. Pelleg. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. of 5th Int. Conf. on Knowledge Discovery and Data Mining*, pages 277–281, 1999.

[11] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer, 1998.

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

[13] J. J. Verbeek, N. Vlassis, and B. J. A. Kröse. Efficient greedy learning of gaussian mixture models. *Neural Computation*, 15(2):469–485, 2003.

[14] N. Vlassis and A. Likas. A greedy EM algorithm for Gaussian mixture learning. *Neural Processing Letters*, 15(1):77–87, February 2002.