Repairing Faulty Mixture Models using Density Estimation

Peter Sand PSAND@CS.CMU.EDU
Andrew W. Moore AWM@CS.CMU.EDU
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

Previous work in mixture model clustering has focused primarily on the issue of model selection. Model scoring functions (including penalized likelihood and Bayesian approximations) can guide a search of the model parameter and structure space. Relatively little research has addressed the issue of how to move through this space. Local optimization techniques, such as expectation maximization, solve only part of the problem; we still need to move between different local optima. The traditional approach, restarting the search from different random configurations, is inefficient. We describe a more directed and controlled way of moving between local maxima. Using multi-resolution kd-trees for fast density estimation, we search by modifying models within regions where they fail to predict the datapoint density. We compare this algorithm with a canonical clustering method, finding favorable results on a variety of large, low-dimensional datasets.

1. Introduction

Gaussian Mixture Models (GMMs) (Duda & Hart, 1973) are commonly used, both for discovering clusters in data and for summarizing multidimensional distributions, because of their statistical soundness and their ability to represent complex relationships in data.

Given a set of mixture probabilities p_i, where $p_i > 0$ and $\sum p_i = 1$, we can define the general form a mixture model distribution with K components:

$$f(x) = \sum_{i=1}^{K} p_i \phi_i(x)$$

In the case of Gaussian mixture models, x is a real-valued vector and each component $\phi_i(x)$ is a normal distribution with mean μ_i and covariance Σ_i. Learning a GMM means finding, in this space of structures and parameters, the model that best describes a dataset's distribution. The traditional solution is repeated local optimization from various initial configurations. We present a new approach that builds GMMs more efficiently by repairing faulty models, rather than throwing them out and starting over.

2. Previous work

2.1 Local GMM optimization

The foundation for most GMM clustering algorithms is Expectation Maximization (EM) (Dempster et al., 1977). EM provides iterative improvements to given GMM parameters (Σ_i, μ_i, and p_i), eventually converging to a local likelihood maximum. It is much faster than, for example, gradient descent, but the computational cost should not be taken lightly. Many iterations of EM are needed for convergence, and each iteration must touch each datapoint-Gaussian pair once, where each of those touches involves non-trivial computation.

Fortunately, recent work (Priebe, 1994; Bradley et al., 1999; Meng & van Dyk, 1997; Jamshidian & Jennrich, 1997; Moore, 1999) has shown that this can be accelerated by approaches such as one-pass incremental algorithms, controlled subsampling, and quasi-Newton accelerators. Many of these methods can be used in conjunction with our algorithm, but for the sake of a fair comparison with existing algorithms we will use standard, slow EM in the results (Section 4).

2.2 Global GMM optimization

EM solves only part of the clustering problem. The quality of the local maximum found by EM depends heavily on the positions of the clusters used to initialize EM (Bradley & Fayyad, 1998). We need some way to choose among the different local maxima found by EM.

2.2.1 Model Selection Criteria

The standard approach to global GMM optimization is to guide the model selection using a scoring function that allows comparisons between models with different values of K (number of clusters) and between different local maxima within a given value of K.

The likelihood of the data given the model is not an acceptable score function because it will continue to increase as we add more components; we need a criterion that favors concise models.

Recent work suggests that cross-validation is effective (Smyth, 2000). By building the model using a set of training data and evaluating the model using the likelihood of a separate set of testing data, we avoid the problem of building excessively complex models. Unfortunately, to do this in a robust manner, the cross-validation needs to use many partitions of the data, resulting in a computational cost much larger than that of other scoring methods.

Among the fastest scoring criteria are penalized likelihood functions. These compute the likelihood on a single training set and give a penalty for model complexity. One such method is the Akaike Information Criterion (AIC) (Akaike, 1974). This is a function of the log-likelihood of the data D given the model M and the number of model parameters M_P: $AIC(M) = \text{loglike}(D \mid M) - M_P$.[1]

Use of AIC scoring generally results in a good fit to the dataset's density distribution, but it often causes an overestimation of the number of components. To address this problem, the Bayesian Information Criterion (BIC) (Schwarz, 1978) provides a larger penalty based on the number of datapoints N:

$$BIC(M) = \text{loglike}(D \mid M) - \frac{M_P \log N}{2}$$

MCLUST-EM (Fraley & Raftery, 1998) uses BIC to choose between different values of K and between different parameterizations of the covariance model. For each covariance model and each value of K, MCLUST-EM uses hierarchical agglomeration (Fraley, 1998) to create a set of initial cluster centers, then runs EM to find a local maximum. MST-HMCLUST-EM (Posse, 1998) uses a minimum spanning tree to reducing the cost of the distance matrix used to perform hierarchical agglomeration.

The AutoClass algorithm (Cheeseman & Stutz, 1995) specifies a prior distribution for each model parameter. Since each prior multiplies the joint probability by a value less than 1, the AutoClass scoring function acts like a penalized likelihood. Rather than looping over values of K, AutoClass samples values of K from a log-normal distribution fit to scores for previous values of K.

The minimum message length (MML) (Wallace & Dowe, 2000) score function is defined as the number of bits required to encode both the model and the data given the model. Minimizing the length of this encoding is equivalent to maximizing the probability of the model given data.

2.2.2 Model Initialization and Modification

The algorithm we present in this paper can use any of the scoring functions described above. The choice of the best GMM selection methods for clustering and density estimation is a hotly debated topic (Keribin, 1997; Geiger et al., 1998), but is also an orthogonal issue to the topic of this paper—once you have a criterion, how do you search for a structure that scores well by the criterion?

The typical answer to this question (the approach used by AutoClass and others) is repeated trials based on randomly sampling data. Experiments have shown that other common methods for initializing EM, such as hierarchical agglomerative clustering, perform no better than the naive approach of random sampling (Meila & Heckerman, 1998).

A new method (Bradley & Fayyad, 1998), one that does perform better than the naive approach, is based on clustering of clusters. A number of small samples are taken to estimate locations of high datapoint density (potential cluster centers), which are then combined and smoothed using K-Means (Duda & Hart, 1973). This method increases the probability that EM converges to a good model, but it does not provide a way to proceed if the model is not good enough; the process must restart, rather than recycle the structure found by the previous optimization.

Reversible-jump Markov chain Monte Carlo (MCMC) (Green, 1995) methods have been used to improve GMMs by splitting and merging clusters (Williams, 2000). This technique is like ours in that it makes structural changes to improve faulty clusterings, rather than forcing the search to restart from a new initial model. While the MCMC approach provides a strong theoretical framework for mixture model search, our emphasis is computational efficiency.

The RefineModelAccuracy algorithm (Shanmugasundaram et al., 1999) has much in common with our work. The algorithm repeatedly adds clusters to regions in which a previous model underpredicts the

[1] AIC is sometimes defined as having the reverse sign of the formulation given here; the convention throughout this paper is that we seek to maximize, not minimize our score functions

data. We extend this approach by placing it within a hill-climbing search framework, where we add and remove clusters guided by an externally specified GMM evaluation criterion.

2.3 Clustering with other models

The focus of this paper is constructing Gaussian mixture models, but we will briefly mention a few other forms of clustering that share common elements with our approach.

Numerous algorithms have been developed that do clustering via combining and separating clusters. This occurs not just in the context of hierarchical clustering; for example, the X-MEANS algorithm (Pelleg & Moore, 2000) splits and merges groups of clusters as part of a hill-climbing search for a good value of K for K-MEANS clustering.

The BIRCH algorithm (Zhang et al., 1996) summarizes data in a compact, balanced tree with sufficient statistics to compute a variety of cluster and intercluster properties. BIRCH is very efficient on large and high-dimensional datasets, but the models it produces are quite different in nature than GMMs. A BIRCH model (like a K-MEANS model) does not represent complex distributions, such as those containing clusters within clusters or unusually shaped clusters (Gaussians with full covariance).

The CF-KERNEL (Zhang et al., 1999) method for density estimation uses a BIRCH tree structure to create what is effectively a Gaussian mixture model. Like other common kernel approaches, this uses spherical Gaussians, which can be advantageous in high-dimensions, but reduces model flexibility in low dimensions. Because the CF-kernel algorithm is aimed at the problem of density estimation, it does not address the issue of selecting the correct number of components for a concise representation of data clusters.

3. Clustering via Model Repair

The KD-CLUST algorithm performs clustering by recycling previously learned models. To conserve computation, we focus on repairing models only where they are damaged, rather than repeatedly rebuilding models from scratch.[2] The primary tool used for our model repair work is density estimation: we modify a model at regions where there is a large residual between the model's predicted density and the actual data density.

[2] Our motivation is similar to the idea behind boosting, used for supervised learning of numeric and symbolic functions (Freund & Schapire, 1996)

On large, low-dimensional datasets, we can perform this density estimation efficiently using multi-resolution kd-trees (see (Moore, 1999)). A kd-tree groups data spatially, using a hierarchy of hyper-rectangles. Each node of the tree includes a bounding box that specifies a subset of the data. The children of a kd-node are smaller bounding boxes, generated by splitting along the parent's widest axis.

3.1 Adding clusters

Given an existing model, we seek to add clusters to regions where the model underpredicts the actual number of datapoints. In a given region R, we compute the residual between $D_{\text{data}}(R)$, the density given by the data, and $D_{\text{model}}(R)$, the density predicted by the model. We wish to ignore regions with negative underprediction (i.e. overprediction), so we define underprediction in the following form:

$$U(R) = \max(D_{\text{data}}(R) - D_{\text{model}}(R), 0)$$

3.1.1 DENSITY ESTIMATION WITH kD-TREES

While underprediction is defined on any region in the dataspace, we consider only regions consisting of the hyper-rectangles of a kd-tree. This provides an efficient partitioning of a low-dimensional dataspace and allows value of $D_{\text{data}}(R)$ to be stored in nodes of the tree. For each node, while building the kd-tree, we compute the data density from $N(R)$, the number of points in the node's region, and $V(R)$, the volume of the node's region:[3]

$$D_{\text{data}}(R) = \frac{N(R)}{V(R)}$$

The use of hyper-rectangular regions also provides a simple way to estimate the predicted density, $D_{\text{model}}(R)$. For a model M, the predicted number of points in a region R is a function of the total number of datapoints, N, and the model's pdf:

$$D_{\text{model}}(R) = \frac{N \int_R \text{pdf}_M \, dR}{V(R)}$$

We use a pseudo-trapezoidal approximation to this integration by taking the average of the model's pdf at each corner of the hyper-rectangle and multiplying by the rectangle's volume. A d-dimensional hyper-rectangle has 2^d corners, so when clustering in high dimensions, a different integral approximation must be

[3] This biases $D_{\text{data}}(R)$ toward higher densities since the region defined by a kd-node is actually a bounding box.

used (for example averaging the values at the center of each face of the hyper-rectangle; this requires just $2d$ values).

This level of approximation will suffice because we are seeding model components that will later be optimized with EM. This allows us to use a crude model to guide the construction of a good model.

3.1.2 Creating a new cluster

Given the underprediction $U(R)$ for each region R, we can now decide where to place a new cluster. We could simply choose the region with the greatest underprediction, but, in order to consider several different locations for placing a new cluster, we choose one region at random, where the probability of selection is proportional to the region's underprediction.

We place a new Gaussian at the center of the selected region's hyper-rectangle. The Gaussian is given a spherical covariance, where the radius is based on the number of existing Gaussians (we create smaller Gaussians when the model already has a large number of Gaussians). We then stretch the sphere according to the width of the data in each dimension.

The mixture weight for the new cluster is tentatively assigned to be $\frac{1}{K}$, where K is the number of clusters (including the new cluster). The total mixture weight is then renormalized to equal 1.

After adding the cluster, we optimize all of the parameters (Gaussian center, covariance, and weight) using a sequence of EM steps. Thus we can maintain a high level of statistical robustness, despite our simplistic methods for determining the initial parameters of our new cluster.

3.1.3 Handling large datasets

The accuracy of the underprediction function $U(R)$ depends on the size of the regions we consider. If a region is too small, we may find that it contains so few datapoints that the data density $D_{\text{data}}(R)$ is dominated by noise. If a region is too large, we may find that the majority of a Gaussian lies within it, causing a large underestimation of the model density $D_{\text{model}}(R)$ because our integral approximation considers only the region's corners.

Given enough data, we can avoid both of these problems. To obtain regions that are neither too small nor too large, we consider only leaf nodes in a kd-tree which has been pruned such that each node has a certain minimal fraction of the datapoints. This method is robust in that the same pruning factor can be used for a wide range of datasets.

By pruning based on a fraction of the dataset size, we can traverse the tree in constant time with respect to the number of datapoints. This allows us to quickly estimate underprediction on very large datasets. The bottleneck then becomes the sequence of EM steps performed after every model modification. However, as mentioned in section 2.1, various methods exist for performing high-speed EM on large datasets.

3.2 Removing clusters

A GMM constructed for clustering (as opposed to a GMM built for density modelling) should concisely describe groupings of data. The importance of model simplicity is reflected in many of the scoring functions described in Section 2.2.1. Thus, in order to maximize our score, we provide a method of model pruning.

Our cluster removal criterion is based on the following intuitive reasoning: if EM determines that a cluster is unimportant, it will assign the cluster a small mixture weight (p_i). Thus, to prune a GMM, we remove one cluster, where the probability of removal is proportional to the inverse mixture weight. We then rescale the remaining mixture weights to sum to 1. This removal operation is quite fast because it does not consider any datapoints (it simply uses the existing set of clusters).

3.3 Structure Search

The KD-Clust algorithm combines these cluster addition and cluster removal operations using a hill-climbing search algorithm that seeks to maximize a given scoring function. The search moves through the GMM parameter and structure space by adding clusters, removing clusters, and performing sequences of EM steps. If a change to the model results in a lower score, we revert to the previous model and try a different change. This repeats until time expires.

A single iteration of the KD-Clust search can be summarized as follows:

1. We first choose whether to move towards the current local maximum or to jump to a different local maximum. With a probability of P_{jump}, we select the later. Otherwise, we skip to step 3.

2. If we choose to jump to a different local maximum, we add or remove one cluster. If we improved the model last time we added a cluster, we add again; otherwise we remove a cluster. (Likewise, if we improved the model last time we removed a cluster, we remove again; otherwise we add a cluster.)

3. We perform a sequence of EM iterations. Ideally we would run EM to convergence, but to save time, we run a small number of iterations (N_{iters}).
4. The resulting model is evaluated using a given score function. If the score is greater than the previous score, we keep the model; otherwise, we revert to the previous model.

Because the algorithm is given no initial knowledge of the correct number of clusters, we start with an empty model containing no clusters. In order to move efficiently toward a better model, we begin our search by making many quick changes to the model. As the search progresses, we move more slowly, spending more time using EM to refine the model.

To do this, we vary P_{jump} from 1 to 0 and N_{iters} from 1 to 10, both linearly as a function of time. We could use a simulated-annealing approach, where these parameters vary with an exponentially decaying temperature parameter, but instead we choose linear variation to reduce the algorithm's quantity of ad-hoc external parameters.

4. Results

4.1 Datasets

In order to evaluate our algorithm on a wide variety of GMMs, we use datasets generated by sampling from artificially constructed mixture models. We create a semi-random GMM by combining a set of K independently generated Gaussians, using uniform mixture weights: $p_i = \frac{1}{K}$. Each Gaussian has a mean within the unit square and a covariance matrix with diagonal elements between 0 and q (where q determines the expected width of the Gaussian) and non-diagonal elements that ensure semi-positive definiteness.

For this paper we consider three artificial datasets: D_1, generated from a set of 50 two-dimensional Gaussians (with $q = 0.0016$), D_2, generated from a set of 100 two-dimensional Gaussians (with $q = 0.0004$), and D_3, generated from a set of 50 three-dimensional Gaussians (with $q = 0.0004$).

Real-world clustering applications are abundant, ranging from web-page grouping to tissue segmentation. We consider three real-world datasets: D_4, which is generated from the density function implied by a protein gel experiment, D_5, which gives the spatial distribution of a set of galaxies within a region in the sky, and D_5, which is sampled from an image in a large astronomical dataset.

The datasets are summarized in Table 1. Datasets D_1 and D_4 are shown in Figure 7.

4.2 Comparison with AutoClass

We compare KD-Clust with AutoClass-C version 3.3.3. We choose AutoClass as our basis of comparison because it is well-known, fully-developed, and well-documented. Furthermore, AutoClass generates results that are said (Wallace & Dowe, 2000) to be comparable with the results generated by other GMM clustering algorithms.

Both KD-Clust and AutoClass were compiled from C source code with optimization enabled. The algorithms were configured to terminate each trial after 3000 seconds. Both algorithms were run on the same dedicated machine, where the OS and other process accounted for less than 1% of the clock time.

We configured KD-Clust and AutoClass to use GMMs with full covariance matrices. We set KD-Clust to use the log-likelihood of a holdout set (30% of the training set) for internal scoring. For the sake of a fair comparison, we used the standard EM algorithm for KD-Clust, the same as used by AutoClass.

The algorithms were scored using 10-fold likelihood cross-validation. A model was generated for each training set and the log-likelihood was computed for that model on the corresponding test set. We show a 95% confidence interval on the mean value of the score for each algorithm on each dataset.

The results are summarized in Table 2. For five of the six datasets, KD-Clust obtained a larger test-set likelihood than AutoClass. On the sixth dataset, the results were inconclusive. As a another point of comparison, we give training set BIC scores in Table 3.

As previously noted, the choice of a GMM evaluation criterion is widely debated. We believe that the use of cross-validated likelihood is a reasonable way to compare GMM algorithms because it avoids the com-

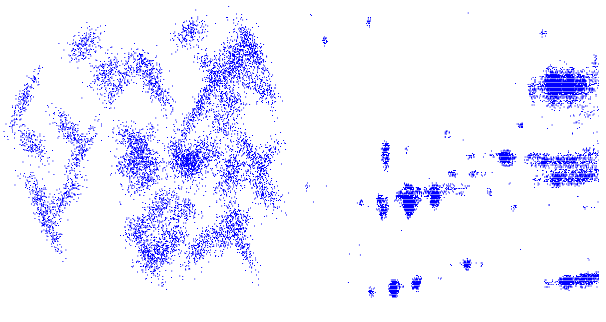

Figure 7. Datasets D_1 (left) and D_4, each shown with a sampling of 10,000 datapoints.

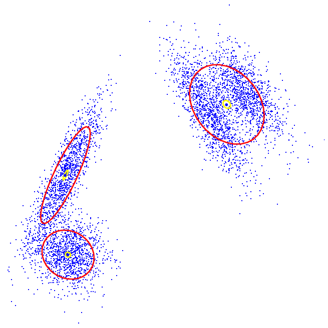

Figure 1: A dataset generated from a mixture of 4 Gaussians, shown with a clustering generated from 3 random cluster centers, after EM has converged to a local maximum.

Figure 2: A plot of the estimated density based on a kd-tree; the thin white lines correspond to gaps between leaf nodes of the kd-tree, i.e. regions between neighboring bounding boxes.

Figure 3: A plot of the density predicted by the model, using a kd-tree with approximate integration.

Figure 4: Underprediction is defined as the residual between the actual density (Figure 2) and predicted density (Figure 3). Note that most of the underprediction density is concentrated in the upper left where a single cluster in the model is trying to fit two actual clusters.

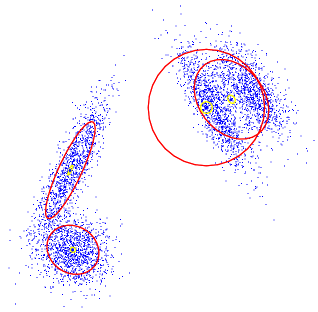

Figure 5: Using the underprediction function, we randomly select a location for a new cluster.

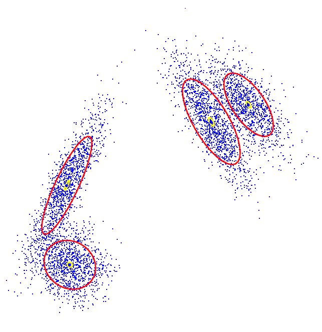

Figure 6: After a series of EM steps, the model fits the data much better than the 3-cluster model shown in Figure 1.

Table 1: Datasets

DSET	POINTS	DIMS
D_1	50000	2
D_2	50000	2
D_3	20000	3
D_4	50000	2
D_5	11195	2
D_6	50000	2

Table 2: Mean test-set log-likelihood

DSET	AUTOCLASS	KD-CLUST
D_1	7631 ± 161	7995 ± 44
D_2	8438 ± 682	9420 ± 88
D_3	10366 ± 495	11056 ± 141
D_4	13691 ± 135	14382 ± 112
D_5	-11297 ± 183	-9434 ± 29
D_6	21279 ± 325	21323 ± 134

Table 3: Mean training-set BIC

DSET	AUTOCLASS	KD-CLUST
D_1	74056 ± 669	78362 ± 310
D_2	78544 ± 5054	89707 ± 1039
D_3	89354 ± 7350	107403 ± 1775
D_4	137375 ± 1312	140965 ± 1274
D_5	-95501 ± 57	-96147 ± 156
D_6	211903 ± 418	212202 ± 229

plicated issues surrounding penalized likelihood and Bayesian approximations. Nonetheless, because each algorithm is trying to optimize it's own model selection criterion, the comparison is not strictly fair.

4.3 Model repair in action

This section provides an illustration of KD-CLUST's ability to repair models generated by other algorithms. We seek simply to provide a proof of concept, so we consider just one dataset, D_1. Half of the dataset is used as a training set and half is used as a test-set. From the training set we create a set of initial models using BIRCH, K-MEANS, AUTOCLASS, and KD-CLUST. We then run KD-CLUST for 1000 seconds on each of these initial models. We compute the test-set log-likelihood of each model to determine whether KD-CLUST has made an improvement.

To create initial models from AUTOCLASS and KD-CLUST, we use the same configuration as used in the previous experiments. For BIRCH and K-means, we need to perform some sort of model conversion.

While AUTOCLASS and KD-CLUST automatically determine the number of clusters, for simplicity, we instruct BIRCH to find exactly 50 clusters. We convert the BIRCH output to a GMM by placing a Gaussian at each BIRCH centroid. Mixture weights are assigned according to the fraction of datapoints that BIRCH assigned to each cluster. Each Gaussian is given a spherical covariance with a variance equal to the squared mean distance from the centroid to the cluster's datapoints. We use this statistic because it is provided directly by BIRCH; the GMM parameters (including the covariance size and shape) are subsequently optimized by running EM to convergence.

We run K-MEANS with $K = 50$ and convert the results to a GMM following the same steps used for the BIRCH model, including a sequence of EM iterations to put the model into a locally optimal state.

The results are given in Table 4. For BIRCH and K-MEANS, the initial score is computed after running EM to convergence.

5. Conclusion

Within this set of experiments, the KD-CLUST performs quite well. Unfortunately, on high-dimensional datasets, the performance suffers due to the computational limitations of kd-trees. We have, nonetheless, demonstrated the feasibility of this model repair approach to clustering, and this concept does not depend on kd-trees. Using different techniques for localizing the regions of disparity between a dataset and a mixture model, this model repair framework may be just as useful in higher dimensions.

In the future we intend to address this dimensionality issue. The problem of high-dimensional density estimation is difficult (Scott, 1992), but not devoid of potential solutions. Recent work suggests that random projection (Dasgupta, 2000) is an effective and efficient way to leverage low-dimensional clustering algorithms into higher dimensions. This and other methods of dimensionality reduction (such as (Talavera, 1999)) could allow our current kd-tree approach to be used on a much wider range of datasets. Another approach could be to replace kd-trees with a different mechanism for spatially representing data distribution, such as a CF-kernel tree (Zhang et al., 1999) or a generalized metric tree (for example (Moore, 2000)). A third solution could be to modify the smoothed subsampling approach (Bradley & Fayyad, 1998) so that instead of finding cluster centers, we find regions in need of model repair (perhaps by filtering the sample sets according to the model pdf in order to remove points already explained by the model).

Table 4: Test-set log-likelihood, before and after model repair.

INITIAL ALGORITHM	INITIAL SCORE	FINAL SCORE
BIRCH	39721	39746
K-MEANS	39808	39830
AUTOCLASS	39615	39807
KD-CLUST	39843	39882

References

Akaike, H. (1974). A new look at Statistical Model Identification. *IEEE Transactions on Automatic Control, 19.*

Bradley, P., & Fayyad, U. (1998). *Refining Initial Points for K-Means Clustering* (Technical Report). Microsoft Research.

Bradley, P., Fayyad, U., & Reina, C. (1999). Scaling clustering algorithms to large databases. *Proceedings Fourth International Conference on Knowledge Discovery and Data Mining.* AAAI Press.

Cheeseman, P., & Stutz, J. (1995). Bayesian Classification (AutoClass): Theory and Results. In U. Fayyad, G. Piatesky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining.* AAAI Press.

Dasgupta, S. (2000). Experiments with Random Projection. *Proceedings of the 16th conference on Uncertainty in Artificial Intelligence.* Morgan Kaufmann.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, *39*, 1–38.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* John Wiley & Sons.

Fraley, C. (1998). Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing.*

Fraley, C., & Raftery, A. E. (1998). *MCLUST: Software for Model-Based Cluster and Discriminant Analysis* (Technical Report). University of Washington.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. *Proceedings of the 13th International Conference on Machine Learning* (pp. 148–156). Morgan Kaufmann.

Geiger, D., Heckerman, D., King, H., & Meek, C. (1998). *Stratified Exponential Families: Graphical Models and Model Selection* (Technical Report). Microsoft Research.

Green, P. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrica*, *82*, 711–732.

Jamshidian, M., & Jennrich, R. I. (1997). Acceleration of the EM Algorithm by using Quasi-Newton Methods. *Journal of the Royal Statistical Society*, *59*, 569–587.

Keribin, C. (1997). *Consistent Estimation of the Order of Mixture Models* (Technical Report). Université d'Evry-Val d'Essonne.

Meila, M., & Heckerman, D. (1998). *An Experimental Comparison of Several Clustering and Initialization Methods* (Technical Report). Microsoft Research.

Meng, X., & van Dyk, D. (1997). The EM Algorithm—An Old Folk-Song Sung to a Fast New Tune. *Journal of the Royal Statistical Society*, *59*, 511–567.

Moore, A. W. (1999). Very fast mixture-model-based clustering using multiresolution kd-trees. *Advances in Neural Information Processing Systems 10* (pp. 543–549). San Francisco: Morgan Kaufmann.

Moore, A. W. (2000). The Anchors Hierarchy: Using the triangle inequality to survive high dimensional data. *Twelfth Conference on Uncertainty in Artificial Intelligence.* AAAI Press.

Pelleg, D., & Moore, A. W. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. *Proceedings of the Seventeenth International Conference on Machine Learning.* San Francisco: Morgan Kaufmann.

Posse, C. (1998). *Hierarchical Model-Based Clustering for Large Datasets* (Technical Report). University of Minnesota.

Priebe, C. (1994). Adaptive Mixtures. *Journal of the American Statistical Association*, *89*, 796–806.

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, *6*, 461–464.

Scott, D. W. (1992). *Multivariate Density Estimation.* Wiley.

Shanmugasundaram, J., Fayyad, U., & Bradley, P. (1999). Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions. *Proceedings Fifth International Conference on Knowledge Discovery and Data Mining.* ACM.

Smyth, P. (2000). Model Selection for Probabilistic Clustering Using Cross-validated Likelihood. *Statistics and Computing*, *10*, 63–72.

Talavera, L. (1999). Feature Selection as a Preprocessing Step for Hierarchical Clustering. *Proceedings of the 16th International Conference on Machine Learning* (pp. 389–397). Morgan Kaufmann.

Wallace, C. S., & Dowe, D. L. (2000). MML Clustering of Multi-state Poisson, von Mises Circular and Gaussian distributions. *Statistics and Computing*, *10*, 73–83.

Williams, C. K. I. (2000). A MCMC approach to Hierarchical Mixture Modelling. *Advances in Neural Information Processing Systems 12* (pp. 680–686). Morgan Kaufmann.

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems : PODS 1996.* ACM.

Zhang, T., Ramakrishnan, R., & Livny, M. (1999). Fast Density Estimation Using CF-kernel for Very Large Databases. *Proceedings Fifth International Conference on Knowledge Discovery and Data Mining.* ACM.