Machine Learning, 42, 9–29, 2001

An Experimental Comparison of Model-Based Clustering Methods

MARINA MEILĂ*
DAVID HECKERMAN heckerma@microsoft.com
Microsoft Research, Redmond, WA 98052, USA

Editor: Douglas Fisher

Abstract. We compare the three basic algorithms for model-based clustering on high-dimensional discrete-variable datasets. All three algorithms use the same underlying model: a naive-Bayes model with a hidden root node, also known as a multinomial-mixture model. In the first part of the paper, we perform an experimental comparison between three batch algorithms that learn the parameters of this model: the Expectation–Maximization (EM) algorithm, a "winner take all" version of the EM algorithm reminiscent of the K-means algorithm, and model-based agglomerative clustering. We find that the EM algorithm significantly outperforms the other methods, and proceed to investigate the effect of various initialization methods on the final solution produced by the EM algorithm. The initializations that we consider are (1) parameters sampled from an uninformative prior, (2) random perturbations of the marginal distribution of the data, and (3) the output of agglomerative clustering. Although the methods are substantially different, they lead to learned models that are similar in quality.

Keywords: clustering, model-based clustering, naive-Bayes model, multinomial-mixture model, EM algorithm, agglomerative clustering, initialization

1. Introduction

A host of algorithms for clustering data have been published (Duda & Hart, 1973; Jain & Dubes, 1988). Of these algorithms, most find clusters by optimizing some criterion that depends on a distance between case pairs or between pairs of centroids of case collections. In this paper, we focus on *model-based* clustering—that is, those algorithms that postulate a *generative statistical model* for the data—and then use a likelihood (or posterior probability) derived from this model as the criterion to be optimized. Model-based clustering has recently gained widespread use both for continuous and discrete domains (Banfield & Raftery, 1993; Clogg, 1995) mainly due to the fact that it allows one to identify clusters based on their shape and structure rather than on proximity between data points. We also stress that we examine only *flat*—as opposed to *hierarchical*—clustering. Although one of the methods presented below is able to output a cluster hierarchy, the hierarchical structure is present only implicitly in the learning algorithm, being ignored in the final model.

In this paper, we test model-based clustering algorithms on a basic but widely used model: the *naive-Bayes model* (NBM) with a hidden root node. In statistical terms, this model is

*Present address: University of Washington, Department of Statistics, Box 354322, Seattle, WA 98195, USA; e-mail: mmp@stat.washington.edu.

a mixture of multinomial distributions under the assumption that the hidden class variable renders all observations mutually independent (e.g., Clogg, 1995; Cheeseman & Stutz, 1995). The conceptual simplicity of the naive-Bayes model and its ease of implementation make it an ideal prototype and benchmark model. Moreover, in higher dimensions, where both the search for model structure and escaping local optima in the parameter space can substantially slow computation, the relatively small number of parameters of the naive-Bayes model make it more appealing than other complex models.

Nonetheless, even for this relatively simple model, high dimensional domains (tens to hundreds of variables) can present a challenge for both structure and parameter search algorithms. The aim of our work is to study the behavior of the commonly used algorithms for learning the parameters of mixture models on high-dimensional data. Additional attention will be given to the initialization issue: How important is the choice of the initial parameters of an iterative algorithm and how do we find a good set of initial parameters?

In Section 2, we introduce the clustering model and the learning algorithms that we shall compare. All algorithms are batch algorithms as opposed to on-line ones. They are Expectation-Maximization (EM), Classification EM (CEM)—a "winner take all" version of the EM algorithm reminiscent of the K-means algorithm—and agglomerative clustering (AC). In Sections 3 and 4, we describe our datasets and experimental procedure, respectively. In Section 5, we compare the three learning algorithms. The experiments suggest that the EM algorithm is the best method under a variety of performance measures. Consequently, in Section 6, we study the initialization problem for EM by comparing three different initialization schemes. Finally, in Section 7, we draw conclusions and point to various directions for further research. Details about AC and its implementation are presented in the Appendix.

2. Model, algorithms, and performance criteria

2.1. The clustering model

We shall consider models for domains consisting of a set of variables $X = X_1, \ldots, X_n$. In general, each variable X_i may be continuous or discrete. Throughout this paper, however, we assume that all the variables are discrete and finitely valued.[1]

Before we describe the model, we need some notation. We denote a variable (which could be a set of variables or vector variable) by a capitalized token (e.g., X, X_i, Class), and the state or value of a corresponding variable by another token in lower case (e.g., x, x_i, k). We use $P(X = x \mid Y = y)$ (or $P(x \mid y)$ as a shorthand) to denote the probability that $X = x$ given $Y = y$. We also use $P(x \mid y)$ to denote the probability distribution for X given Y. Whether $P(x \mid y)$ refers to a probability or a probability distribution will be clear from context.

In this paper, we concentrate on the following simple clustering model:

$$P(x) = \sum_{k=1}^{K} P(\text{Class} = k) \prod_{i=1}^{n} P(x_i \mid \text{Class} = k) \tag{1}$$

In this model, there is a single class variable Class having K mutually exclusive and collectively exhaustive states or values. Each state corresponds to a cluster. Furthermore, conditioned on the value of Class, the variables $X_1, \ldots, X_n$ are mutually independent. As we describe shortly, this model is closely related to the naive-Bayes model for classification.

Each variable in Eq. (1) is a discrete variable and hence can be modeled with a multinomial distribution:

$$P(x_i = j \mid \text{Class} = k) = \theta_{ij}^{(k)}; \qquad \sum_{j=1}^{r_i} \theta_{ij}^{(k)} = 1 \quad \text{for } k = 1, \ldots, K \tag{2}$$

$$P(\text{Class} = k) = \lambda_k; \qquad \sum_{k=1}^{K} \lambda_k = 1, \qquad \lambda_k \geq 0 \tag{3}$$

where r_i is the number of states of variable X_i. The quantities $\{\lambda_k; k = 1, \ldots, K\}$ and $\{\theta_{ij}^{(k)}; j = 1, \ldots r_i,\ i = 1, \ldots, n,\ k = 1, \ldots, K\}$ are the *parameters* of the multinomial distributions and are sometime collectively referred to as the parameters of the model and denoted θ. In the statistics literature, this model is often referred to as a *multinomial-mixture model*.

As we mentioned, this model is closely related to the naive-Bayes model for classification. In both models, the domain variables X are rendered mutually independent by the variable Class. Nonetheless, the model described by Eq. (1) plays different roles in classification and clustering. Having a database of N observations or *cases* $D = \{x^1, \ldots, x^N\}$ the clustering problem consists of finding the *model structure* (i.e., the number of classes K) and parameters θ for that structure that best fits the data D according to some criterion.[2] On the other hand, in classification, the number of classes K and the value of the Class variable for each of the cases in D are given. The task is to learn the model parameters that optimally classify new cases drawn from the same distribution as D.

In what follows, we sometimes refer to the distribution $P(x \mid \text{Class} = k)$ as the *class* or *cluster* k. Also, we sometimes refer to a case for which the variable Class is set to k as being *assigned* to class k, and sometimes refer to the set of all cases having this property as the *class* or *cluster* k.

2.2. Choosing the number of clusters

The clustering algorithms that we consider learn a model in two stages. First, a Bayesian criterion is used to choose the best model structure. Then, the parameters for the best model structure are chosen to be either those parameters whose posterior probability given data is a local maximum (maximum a-posteriori or MAP parameters) or those parameters whose corresponding likelihood is a local maximum (maximum likelihood or ML parameters).

The Bayesian criterion for selecting model structure that we use is the (log) posterior probability of model structure given the training data $\log P(K \mid D_{\text{train}})$, where "$K$" is a shorthand for the model structure with K clusters. By Bayes' formula

$$P(K \mid D_{\text{train}}) = \frac{P(K)P(D_{\text{train}} \mid K)}{P(D_{\text{train}})} \tag{4}$$

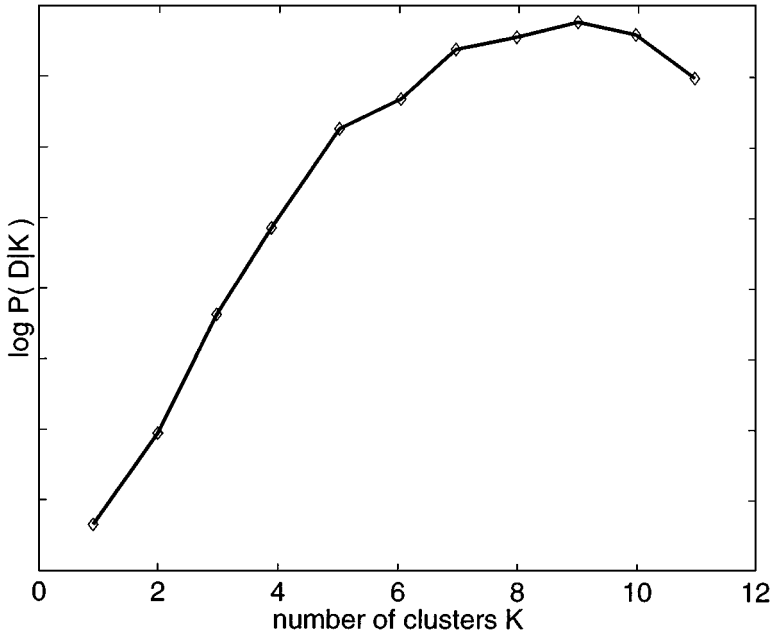

Figure 1. An example of log marginal likelihood as a function of the number of clusters K.

In this paper, we assume equal prior probability for each model structure in a sufficiently large range $K = 1, \ldots, K_{\max}$. Consequently, the criterion reduces to the (log) *marginal likelihood* of the model structure $\log P(D_{\text{train}} \mid K)$. In our experiments, we approximate this quantity using the method of Cheeseman and Stutz (1995) (see also Chickering & Heckerman, 1997).

Figure 1 shows an example of how the log marginal likelihood varies as a function of K. Notice that, as we increase K, the marginal likelihood first increases and then decreases. The "best" number of clusters corresponds to the one at the peak.

An important property of the marginal likelihood, illustrated by this figure, is that it has a built-in penalty for complex models. That is, use of this criterion guards against the overfitting of models to data. To understand this, first notice that the marginal likelihood is the ordinary likelihood of the model averaged over its parameters:

$$P(D_{\text{train}} \mid K) = \int P(D_{\text{train}} \mid \theta, K)\; p(\theta \mid K)\, d\theta \tag{5}$$

This formula highlights the Bayesian methodology wherein all uncertainty is encoded using probability. Namely, the parameters θ are viewed as uncertain with probability density function $p(\theta \mid K)$. Now, consider what happens to the value of the integral in Eq. (5) as we increase the number of clusters in the model, starting at $K = 1$. The peak of the likelihood $P(D_{\text{train}} \mid \theta, K)$ will increase, because each successive model structure can better represent

the training data. On the other hand, the number of parameters in θ will also increase, so that the likelihood $P(D_{\text{train}} \mid \theta, K)$—regarded as a function of theta—will drop off from this peak in more dimensions. This behavior will tend to decrease the value of the integral as K increases. For large enough K, the second tendency will outweigh the first, and the value of the integral will begin to fall.

2.3. Clustering algorithms

We examine several algorithms for learning the parameters of a given model structure: the Expectation-Maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977), the Classification EM (CEM) algorithm (Celeux & Govaert, 1992), and model-based agglomerative clustering (AC) (e.g., Banfield & Raftery, 1993). Sometimes, we shall refer to these parameter-learning algorithms simply as clustering algorithms.

The EM algorithm is iterative, consisting of two alternating steps: the Expectation (E) step and the Maximization (M) step. In the E step, for every x^j, we use the current parameter values of the model to evaluate the posterior distribution of the hidden class node given x^j. We then assign the case fractionally to each cluster according to this distribution. In the M step, we reestimate the parameters to be the MAP (or ML) values given this fractional assignment. The EM algorithm finds a local maximum for the parameters (MAP or ML). The maximum is local in the sense that any perturbation of the parameters would decrease the parameter posterior probability or data likelihood. The algorithm finds the maximum to any desired (non-perfect) precision in a finite number of steps.

The CEM algorithm is similar to the EM algorithm in that CEM also has E and M steps. The algorithm differs from EM in that, within the E step, a case x^j is assigned fully to the class k that has the highest posterior probability given x^j and the current parameter values. That is, no fractional assignments are permitted. The M step in the CEM algorithm is identical to that in the EM algorithm. For continuous variable domains in which the mixture components are spherical Gaussian distributions and the parameters λ_k are all fixed to $1/K$, the CEM algorithm for ML parameters is equivalent to the K-means algorithm. The CEM algorithm converges to local MAP (or ML) parameters under the constraint that each case is assigned to only one cluster. Unlike EM, the CEM algorithm converges completely in a finite number of steps.

The EM and CEM algorithms require initial parameter values. We consider various initialization methods in the following section.

Agglomerative clustering is substantially different from EM or CEM. When using agglomerative clustering, we construct K clusters from a larger number of smaller clusters by recursively merging the two clusters that are closest together. We start with N clusters, each containing one case x^j. Each merge reduces the number of clusters by one. The algorithm stops when the user-specified number of clusters K is reached. One essential ingredient of the algorithm is, of course, the intercluster "distance"[3] $d(k, l)$. The particular form of AC that we examine is model based in the sense that the distance used for agglomeration is derived from a probabilistic model. Banfield and Raftery (1993) introduced a distance $d(k, l)$ derived from a Gaussian mixture model. This distance is equal to the decrease in likelihood resulting by the merge of clusters k and l. Fraley (1997) derives this distance

measure for special cases of Gaussian models, and describes algorithms for accomplishing the clustering in time and memory proportional to N^2. Here we derive this distance metric for mixtures of independent multinomially distributed variables (i.e., discrete naive-Bayes models). (The distance metric can be extended in a straightforward manner to consider differences in parameter posterior probability.) The algorithm we use requires memory only linear in N, and its running time is typically quadratic in N. With minor modifications, the same algorithm can handle certain classes of Gaussian mixtures.

Consider the likelihood of the data D given an assignment of the cases to clusters:

$$\begin{aligned} L(D \mid C_1, \ldots, C_K) &= \sum_{k=1}^{K} \sum_{x^j \in C_k} \log P(x^j \mid \text{Class} = k) \\ &= \sum_{k=1}^{K} L_k(\theta_k) \end{aligned} \tag{6}$$

where $L_k(\theta_k)$ denotes the contribution of cluster C_k to the log-likelihood, and θ_k represents the set of ML parameters of the distribution $P(x \mid \text{Class} = k)$. Note that the term $L_k(\theta_k)$ depends only on the cases in cluster C_k. By merging clusters k and l and assigning all their cases to the newly formed cluster $C_{\langle k,l \rangle}$, the log-likelihood L decreases by

$$L_k(\theta_k) + L_l(\theta_l) - L_{\langle k,l \rangle}\left(\theta_{\langle k,l \rangle}\right) \equiv d(k,l) \geq 0 \tag{7}$$

Because $d(k,l)$ depends only on the cases belonging to the clusters involved in the merge, all the distances $d(k',l')$ between other cluster pairs remain unchanged. Moreover, the ML parameter set θ_k and the value of $d(k,l)$ depend only on a set of sufficient statistics that are stored for each cluster; and these items can be recursively updated when two clusters are merged without direct access to the cases. Using this fact, we have developed a memory efficient distance updating algorithm, described in the Appendix.

Our AC implementation requires $\mathcal{O}(N)$ memory and time between $\mathcal{O}(N^2)$ and $\mathcal{O}(N^3)$. Experiments (Section 6.2) show that typically the running time is close to the lower bound $\mathcal{O}(N^2)$. The algorithm can be generalized to any distance measure that is *local* (i.e., $d(k,l)$ depends only on C_k and C_l). It can also be extended with only minor modifications to certain classes of Gaussian mixtures. Finally, note that, whereas the EM and CEM require an initial point in the parameter space, AC does not.

2.4. Initialization methods

In this section, we describe the initialization methods for EM and CEM that we compare. First, however, we caution that "the best initialization method" is an ill-defined notion, because there is no formal delimitation between initial search and search. For example, when considering the EM algorithm, the ideal initial point would lie somewhere in the domain of attraction of the global optimum. Finding such a point, however, means finding the global optimum to a certain accuracy. This task represents a search per se, and perhaps a more challenging one than that performed by the main EM algorithm. Furthermore, the notion of

"best" will generally involve a tradeoff between accuracy and computation cost. For these reasons, we should not expect to find an initialization method that outperforms all the others on all tasks. Rather, the performance (and relative performance) of an initialization procedure will depend on the data, the accuracy/cost tradeoff, and the "main" search algorithm. As is the case for search algorithms, one can only hope to find initialization methods that perform well on limited classes of tasks that arise in practice. This is the aim of our study on initialization methods.

The first method, the Random approach, consists of initializing the parameters of the model structure independently of the data. Using this approach, we sample the parameter values from an uninformative distribution. In our experiments, we sample the parameters of $P(x_i \mid \text{Class} = k)$ from a uniform Dirichlet distribution with an equivalent sample size equal to the number of states of variable X_i.[4]

The noisy-marginal method of Thiesson et al. (1999) (herein denoted "Marginal") is a data dependent initialization method. Using this approach, we first determine the ML (or MAP) parameter configuration under the assumption that there is only one class. This step can be done in closed form. Next, for each class k and each variable X_i, we create a conjugate (Dirichlet) distribution for the parameters corresponding to $P(x_i \mid \text{Class} = k)$ whose parameter configuration of maximum value agrees with the ML or MAP configuration just computed and whose equivalent sample size is specified by the user. We then sample the parameters corresponding to $P(x_i \mid k)$ from this distribution. In our experiments, we use an equivalent sample size of two and match to the MAP configuration (given a uniform parameter prior).

The distribution of the hidden class variable is initialized to be the uniform distribution when using either of the above methods.

The last initialization method is agglomerative clustering itself. In this data-dependent method, we perform AC on a random sample of the data. From the resulting clusters, we extract a set of sufficient statistics (counts). We then set the model parameters to be the MAP given the counts obtained from agglomeration and a uniform (Dirichlet) prior distribution.

We apply agglomeration to a random sample of the data, because using all the data is often intractable. Furthermore, inaccuracies due to subsampling may be corrected by the EM and CEM clustering algorithms.

2.5. Performance criteria

In this section, we describe the criteria that we use to compare learned models. Some of our criteria measure the quality of the entire model (structure and parameters), whereas other criteria measure the quality of only the model structure—that is, the quality of the assertion that the data is generated from a mixture model with K components.

As we have discussed, the log-marginal-likelihood criterion is used to select the best model structure (best number of clusters). We use this score as one of our criteria.

Another criterion for model structure is the number of clusters in the model and the deviation of this quantity from the true number of clusters K^{true}. Such a measure reflects (in part) how well the learned models help the user to understand the domain under study. This criterion can only be determined for synthetic datasets where the true number of clusters

is available. Furthermore, even when the true model is available, this criterion may be misleading for small datasets, when there is insufficient data to support the true number of clusters. We note that, under certain experimental conditions, some of the learned clusters can be quite small. In the results that we present, we discard (i.e., do not include in the count of number of clusters) any cluster that has less than one case as its member. This situation can arise in models learned by the EM algorithm, due to the fractional assignment of cases to clusters.

A criterion for the entire model is its ability to predict new data. A common practice, both in Machine Learning and in Statistics, is to evaluate the model on "test data" (D_{test}) thought to be a sample from the same true distribution as the training data. We employ the most frequently used such criterion

$$\log P(D_{\text{test}} \mid model) = \sum_{x \in D_{\text{test}}} \log P(x \mid model). \tag{8}$$

This criterion is sometimes called "holdout (log) likelihood". Fisher (1996) describes another model-evaluation criterion that uses a test dataset.

Another criterion for the entire model is *classification accuracy*, defined to be the proportion of cases for which the most likely class k (according to the model) is the true one. We determine the classification accuracy as follows. Because clusters in a learned model are interchangeable, we first must map the learned cluster labels to the actual ones. To do so, we construct the confusion matrix C:

$$C_{ii'} = \#\text{cases}\, j \quad \text{for which } k^j = i, \;\; k^{j*} = i' \tag{9}$$

where j is the case number, k^j is the class that the learned model assigns to case j, and k^{j*} is the true class of case j. Then, we map each cluster k in the learned model to the cluster k' of the true model to which most of its cases belong (this corresponds to a permutation of the rows of C). Once we have mapped the cluster labels, we simply count the cases that have correct labels, and divide by the total number of cases. This criterion can be computed only for synthetic datasets where the true class information is available.

Practical criteria for algorithm comparison include running time and memory requirements. Because all the algorithms that we consider require memory proportional to the number of cases N, the number of observable variables n, and the number of clusters K, there is no further need to make experimental comparison with respect to memory size. Nonetheless, running times per iteration differ for the considered algorithms. Moreover, the number of iterations to convergence for EM and CEM is a factor that cannot be predicted. Hence, experimental comparisons with respect to running time should be informative.

A summary of the performance criteria is given in Table 1. In the results that we report, we divide the marginal and holdout scores by the number of cases in the appropriate dataset. In addition, we use base-two logarithms. Hence, both likelihoods are measured in bits per case. Also, for comparison, we measure the marginal likelihood and holdout scores for a reference model: the one-component ($K = 1$) cluster model.

Table 1. Performance criteria for a model.

Criterion	Expression	Comment
Marginal L	$\frac{1}{\lvert D_{\text{train}}\rvert} \log_2 P(D_{\text{train}} \lvert K^*)$	Bayesian criterion
K^*	K^*	Number of clusters in the selected model
Class acc	$\frac{1}{\lvert D_{\text{test}}\rvert}$ #cases correctly classified	Classification accuracy
Holdout L	$\frac{1}{\lvert D_{\text{test}}\rvert} \log_2 P(D_{\text{test}} \lvert K^*)$	Prediction accuracy on a test set
Runtime		Time to learn parameters of the selected model

3. Datasets

3.1. The synthetic dataset

Synthetic datasets have the advantage that all of our criteria can be used to compare the clustering and initialization algorithms. Of course, one disadvantage of using such datasets is that the comparisons may not be realistic. To help overcome this concern, we constructed a synthetic model so as to mimic (as much as we could determine) a real-world dataset.

The real-world dataset that served as the template for our synthetic model was obtained from the MSNBC news service. The dataset is a record of the stories read and not read by users of the www.msnbc.com web site during a one-week period in October of 1998. In this dataset, each observable variable corresponds to a story and has two states: "hit" (read) and "not hit" (not read). We shall use X_i to refer both to a particular story and its corresponding variable.

A preliminary clustering analysis of this dataset, using both EM and CEM with random initialization, showed the following. (1) There were approximately 10 clusters. (2) The size of clusters followed Zipf's law (Zipf, 1949). That is, the probabilities $P(\text{Class} = k), k = 1, \ldots, K$, when sorted in descending order, showed a power-law decay. (3) The marginal probabilities of story hits also followed Zipf's law. That is, the probabilities $P(x_i = \text{hit})$ for all stories, when sorted in descending order, showed a power-law decay. (4) The clusters overlapped. That is, many users had substantial class membership in more than one cluster. (5) Users in smaller clusters tended to hit more stories. (6) The clusters obtained did not vary significantly when all but the 150 most commonly hit stories were discarded from the analysis. This finding is likely due to item 3.

We used all of these observations in the construction of the synthetic model. In addition, we wanted the synthetic model to be more complex than the models we would attempt to learn (the naive-Bayes model). Consequently, we constructed the model as follows. First, we built a naive-Bayes model where the hidden variable Class had $K = 10$ states and where the observable variables corresponded to the 300 most commonly hit stories. We assigned the distribution $P(\text{Class})$ to be (0.25, 0.18, 0.18, 0.09, 0.09, 0.09, 0.045, 0.035, 0.025, 0.015)–roughly approximating a power decay. Then, for each story variable X_i and for $k = 1, \ldots, 10$, we assigned $P(x_i = \text{hit} \mid \text{Class} = k)$ to be the marginal distribution for story X_i, and perturbed these conditional distributions with noise to separate the clusters.

In particular, for every X_i and for $k = 1, \ldots, 10$, we perturbed the log odds $\log P(x_i = \text{hit} \mid \text{Class} = k)/(1 - P(x_i = \text{hit} \mid \text{Class} = k))$ with normal noise $N(\alpha, 1)$, where $\alpha = -0.5$ for Class $= 1, 2, 3$ (the large clusters), $\alpha = 0$ for Class $= 4, 5, 6$ (the medium-size clusters), and $\alpha = 1$ for Class $= 7, 8, 9, 10$ (the small clusters). These values for α produced a model with overlapping clusters such that smaller clusters contained more hits. Next, we added directed arcs between the observable variables such that each observed variable had an average of two parents and a maximum of three parents. To parameterize the conditional dependencies among the observed variables, we perturbed the log odds $\log P(x_i = \text{hit} \mid pa_i, \text{Class} = k)/(1 - P(x_i = \text{hit} \mid pa_i, \text{Class} = k))$—for every parent configuration pa_i—with normal noise $N(0, 0.25)$.

Finally, we sampled 32,000 cases from the model and then discarded the 150 least commonly hit stories. By discarding these variables (i.e., making them unobserved), we introduced additional dependencies among the remaining observed variables. We refer to this dataset as SY_{32K}. The generative model and datasets are available via anonymous ftp at ftp.research.microsoft.com/pub/dtg/msnbc-syn.

3.2. The digits datasets

Another source of data for our comparison consists of images of handwritten digits made available by the US Postal Service Office for Advanced Technology (Frey, Hinton, & Dayan, 1995). The images were normalized and quantized to 8x8 dimensional binary patterns. For each digit we had a training set of 700 cases and a test set of 400 cases. We present detailed results on one digit, namely **digit6**. Results for other digits that we examined ("0" and "2") are similar.

4. Experimental procedure

An experimental condition is defined to be a choice of clustering algorithm, initialization algorithm, and parameters for each algorithm (e.g., the convergence criterion for EM and the number of subsamples for AC). We evaluate each experimental condition as follows. First, we learn a sequence of models $K_{\min}, \ldots, K_{\max}$ using the training set D_{train} ($2 \leq K \leq 14$ for synthetic data; $2 \leq K \leq 22$ for digits data). For each K, the corresponding model structure is evaluated using the log-marginal-likelihood criterion (in particular, the Cheeseman-Stutz approximation). Then, the number of clusters K^* is chosen to be

$$K^* = \arg\max_K \log P(D_{\text{train}} \mid K) \tag{10}$$

Once the number of clusters K^* is selected, the corresponding model is evaluated on all criteria.

Because the quality of learned models is vulnerable to noise—randomness in the initial set of parameters for EM and CEM, and the subsample of points used for AC—we repeat each experimental condition several times using different random seeds, and average the evaluation results. We call an evaluation for a given experimental condition and random seed a "run". The quantity K^* can vary from run to run for the same experimental condition.

We compute MAP parameters (as opposed to ML parameters) for the EM and CEM algorithms, using an uniform prior for each parameter. For AC, we use ML-based distance. In all trials (except those to be noted), we run EM and CEM until either the relative difference between successive values for the log posterior parameter probability is less than 10^{-6} or 150 iterations are reached.

All experiments are run on a P6 200 MHz computer.

5. Comparison of clustering algorithms

This section presents a comparison between the EM, CEM, and AC algorithms on the **synthetic** and **digit6** datasets. Because AC was too slow to run on a full $32K$ training set (for synthetic data), we experimented with AC on 8,000-case subsets of SY_{32K}. These subsets were too small to result in model structures with the complexity of the true one. Hence, to provide a fair comparison, we first compared EM and CEM using the full SY_{32K} dataset, and then compared the better of these two algorithms with AC using the smaller datasets.

5.1. Synthetic data: EM versus CEM

As already mentioned, for the purpose of this comparison, $D_{\text{train}} = SY_{32K}$ and $D_{\text{test}} = SY_{8K}$. Because both EM and CEM require initial parameters, we compared these algorithms using all three initialization methods.

The results on the **synthetic** data are presented in Table 2. In this and subsequent tables, boldface is used to indicate the best algorithm for each criterion. Here, and in subsequent results, a difference between two algorithms is considered significant if it exceeds the 95% confidence level in the t-test. The table shows the clear dominance of EM over CEM for all initialization methods and for all criteria. The most striking difference is in the choice of K^*, the number of clusters. CEM constantly underestimated K^*, whereas EM, for the two out of three initialization methods used, successfully found the true number of clusters (10).

Several issues concerning classification accuracy are worth noting. First, all classification accuracies were low, because the clusters overlapped significantly. In particular, the classification accuracy for the true model was only 73%. Also, classification accuracy correlated closely with the choice of K^*. When $K^* = 10$, the classification accuracy of the learned model was close to that of the true model. Whereas, when $K^* = 7$ or lower, the classification accuracy of the learned model was approximately two-thirds of that of the true model. From an examination of the confusion matrices for CEM, we found that the underestimation of K^* had two sources: confusion among the 3 largest clusters, and the failure to discover the existence of the smallest clusters. The second source had a smaller impact on classification accuracy.

The only possible advantage of CEM over EM is running time. CEM was about four times faster than EM, because (1) EM requires the accumulation of fractional statistics whereas CEM does not, and (2) CEM takes fewer iterations to converge. We attribute the

Table 2. Performance of the EM and CEM algorithms with three different initialization methods on the **synthetic** dataset SY_{32K} (average and standard deviation over five runs). Runtimes are reported in minutes per class and exclude initialization. Boldface is used to indicate the best algorithm for each criterion. For comparison, the one-component cluster model has marginal and holdout likelihoods of -21.47 and -21.53, respectively. The classification accuracy of the true model is 0.73.

	Initialization					
	Random		Marginal		AC	
	EM	CEM	EM	CEM	EM	CEM
Marginal L	$-$**20.53**	-20.57	$-$**20.51**	-20.66	$-$**20.52**	-20.72
(bits/case)	$\pm$**0.009**	± 0.032	$\pm$**0.0045**	± 0.031	$\pm$**0.013**	± 0.031
K^*	$\mathbf{7 \pm 1}$	6 ± 3	$\mathbf{10 \pm 1}$	7 ± 3	$\mathbf{10 \pm 1}$	7 ± 3
Holdout L	$-$**20.41**	-20.50	$-$**20.36**	-20.52	$-$**20.34**	-20.72
(bits/case)	$\pm$**0.036**	± 0.036	$\pm$**0.018**	± 0.036	$\pm$**0.018**	± 0.072
Class acc	$\mathbf{0.50 \pm 0.05}$	$\mathbf{0.44 \pm 0.01}$	$\mathbf{0.66 \pm 0.04}$	0.46 ± 0.06	$\mathbf{0.68 \pm 0.01}$	0.43 ± 0.04
Runtime	2	**0.5**	2	**0.5**	2	**0.5**

second phenomenon to the fact that CEM's explorations were more constrained, so that the impossibility of an upward move occurred earlier. We shall further examine this issue in Section 5.4.

5.2. Synthetic data: EM versus AC

We next compared EM—the better of EM and CEM—with the AC algorithm using subsets of SY_{32K}. In particular, we created four subsets, creating each subset by randomly sampling 8,000 cases from SY_{32K}.

The results in Table 3 show the clear superiority of EM over AC. For such a small dataset, both algorithms fare poorly in terms of the number of clusters found, but EM finds twice as many. Indeed, an analysis of the confusion matrices showed that AC was unable to distinguish the three largest clusters. In addition, an important difference is seen in the running time. AC runs approximately 60 times slower than EM; and this ratio grows with N because the running times of EM and AC are $\mathcal{O}(N)$ and approximately $\mathcal{O}(N^2)$, respectively.

5.3. Digits data: Comparison of EM, CEM, and AC

For the **digit6** dataset, all three algorithms were trained (tested) on the same training (test) set. The results, showing marginal-likelihood and holdout scores for **digit6** are given in Table 4. Only the results for Marginal initialization of the EM and CEM algorithms are shown. The results for the other initialization methods are similar.

The EM algorithm performed best by both criteria. Also, for this dataset, all the methods choose about the same number of clusters.

Table 3. Performance of the EM and AC algorithms on 8,000-case subsets of SY_{32K} (average and standard deviation over four datasets). Runtimes are reported in minutes per class. For comparison, one-component cluster models have marginal and holdout likelihoods of -21.59 ± 0.12 and -21.54 ± 0.01, respectively.

	EM	AC
Marginal L	$\mathbf{-20.93 \pm 0.108}$	-21.08 ± 0.144
K^*	$\mathbf{4 \pm 0}$	2 ± 1
Holdout L	$\mathbf{-20.63 \pm 0.018}$	-20.88 ± 0.018
Class acc	$\mathbf{0.41 \pm 0.01}$	0.33 ± 0.02
Runtime	**0.6**; $\mathcal{O}(N)$	35; $\mathcal{O}(N^2)$

Table 4. Performance of the EM, CEM, and AC algorithms on the **digit6** dataset (averaged over 20 runs for EM and 10 runs for CEM). EM and CEM were both initialized by the Marginal method. For comparison, the one-component cluster model has marginal and holdout likelihoods of -48.04 and -48.11, respectively.

	EM	CEM	AC
Marginal L	$\mathbf{-35.09 \pm 0.04}$	-35.30 ± 0.07	-35.76
Holdout L	$\mathbf{-32.36} \pm 0.22$	-32.97 ± 0.30	-32.42

We note that, for the digits data, the marginal and holdout likelihoods for the best models are more than 12 bits better than those for the one-component model. In contrast, for the synthetic dataset, improvements are on the order of only 0.5 bit. This difference is consistent with our observation that there was much more cluster overlap in the synthetic data.

5.4. EM versus CEM: Runtime considerations

On our datasets, the EM algorithm dominates AC, because EM is both more accurate and more efficient. So far, however, the case for EM versus CEM is not so clear. As we have seen, the EM algorithm is more accurate, but less efficient. This brings us to the question: If EM is forced to run for a time shorter or equal to the time taken by CEM, is it still more accurate than CEM? In this section, we examine this question.

We adjusted the running time of EM by changing the convergence threshold. We conducted timing experiments and found that convergence thresholds of 4×10^{-4} and 10^{-3} (for the synthetic and digits datasets, respectively) yielded EM speeds that were slightly faster than CEM. We repeated the comparison of EM and CEM, using these new thresholds.

The results for the synthetic and digits datasets are shown in Tables 5 and 6, respectively. For both datasets and all criteria, EM is still more accurate than CEM, although the differences in accuracy are less than what we obtained for the original convergence threshold. All experimental conditions yield a significant difference at the 95% level, except the K^*

Table 5. Performance of the EM and CEM algorithms on the **synthetic** dataset SY_{32K} when EM is allowed to run no longer than CEM (averaged over five runs). Runtimes are reported in seconds per class.

	EM	CEM
Marginal L	$\mathbf{-20.57 \pm 0.026}$	-20.66 ± 0.031
K^*	$\mathbf{9 \pm 2}$	7 ± 3
Holdout L	$\mathbf{-20.42 \pm 0.032}$	-20.52 ± 0.036
Class acc	$\mathbf{0.57 \pm 0.03}$	0.46 ± 0.06
Runtime	**25**	27

Table 6. Performance of the EM and CEM algorithms on the **digit6** dataset when the convergence criterion for EM is set so that EM runs slightly faster than CEM (averaged over ten runs). Runtimes are reported in seconds per class.

	EM	CEM
Marginal L	$\mathbf{-35.26 \pm 0.09}$	$\mathbf{-35.30 \pm 0.07}$
Holdout L	$\mathbf{-32.47 \pm 0.23}$	-32.97 ± 0.03
Runtime	**0.082**	0.089

criterion in the synthetic dataset and the marginal-likelihood criterion in the digits dataset which are significant only at the 85% confidence level.

6. Comparison of initialization methods

We now examine the influence of initialization procedures on the performance of the EM algorithm—the algorithm that performed best in our previous comparison.

6.1. Synthetic data

The results on the synthetic dataset were shown previously in Table 2. We used $D_{\text{train}} = SY_{32K}$ and $D_{\text{test}} = SY_{8K}$, and 2,000-case random samples of the training set for the AC initialization method (a different random sample on each run).

The data independent (Random) method fared worse than did the data dependent methods across all criteria. All differences between the data dependent methods and Random except one—AC versus Random on Marginal Likelihood—are significant. On the other hand, there are no significant differences between Marginal and AC.

The initialization runtimes for Random, Marginal, and AC were 0, 4, and 1800 seconds, respectively. Random is fastest (taking constant time), Marginal is slightly slower (requiring one sweep through the dataset), and AC is hundreds of times slower yet, even though only a portion of the training data was used.

6.2. Sample size for agglomerative clustering

In the previous section, we saw that AC used on a sample of the original training set can provide a good set of initial parameters for the EM algorithm. In this section, we ask the question: How small can we make this sample (of size N') and still obtain good performance?

To answer this question, we varied N' from 125 to 4000. Figure 2 shows the classification accuracy of the resulting models. We obtained similar curves for the other performance criteria.

The graph shows that, as N' decreases from 4000 to 500, the performance decays slowly, accompanied by an increasing variability. Not surprising, an analysis of the confusion matrices showed that performance decreases because, as N' decreases, the smaller clusters are missed and the confusion between the larger ones increase. These observations suggest that the lower limit on N' should be influenced by prior knowledge about the size of

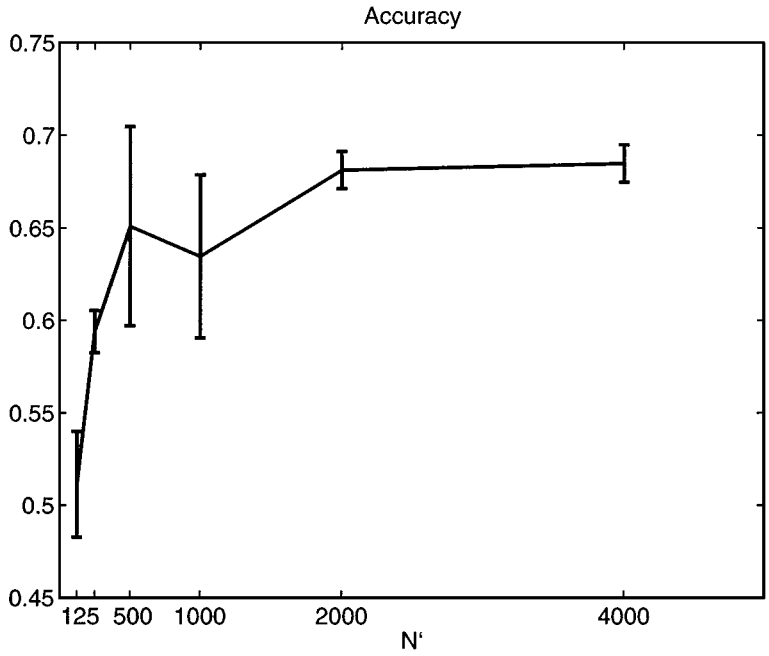

Figure 2. Classification accuracy of the final EM solution versus the AC subsample size N' for the **synthetic** data. Statistics are computed over five runs.

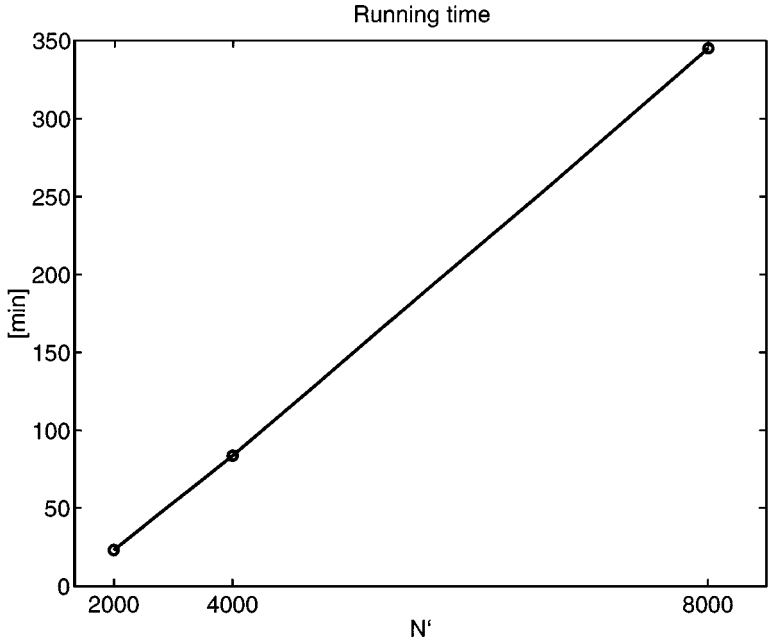

Figure 3. Running time of the EM algorithm initialized with AC versus the AC subsample size N'. The error bars are small and have been omitted. The scale for N' is quadratic.

the smallest cluster. For N' below 500, the classification performance degrades abruptly. Detailed analysis showed that, in this range, the clustering algorithm was failing to separate the largest clusters.

The running time of AC versus the number of samples is shown in figure 3. In this range for N', the time taken by AC initialization strongly dominates the running time for the EM algorithm. Although the theoretical worst case is $\mathcal{O}(N^3)$, the graph shows that the running time approximately follows a quadratic law.

6.3. The digits6 dataset

We compared the three initialization methods on the **digit6** data as well. We examined AC with $N' = N = 700$ and with lower values $N' = 100, 300$ (different random samples in each run).

Table 7 summarizes the results. There were no clear winners. In contrast to the results for the synthetic dataset, Random did not perform worse. Nonetheless, we found two related qualitative differences between the models produced by Random initialization and the other two methods. The clusters learned using Random initialization were greater in number and showed more variability in size. Moreover, Random initialization tended to yield more

Table 7. Performance of the EM algorithm when initialized by the Random, Marginal, and AC methods on the **digit6** dataset (average and standard deviation over 12 or more runs).

	Initialization				
			AC		
	Random	Marginal	$N' = 100$	$N' = 300$	$N' = N = 700$
Marginal L	−35.063	−35.089	−35.091	−35.087	−35.073
(bits/case)	±0.040	±0.042	±0.047	±0.042	
Holdout L	−32.53	−32.36	−32.04	−32.18	−32.27
(bits/case)	±0.255	±0.224	±0.513	±0.238	

small clusters. On average, for the models whose performance was recorded in Table 7, 1.5 clusters were supported by less than one case.

Finally, note that AC followed by EM produced better models than did AC alone (see Tables 4 and 7).

7. Discussion

We have compared several popular batch algorithms for model-based clustering of high-dimensional discrete data. To do so, we have formulated a likelihood based distance measure for agglomerative clustering over discrete variable domains and have introduced a new, memory efficient AC algorithm. Although comparisons with additional datasets (including ones with larger dimension and continuous variables) are needed, our results suggest that the EM algorithm produces better models than do CEM and AC, regardless of the criterion for model evaluation. We found that, for original convergence settings, EM was slower than CEM, and AC was slower yet. Nonetheless, we found that the convergence setting of EM could be adjusted so that EM ran faster than CEM and still produced better models, albeit of lower quality than for the original settings.

These results suggest that the quality of a clustering algorithm correlates well with assignment "softness". Namely, AC assigns each case to only one cluster and this assignment cannot be changed in subsequent iterations. CEM also assigns a case to only one cluster, but each iteration recomputes the assignment. EM not only reevaluates its assignments at each iteration (like CEM), but also allows for partial credit to several clusters.

Finding that the EM algorithm performed best, we studied various ways of choosing initial parameters. Although the three methods used are dissimilar, their performance is similar on all the datasets we examined. On the synthetic dataset, the Random method performed worse than did the data-dependent initialization method, but this difference was not seen for the real-world data. Because we found no difference between AC (as an initialization method) and Marginal, except that Marginal is more efficient, our results suggest that that Marginal initialization is the method of choice. Finding situations

when Random and Marginal produce different results is a possible topic for further research.

Although our results are suggestive, comparisons using additional datasets—for example, ones having 200 or more dimensions and ones that contain non-binary and continuous variables—are needed. In addition, comparisons with more sophisticated variants of the EM algorithm such as EM with conjugate-gradient acceleration (Thiesson, 1995) and the EMν algorithm (Bauer, Koller, & Singer, 1997) should be performed. Finally, model-based clustering algorithms based on models that relax the mutual independence assumption—such as those described by Thiesson et al. (1999)—should be compared.

Appendix: The AC algorithm

Here we describe our implementation of the AC algorithm for discrete variables. The present implementation has the advantage of using an amount of storage which is *linear* in N at the expense of a typically small increase in the running time.

As stated in Section 2 the algorithm starts with N clusters, each containing one case x^j. At each subsequent step, it chooses the two clusters that are closest together according to the distance measure d and merges them, thus reducing the number of clusters by one. The iteration ends when a prescribed number of clusters K is reached. Figure 4 illustrates this procedure.

Computing the intercluster distance. The distance between two clusters k and l is defined by Eq. (7) as:

$$d(k,l) = L_k(\theta_k) + L_l(\theta_l) - L_{\langle k,l\rangle}\left(\theta_{\langle k,l\rangle}\right) \tag{11}$$

In the case of the NBM, $L_k(\theta_k)$ can be computed based on the sufficient statistics $n^k = \#\text{cases}$ in C_k and $n^k_{ij} = \#\text{cases}\ x$ in C_k such that $x_i = j$ for $k = 1,\ldots,K$; $i = 1,\ldots,n$; $j = 1,\ldots,r_i$:

$$\begin{aligned} -L_k(\theta_k) &= -\sum_{j:x^j\in C_k} \log P(x^j \mid \theta_k) \\ &= \sum_{i=1}^{n} n^k \sum_{m=1}^{r_i} \frac{n^k_{im}}{n^k} \log \frac{n^k}{n^k_{im}} \\ &\equiv \sum_{i=1}^{n} n^k h^k_i \end{aligned} \tag{12}$$

The first equality follows from the definitions in Eqs. (1) and (6); in the second equality, the reader can recognize the expression of the ML parameters θ_k in terms of the sufficient statistics n^k_{im} and n^k; in the third equality, the innermost sum is represented by h^k_i, the entropy of variable X_i in cluster k. The terms $L_l(\theta_l)$ and $L_{\langle k,l\rangle}(\theta_{\langle k,l\rangle})$ have similar expressions. In the special case when the two clusters each contain one case, the intercluster distance can be expressed using the Hamming distance[5] between the two cases by

$$d(k,l) = 2\log(2) H(x^k, x^l) \tag{13}$$

This observation provides us with a fast method of initializing the intercluster distances in Step 2 of the AC algorithm. As the practical simulations show, implementing this simple improvement is worth the effort. The running time of Step 2 is only a small fraction of the running time of Step 4, although the number of distance evaluations is roughly the same $(N(N-1)/2)$.

Merging the two closest clusters. The algorithm maintains an ordered list of the currently existing clusters. For each cluster C_k in the list, we store the sufficient statistics necessary to compute the parameters θ_k and the value $L_k(\theta_k)$ itself. Additionally, for each cluster C_k, we store a pointer l_k to the closest cluster to C_k that follows after it in the list and the value δ_k of that distance (Figure 4, Step 2.2).

Obtaining the sufficient statistics and likelihood for a new cluster $C_{\langle k,l\rangle}$ is straightforward:

$$n_{ij}^{\langle k,l\rangle} = n_{ij}^{k} + n_{ij}^{l}; \qquad n^{\langle k,l\rangle} = n^{k} + n^{l} \tag{14}$$

$$L_{\langle k,l\rangle}(\theta_{\langle k,l\rangle}) = L_k(\theta_k) + L_l(\theta_l) - \delta_k. \tag{15}$$

Note that the above update rules make no access to the actual cases. Hence the merge can be performed in constant time no matter how large the clusters. Moreover, the rule is not restricted to NBMs; the same constant time update, using different sets of sufficient statistics, can be applied to the parameters of any distribution in the exponential family (Dobson, 1990).

In contrast to θ_k and $L_k(\theta_k)$, δ_k and l_k cannot be updated locally (i.e., using only the information available in the clusters C_k, C_l). Their update procedure is worst case $\mathcal{O}(N)$ but typically much smaller (Meilă & Heckerman, 1998).

Thus, our implementation requires $\mathcal{O}(N)$ memory and runtime between $\Omega(N^2)$ and $\mathcal{O}(N^3)$. Experimental results presented in Section 6.2 and figure 3 suggest that the average case for runtime is close to the lower bound. Notice that a brute force approach should compute $\mathcal{O}(K'^2)$ distances after the $(N-K')$-th cluster merge. The algorithms presented in Fraley (1997) have runtimes that are strictly quadratic in N, but use $\mathcal{O}(N^2)$ memory.

1. form a list of N clusters and initialize $C_{k^*} \leftarrow x^k$, for $k = 1, \ldots, N$ /* *Start* */
2. for $k = 1, \ldots, N-1$
   2.1. compute $d(k,l)$ for $l = k+1, \ldots, N$
   2.2. $\delta_k = \min_{l>k} d(k,l)$
   $l_k = \operatorname{argmin}_{l>k} d(k,l)$
3. $k^* = \operatorname{argmin}_k \delta_k$; $\delta = \min_k \delta_k$
4. for $K' = N, N-1, \ldots, K$ /* *Iteration* */
   4.1. $C_{k^*} \leftarrow C_{\langle k^*, l_{k^*}\rangle}$ /**merge the two closest clusters and store result over* C_{k^*} */
   4.2. update δ_k, l_k for $k = 1, \ldots, K'-1$
   4.3. erase $C_{l_{k^*}}$
5. for $k = 1, \ldots, K$ /* *Finish* */
   5.1. compute θ_k /* *the parameters of* $P(x|\text{Class} = k)$ */
   5.2. compute $\lambda_k = P(\text{Class} = k)$

Figure 4. The agglomerative clustering algorithm.

When can we use this AC algorithm? The distance updating algorithm presented above depends on an essential property of the intercluster distance d: a merge between clusters k and l does not affect the distance between two other clusters k', l' not involved in the merge. This property is satisfied by all the models where d represents a decrease in log-likelihood as in Eq. (7) and the parameters of the distribution $P(x \mid \text{Class} = k)$ are independent of the parameters of $P(x \mid \text{Class} = l)$ for all $k \neq l$. Fraley (1997) presents some examples of Gaussian cluster models for which the above property does not hold.

Notes

1. A technical point worth mentioning is our use of the term variable and its relationship to the standard definition of a random variable. A *discrete random variable* is a function $X : \Omega \to E$ where E is a discrete set such that $\{w \mid X(w) = x\} \in \mathcal{A}$ for every $x \in E$ where $\mathcal{A}$ is a σ-field and Ω is a sample space of a probability space $(\Omega, \mathcal{A}, P)$. We use the term discrete *variable*, as common to much of the literature on graphical models, to mean a function $X_i : \Omega \to E_i$, parallel to the usual definition of a random variable, but without fixing a specific probability measure P. A model for a set of discrete variables X is simply a probability measure on the Cartesian product $\times_i E_i$. Once a model for X is learned, a variable in our sense becomes a random variable in the usual sense. A similar comment applies to sets X that include continuous variables.
2. Our nomenclature deviates from the standard in statistics. Namely, what we call a "model structure" is typically called a "model"; and what we call a "model" is typically called a "parameterized model".
3. This measure is positive and symmetric, but may not always satisfy the triangle inequality.
4. DeGroot (1970) contains a good introduction to the Dirichlet distribution.
5. The Hamming distance between two vectors of discrete variables is defined as the number of components in which the two vectors are differing—that is, $H(x, x') = \sum_{i=1}^{n} \delta_{x_i x'_i}$, where δ_{ab} is the Kronecker delta.

Acknowledgments

We thank Max Chickering, Chris Meek, and Bo Thiesson for their assistance with the implementation of the algorithms and for many useful and interesting discussions. We also thank Steven White and Ian Marriott for providing the original MSNBC dataset.

References

Banfield, J. & Raftery, A. (1993). Model-based Gaussian and non-Gaussian clustering. *Biometrics, 49*, 803–821.

Bauer, E., Koller, D., & Singer, Y. (1997). Update rules for parameter estimation in Bayesian networks. In D. Geiger and P. Shenoy (Eds.), *Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence*, Providence, RI, (pp. 3–13). San Mateo, CA: Morgan Kaufmann.

Celeux, G. & Govaert, G. (1992). A classification EM algorithm for clustering and two stochastic versions. *Computational Statistics and Data Analysis, 14*, 315–332.

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* (pp. 153–180). Menlo Park, CA: AAAI Press.

Chickering, D. & Heckerman, D. (1997). Efficient approximations for the marginal likelihood of Bayesian networks with hidden variables. *Machine Learning, 29*, 181–212.

Clogg, C. (1995). Latent class models. In *Handbook of Statistical Modeling for the Social and Behavioral Sciences* (pp. 311–359). New York: Plenum Press.

DeGroot, M. (1970). *Optimal Statistical Decisions*. New York, NY: McGraw–Hill.

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

Dobson, A. J. (1990). *An Introduction to Generalized Linear Models*. New York, NY: Chapman and Hall.

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

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clustering. *Journal of Artificial Intelligence Research*, 4:270:281.

Fraley, C. (1997). Algorithms for model-based Gaussian hierarchical clustering. *SIAM Journal on Scientific Computing, 20*, 270–281.

Frey, B., Hinton, G., & Dayan, P. (1996). Does the wake-sleep algorithm produce good density estimators? In D. Touretsky, M. Mozer, & M. Hasselmo, (Eds.), *Neural Information Processing Systems* (Vol. 8, pp. 661–667). Cambridge, MA: MIT Press.

Jain, A. K. & Dubes, R. C. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall.

Meilă, M. & Heckerman, D. (February, 1998). An experimental comparison of several clustering and initialization methods. Technical Report MSR-TR-98-06, Microsoft Research, Redmond, WA.

Thiesson, B. (1995). Accelerated quantification of Bayesian networks with incomplete data. In *Proceedings of First International Conference on Knowledge Discovery and Data Mining*, Montreal, QU (pp. 306–311). San Francisco, CA: Morgan Kaufmann.

Thiesson, B., Meek, C., Chickering, D., & Heckerman, D. (1999). Computationally efficient methods for selecting among mixtures of graphical models, with discussion. In *Bayesian Statistics 6: Proceedings of the Sixth Valencia International Meeting* (pp. 631–656), Oxford: Oxford University Press.

Zipf, G. (1949). *Human Behavior and the Principle of Least Effort*. Cambridge, MA: Addison–Wesley.

Received February 15, 1999
Revised January 21, 2000
Accepted March 29, 2000
Final manuscript March 29, 2000