# Very Fast EM-based Mixture Model Clustering using Multiresolution kd-trees

Andrew Moore
Robotics Institute and School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, (412) 268-7599, awm@cs.cmu.edu

**Abstract**

Clustering is important in many fields including manufacturing, biology, finance, and astronomy. Mixture models are a popular approach due to their statistical foundations, and EM is a very popular method for finding mixture models. EM, however, requires many accesses of the data, and thus has been dismissed as impractical (e.g. (Zhang, Ramakrishnan, & Livny, 1996)) for data mining of enormous datasets. We present a new algorithm, based on the multiresolution *k*d-trees of (Moore, Schneider, & Deng, 1997), which dramatically reduces the cost of EM-based clustering, with savings rising linearly with the number of datapoints. Although presented here for maximum likelihood estimation of Gaussian mixture models, it is also applicable to non-Gaussian models (provided class densities are monotonic in Mahalanobis distance), mixed categorical/numeric clusters, and Bayesian methods such as Autoclass (Cheeseman & Oldford, 1994).

## 1 Learning Mixture Models

In a Gaussian mixture model (e.g. (Duda & Hart, 1973)), we assume that datapoints $\{\mathbf{x}_1 \ldots \mathbf{x}_R\}$ have been generated independently by the following process. For each $\mathbf{x}_i$ in turn, nature begins by randomly picking a class,

$c_j$, from a discrete set of classes $\{c_1 \ldots c_N\}$. Then nature draws $\mathbf{x}_i$ from an $M$-dimensional Gaussian whose mean $\mu_j$ and covariance $\Sigma_j$ depend on the class. Thus we have

$$P(\mathbf{x}_i \mid c_j, \boldsymbol{\theta}) \sim (2\pi\|\Sigma_j\|)^{-1/2} \exp(-\frac{1}{2}(\mathbf{x}_i - \mu_j)^T {\Sigma_j}^{-1}(\mathbf{x}_i - \mu_j)) \tag{1}$$

where $\boldsymbol{\theta}$ denotes all the parameters of the mixture: the class probabilities $p_j$ (where $p_j = P(c_j \mid \boldsymbol{\theta})$), the class centers $\mu_j$ and the class covariances $\Sigma_j$.

The job of a mixture model learner is to find a good estimate of the model, and Expectation Maximization (EM), also known as "Fuzzy $k$-means", is a popular algorithm for doing so. The $t$th iteration of EM begins with an estimate $\boldsymbol{\theta}^{\mathrm{t}}$ of the model, and ends with an improved estimate $\boldsymbol{\theta}^{\mathrm{t+1}}$. Write

$$\boldsymbol{\theta}^{\mathrm{t}} = (p_1, \ldots p_N, \mu_1, \ldots \mu_N, \Sigma_1, \ldots, \Sigma_N) \tag{2}$$

EM iterates over each point-class combination, computing for each class $c_j$ and each datapoint $\mathbf{x}_i$, the extent to which $\mathbf{x}_i$ is "owned" by $c_j$. The ownership is simply $w_{ij} = P(c_j \mid \mathbf{x}_i, \boldsymbol{\theta}^{\mathrm{t}})$. Throughout this paper we will use the following notation:

$$\begin{aligned} a_{ij} &= P(\mathbf{x}_i \mid c_j, \boldsymbol{\theta}^{\mathrm{t}}) \\ w_{ij} &= P(c_j \mid \mathbf{x}_i, \boldsymbol{\theta}^{\mathrm{t}}) = a_{ij} p_j / \textstyle\sum_{k=1}^{N} a_{ik} p_k (\text{by Bayes' Rule}) \end{aligned}$$

Then the new value of the centroid, $\mu_j$, of the $j$th class in the new model $\boldsymbol{\theta}^{\mathrm{t+1}}$ is simply the weighted mean of all the datapoints, using the values $\{w_{1j}, w_{2j}, \ldots w_{Rj}\}$ as the weights. A similar weighted procedure gives the new estimates of the class probabilities and the class covariances:

$$p_j \leftarrow \frac{\mathrm{sw}_j}{R} \quad , \quad \mu_j \leftarrow \frac{1}{\mathrm{sw}_j} \sum_{i=1}^{R} w_{ij} \mathbf{x}_i \quad , \quad \Sigma_j \leftarrow \frac{1}{\mathrm{sw}_j} \sum_{i=1}^{R} w_{ij} (\mathbf{x}_i - \mu_j)(\mathbf{x}_i - \mu_j)^T \tag{3}$$

where $\mathrm{sw}_j = \sum_{i=1}^{R} w_{ij}$. Thus each iteration of EM visits every datapoint-class pair, meaning $NR$ evaluations of a $M$-dimensional Gaussian, and so needing $O(M^2 NR)$ arithmetic operations per iteration. This paper aims to reduce that cost.

An *mrk*d-tree (Multiresolution KD-tree), introduced in (Deng & Moore, 1995) and developed further in (Moore et al., 1997), is a binary tree in which each node is associated with a subset of the datapoints. The root node owns all the datapoints. Each non-leaf-node has two children, defined by a splitting dimension Nd.splitdim and a splitting value Nd.splitval. The two children divide their parent's datapoints between them, with the left child owing those datapoints that are strictly less than the splitting value in the splitting dimension, and the right child owning the remainder of the parent's datapoints:

$$\mathbf{x}_i \in \text{Nd.left} \quad \Leftrightarrow \quad \mathbf{x}_i[\text{Nd.splitdim}] < \text{Nd.splitval} \text{ and } \mathbf{x}_i \in \text{Nd} \tag{4}$$

$$\mathbf{x}_i \in \text{Nd.right} \quad \Leftrightarrow \quad \mathbf{x}_i[\text{Nd.splitdim}] \geq \text{Nd.splitval} \text{ and } \mathbf{x}_i \in \text{Nd} \tag{5}$$

The distinguishing feature of *mrk*d-trees is that their nodes contain the following:

- Nd.numpoints: The number of points owned by Nd (equivalently, the average density in Nd).
- Nd.centroid: The centroid of the points owned by Nd (equivalently, the first moment of the density below Nd).
- Nd.cov: The covariance of the points owned by Nd (equivalently, the second moment of the density below Nd).
- Nd.hyperrect: The bounding hyper-rectangle of the points below Nd

We construct *mrk*d-trees top-down, identifying the bounding box of the current node, and splitting in the center of the widest dimension. A node is declared to be a leaf, and is left unsplit, if the widest dimension of its bounding box is $\leq$ some threshold, *MBW*. If *MBW* is zero, then all leaf nodes denote singleton or coincident points, the tree has $O(R)$ nodes and so requires $O(M^2R)$ memory, and (with some care) the construction cost is $O(M^2R + MR\log R)$. In practice, we set *MBW* to 1% of the range of the datapoint components. The tree size and construction thus cost considerably less than these bounds because in dense regions, tiny leaf nodes were able to summarize dozens of datapoints. Note too that the cost of tree-building is amortized—the tree must be built once, yet EM performs many iterations.

To perform an iteration of EM with the *mrk*d-tree, we call the function MakeStats (described below) on the root of the tree. MakeStats(Nd, $\boldsymbol{\theta}^t$)

outputs $3N$ values: $(\text{SW}_1, \text{SW}_2, \ldots \text{SW}_N, \text{SWX}_1, \ldots \text{SWX}_N, \text{SWXX}_1, \ldots \text{SWXX}_N)$ where

$$\text{SW}_j = \sum_{\mathbf{x}_i \in \text{ND}} w_{ij} \quad , \quad \text{SWX}_j = \sum_{\mathbf{x}_i \in \text{ND}} w_{ij}\mathbf{x}_i \quad , \quad \text{SWXX}_j = \sum_{\mathbf{x}_i \in \text{ND}} w_{ij}\mathbf{x}_i\mathbf{x}_i^T \tag{6}$$

The results of MAKESTATS(ROOT) provide sufficient statistics to construct $\boldsymbol{\theta}^{\mathrm{t}+1}$:

$$p_j \leftarrow \text{SW}_j/R \quad , \quad \mu_j \leftarrow \text{SWX}_j/\text{SW}_j \quad , \quad \Sigma_j \leftarrow (\text{SWXX}_j/\text{SW}_j) - \mu_j\mu_j^T \tag{7}$$

If MAKESTATS is called on a leaf node, we simply compute, for each $j$,

$$\bar{w}_j = P(c_j \mid \bar{\mathbf{x}}, \boldsymbol{\theta}^{\mathrm{t}}) = P(\bar{\mathbf{x}} \mid c_j, \boldsymbol{\theta}^{\mathrm{t}})P(c_j \mid \boldsymbol{\theta}^{\mathrm{t}}) / \sum_{k=1}^{N} P(\bar{\mathbf{x}} \mid c_k, \boldsymbol{\theta}^{\mathrm{t}})P(c_k \mid \boldsymbol{\theta}^{\mathrm{t}}) \tag{8}$$

where $\bar{\mathbf{x}} = \text{ND.CENTROID}$, and where all the items in the right hand equation are easily computed. We then return $\text{SW}_j = \bar{w}_j \times \text{ND.NUMPOINTS}$, $\text{SWX}_j = \bar{w}_j \times \text{ND.NUMPOINTS} \times \bar{\mathbf{x}}$ and $\text{SWXX}_j = \bar{w}_j \times \text{ND.NUMPOINTS} \times \text{ND.COV}$. The reason we can do this is that, if the leaf node is very small, there will be little variation in $w_{ij}$ for the points owned by the node and so, for example $\sum w_{ij}\mathbf{x}_i \approx \bar{w}_j \sum \mathbf{x}_i$. In the experiments below we use very tiny leaf nodes, ensuring accuracy.

If MAKESTATS is called on a non-leaf-node, it can easily compute its answer by recursively calling MAKESTATS on its two children and then returning the sum of the two sets of answers. In general, that is exactly how we will proceed. If that was the end of the story, we would have little computational improvement over conventional EM, because one pass would fully traverse the tree, which contains $O(R)$ nodes, doing $O(NM^2)$ work per node.

We will win if we ever spot that, at some intermediate node, we can *prune*, i.e. evaluate the node as if it were a leaf, without searching its descendents, but without introducing significant error into the computation.

To do this, we will compute, for each $j$, the minimum and maximum $w_{ij}$ that any point inside the node could have. This procedure is more complex than in the case of locally weighted regression (Moore et al., 1997).

We wish to compute $w_j^{\min}$ and $w_j^{\max}$ for each $j$, where $w_j^{\min}$ is a lower bound on $\min_{\mathbf{x}_i \in \text{ND}} w_{ij}$ and $w_j^{\max}$ is an upper bound on $\max_{\mathbf{x}_i \in \text{ND}} w_{ij}$. This is hard because $w_j^{\min}$ is determined not only by the mean and covariance of the $j$th class but also the other classes. For example, in Figure 1, $w_{32}$ is approximately 0.5, but it would be much larger if $c_1$ were further to the left, or had a thinner covariance.

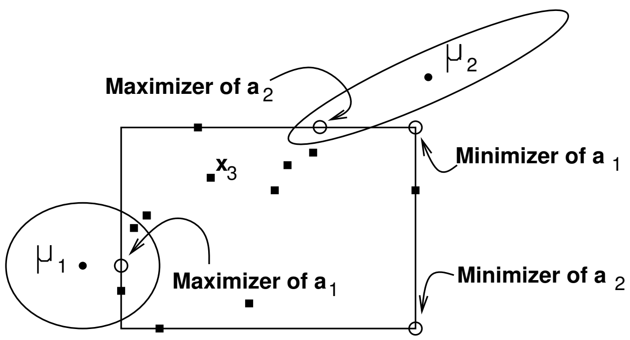



Figure 1: The rectangle denotes a hyperrectangle in the mrkd-tree. The small squares denote datapoints "owned" by the node. Suppose there are just two classes, with the given means, and covariances depicted by the ellipses. Small circles indicate the locations within the node for which $a_j$ (i.e. $P(x \mid c_j)$) would be extremized.

But remember that the $w_{ij}$'s are defined in terms of $a_{ij}$'s, thus: $w_{ij} = a_{ij}p_j / \sum_{k=1}^{N} a_{ik}p_k$. We *can* put bounds on the $a_{ij}$'s relatively easily. It simply requires that for each $j$ we compute[1] the closest and furthest point from $\mu_j$ within ND.HYPERRECT, using the Mahalanobis distance $MHD(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^T \Sigma_j^{-1} (\mathbf{x} - \mathbf{x}')$. Call these shortest and furthest squared distances $MHD^{\min}$ and $MHD^{\max}$. Then

$$a_j^{\min} = (2\pi \|\Sigma_j\|)^{-1/2} \exp(-\frac{1}{2} MHD^{\max}) \tag{9}$$

is a lower bound for $\min_{\mathbf{x}_i \in \mathrm{ND}} a_{ij}$, with a similar definition of $a_j^{\max}$. Then write

$$\begin{aligned} \min_{\mathbf{x}_i \in \mathrm{ND}} w_{ij} &= \min_{\mathbf{x}_i \in \mathrm{ND}} (a_{ij}p_j / \sum_k a_{ik}p_k) = \min_{\mathbf{x}_i \in \mathrm{ND}} (a_{ij}p_j / (a_{ij}p_j + \sum_{k \neq j} a_{ik}p_k)) \\ &\geq a_j^{\min} p_j / (a_j^{\min} p_j + \sum_{k \neq j} a_k^{\max} p_k) = w_j^{\min} \end{aligned}$$

where $w_j^{\min}$ is our lower bound. There is a similar definition for $w_j^{\max}$. The inequality is proved by elementary algebra, and requires that all quantities are positive (which they are).

We can often tighten the bounds further using a procedure that exploits the fact that $\sum_j w_{ij} = 1$, but space does not permit further discussion.

[1] Computing these points requires non-trivial computational geometry because the covariance matrices are not necessarily axis-aligned. There is no space here for details.

We will prune if $w_j^{\min}$ and $w_j^{\max}$ are close for all $j$. What should be the criterion for closeness? The first idea that springs to mind is: Prune if $\forall j \,.\, (w_j^{\max} - w_j^{\min} < \epsilon)$. But such a simple criterion is not suitable: some classes may be accumulating very large sums of weights, whilst others may be accumulating very small sums. The large-sum-weight classes can tolerate far looser bounds than the small-sum-weight classes. Here, then, is a more satisfactory pruning criterion: Prune if $\forall j \,.\, (w_j^{\max} - w_j^{\min} < \tau w_j^{\text{total}})$ where $w_j^{\text{total}}$ is the total weight awarded to class $j$ over the entire dataset, and $\tau$ is some small constant. Sadly, $w_j^{\text{total}}$ is not known in advance, but happily we can find a lower bound on $w_j^{\text{total}}$ of $w_j^{\text{sofar}} + \textsc{Nd.numpoints} \times w_j^{\min}$, where $w_j^{\text{sofar}}$ is the total weight awarded to class $j$ so far during the search over the $k$d-tree.

The algorithm as described so far performs divide-and-conquer-with-cutoffs on the set of datapoints. In addition, it is possible to achieve an extra acceleration by means of divide and conquer on the class centers. Suppose there were $N = 100$ classes. Instead of considering all 100 classes at all nodes, it is frequently possible to determine at some node that the maximum possible weight $w_j^{\max}$ for some class $j$ is less than a miniscule fraction of the minimum possible weight $w_k^{\min}$ for some other class $k$. Thus if we ever find that in some node $w_j^{\max} < \lambda w_k^{\min}$ where $\lambda = 10^{-4}$, then class $c_j$ is removed from consideration from all descendents of the current node. Frequently this means that near the tree's leaves, only a tiny fraction of the classes compete for ownership of the datapoints, and this leads to large time savings.

# 2 Results

We have subjected this approach to numerous Monte-Carlo empirical tests. Here we report on one set of such tests, created with the following methodology.

- We randomly generate a mixture of Gaussians in $M$-dimensional space (by default $M = 2$). The number of Gaussians, $N$ is, by default, 20. Each Gaussian has a mean lying within the unit hypercube, and a covariance matrix randomly generated with diagonal elements between 0 up to $4\sigma^2$ (by default, $\sigma = 0.05$) and random non-diagonal elements that ensure symmetric positive definiteness. Thus the distance from a

Gaussian center to its 1-standard-deviation contour is of the order of magnitude of $\sigma$.

- We randomly generate a dataset from the mixture model. The number of points, $R$, is (by default) 160,000. Figure 2 shows a typical generated set of Gaussians and datapoints.
- We then build an *mrk*d-tree for the dataset, and record the memory requirements and real time to build (on a Pentium 200Mhz, in seconds).
- We then run EM on the data. EM begins with an entirely different set of Gaussians, randomly generated using the same procedure.
- We run 5 iterations of the conventional EM algorithm and the new *mrk*d-tree-based algorithm. The new algorithm uses a default value of 0.1 for $\tau$. We record the real time (in seconds) for each iteration of each algorithm, and we also record the mean log-likelihood score $(1/R)\sum_{i=1}^{R} \log P(\mathbf{x}_i \mid \boldsymbol{\theta}^t)$ for the $t$th model for both algorithms.

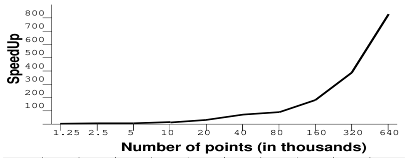
Figure 3 shows the nodes that are visited during Iteration 2 of the Fast EM with $N = 6$ classes. Table 1 shows the detailed results as the experimental parameters are varied. Speedups vary from 8-fold to 1000-fold. There are 100-fold speedups even with very wide (non-local) Gaussians. In other experiments, similar results were also obtained on real datasets that disobey the Gaussian assumption. There too, we find one- and two-order-of-magnitude computational advantages with indistinguishable statistical behavior (no better and no worse) compared with conventional EM.

# 3 Conclusion

Little difficulty is expected in extending this approach to datasets with mixed categorical and numeric attributes.

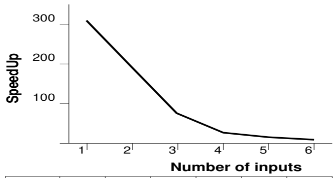
The use of variable resolution structures for clustering has been suggested in many places (e.g. (Omohundro, 1987, 1991; Ester, Kriegel, & Xu, 1995; Zhang et al., 1996)). The BIRCH system, in particular, is popular in the database community. BIRCH is, however, unable to identify second-moment features of clusters (such as non-axis-aligned spread). Our contributions

Effect of **Number of Datapoints,** $R$:
As $R$ increases so does the computational advantage, essentially linearly. The tree-build time (11 seconds at worst) is a tiny cost compared with even just one iteration of Regular EM (2385 seconds, on the big dataset.) In the rightmost experiment, LogLikelihood went from 1.112 to 1.526 during 5 EM iterations of Fast Method. The slow method's final LogLikelihood differed by 0.009. **FinalSlowSecs:** 2385. **FinalFastSecs:** 3.



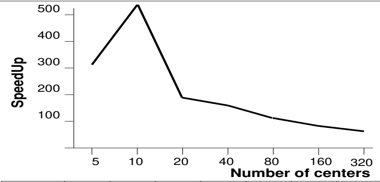

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Build | 0 | 1 | 1 | 1 | 1 | 1 | 2 | 5 | 7 | 11 |
| Nodes | 1847 | 3149 | 5161 | 7891 | 10827 | 13997 | 19223 | 22737 | 26475 | 29659 |

Effect of **Number of Dimensions,** $M$:
As with many *k*d-tree algorithms, the benefits decline as dimensionality increases, yet even in 6 dimensions, there is an 8-fold advantage. In the rightmost experiment, LogLikelihood went from -10.2 to 9.54 during 5 EM iterations of Fast Method. The slow method's final LogLikelihood differed by 0.08. **FinalSlowSecs:** 2742. **FinalFastSecs:** 310.25.



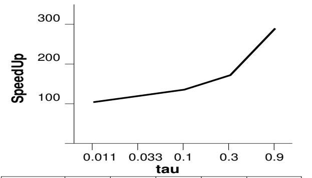

| | | | | | | |
|---|---|---|---|---|---|---|
| Build | 2 | 5 | 7 | 12 | 20 | 35 |
| Nodes | 1109 | 22737 | 39367 | 72647 | 122367 | 185609 |

Effect of **Number of Classes,** $N$:
Conventional EM slows down linearly with the number of classes. Fast EM is clearly sublinear, with a 70-fold speedup even with 320 classes. Note how the tree size grows. This is because more classes mean a more uniform data distribution and fewer datapoints "sharing" tree leaves. In the rightmost experiment, LogLikelihood went from 0.692 to 0.768 during 5 EM iterations of Fast Method. The slow method's final LogLikelihood differed by 0.004. **FinalSlowSecs:** 9278. **FinalFastSecs:** 143.3.



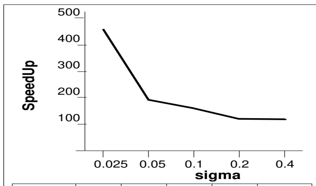

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Build | 5 | 5 | 5 | 5 | 6 | 6 | 6 |
| Nodes | 10643 | 17017 | 22737 | 29797 | 31427 | 41627 | 45485 |

Effect of **Tau,** $\tau$:
The larger $\tau$, the more willing we are to prune during the tree search, and thus the faster we search, but the less accurately we mirror EM's statistical behavior. Indeed when $\tau$ is large, the discrepancy in the log likelihood is relatively large. In the rightmost experiment, LogLikelihood went from 1.222 to 1.542 during 5 EM iterations of Fast Method. The slow method's final LogLikelihood differed by 0.06. **FinalSlowSecs:** 584.5. **FinalFastSecs:** 2.




| | | | | | |
|---|---|---|---|---|---|
| Build | 5 | 5 | 5 | 5 | 5 |
| Nodes | 22737 | 22737 | 22737 | 22737 | 22737 |

Effect of **Standard Deviation,** $\sigma$:
Even with very wide Gaussians, with wide support, we still get large savings. The nodes that are pruned in these cases are rarely nodes with one class owning all the probability, but instead are nodes where all classes have non-zero, but little varying, probability. In the rightmost experiment, LogLikelihood went from -1.072 to -0.593 during 5 EM iterations of Fast Method. The slow method's final LogLikelihood differed by 0.003. **FinalSlowSecs:** 583. **FinalFastSecs:** 4.75.




| | | | | | |
|---|---|---|---|---|---|
| Build | 4 | 5 | 8 | 9 | 11 |
| Nodes | 10581 | 22737 | 41289 | 76407 | 128235 |

Table 1: In all the above results all parameters were held at their default values except for one, which varied as shown in the graphs. Each graph shows the factor by which the new EM is faster than the conventional EM. Below each graph is the time to build the *mrk*d-tree in seconds and the number of nodes in the tree. Note that although the tree building cost is not included in the speedup calculation, it is negligible in all cases, especially considering that only one tree build is needed for all EM iterations. The bottom left of each block gives more information about the rightmost experiment in each graph. It shows the absolute time for the final EM iteration of each algorithm. It also indicates that, in general, there is little difference between the models found by the slow and fast EMs. The difference in log-likelihood of the model after the 5th iteration is always small compared with the change in log-likelihood during the first 5 EM iterations.

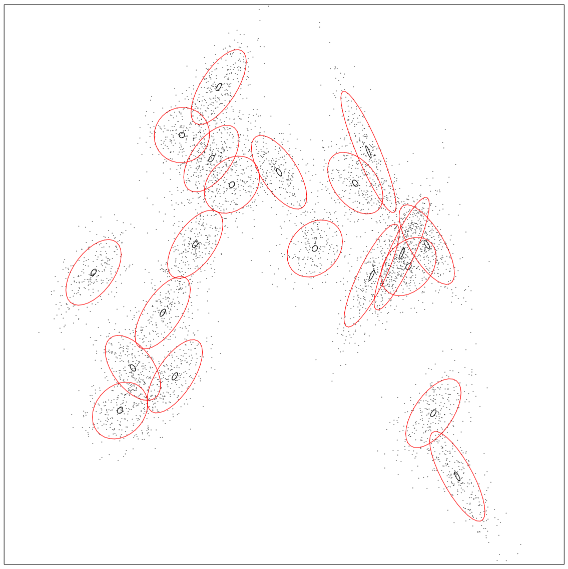


Figure 2: A typical set of Gaussians generated by our random procedure. They in turn generate the datasets upon which we compare the performance of the old and new implementations of EM.

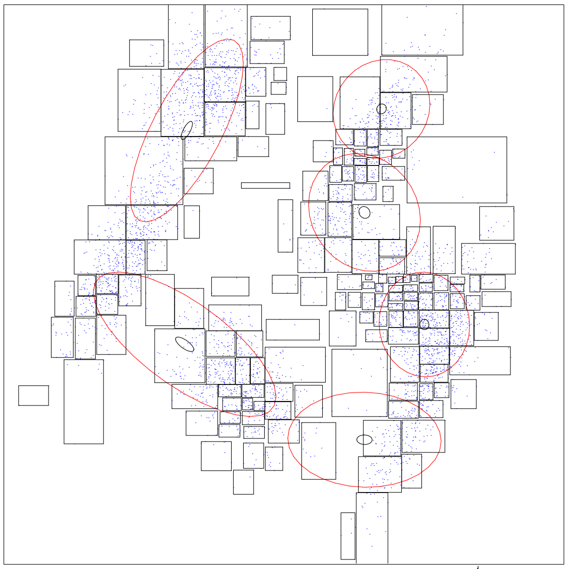


Figure 3: The ellipses show the model $\boldsymbol{\theta}^{t}$ at the *start* of an EM iteration. The rectangles depict the *mrk*d-tree nodes that were pruned. Observe larger rectangles (and larger savings) in areas with less variation in class probabilities. Note this is not merely an algorithm that can only prune where the data density is low.

have been the use of a multi-resolution approach, with associated computational benefits, and the introduction of an efficient algorithm that leaves the statistical aspects of mixture model estimation unchanged. The growth of recent data mining algorihms that are *not* based on statistical foundations has freqently been justified by the following statement: Using state-of-the-art statistical techniques is too expensive because such techniques were not designed to handle large datasets and become intractable with millions of datapoints. In earlier work we provided evidence that this statement may not apply for locally weighted regression (Moore et al., 1997) or Bayesian network learning (Moore & Lee, 1998), and we hope this paper provides some evidence that it also needn't apply to clustering.

# References


Cheeseman, P., & Oldford, R. (1994). *Selecting Models from Data: Artificial Intelligence and Statistics IV. Lecture Notes in Statistics, vol. 89.* Springer Verlag.

Deng, K., & Moore, A. W. (1995). Multiresolution Instance-based Learning. In *Proceedings of IJCAI-95*. Morgan Kaufmann.

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

Ester, M., Kriegel, H. P., & Xu, X. (1995). A Database Interface for Clustering in Large Spatial Databases. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*. AAAI Press.

Moore, A. W., Schneider, J., & Deng, K. (1997). Efficient Locally Weighted Polynomial Regression Predictions. In D. Fisher (Ed.), *Proceedings of the 1997 International Machine Learning Conference*. Morgan Kaufmann.

Moore, A. W., & Lee, M. S. (1998). Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research*, *8*.

Omohundro, S. M. (1987). Efficient Algorithms with Neural Network Behaviour. *Journal of Complex Systems*, *1*(2), 273–347.

Omohundro, S. M. (1991). Bumptrees for Efficient Function, Constraint, and Classification Learning. In Lippmann, R. P., Moody, J. E., & Touretzky, D. S. (Eds.), *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann.

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