Bayesian $k$-Means as a “Maximization-Expectation” Algorithm

Kenichi Kurihara
kurihara@mi.cs.titech.ac.jp
Department of Computer Science, Tokyo Institute of Technology, Meguro-ku, Tokyo 152-8552, Japan

Max Welling
welling@ics.uci.edu
Department of Computer Science, University of California, Irvine, CA 92697-3425, U.S.A.

We introduce a new class of “maximization-expectation” (ME) algorithms where we maximize over hidden variables but marginalize over random parameters. This reverses the roles of expectation and maximization in the classical expectation-maximization algorithm. In the context of clustering, we argue that these hard assignments open the door to very fast implementations based on data structures such as kd-trees and congalo lines. The marginalization over parameters ensures that we retain the ability to infer model structure (i.e., number of clusters). As an important example, we discuss a top-down Bayesian $k$-means algorithm and a bottom-up agglomerative clustering algorithm. In experiments, we compare these algorithms against a number of alternative algorithms that have recently appeared in the literature.

1 Introduction

$K$-means is undoubtedly one of the workhorses of machine learning. Faced with the exponential growth of data, researchers have recently started to study strategies to speed up $k$-means and related clustering algorithms. Most notably, methods based on kd-trees (Pelleg & Moore, 1999; Moore, 1998; Zhang, Ramakrishnan, & Livny, 1996; Verbeek, Nunnink, & Vlassis, 2003) and methods based on triangle inequality (Moore, 2000; Elkan, 2003) have been very successful in achieving efficiency that is improved by orders of magnitude.

Another topic of intense study has been to devise methods that automatically determine the number of clusters from the data (Pelleg & Moore, 1999; Hamerly & Elkan, 2003). Some of the most promising algorithms in this respect are based on the variational Bayesian (VB) paradigm (Attias, 2000; Ghahramani & Beal, 2000). Here, distributions over cluster assignments and distributions over stochastic parameters are alternatingly estimated.
in an expectation-maximization-like fashion. Unfortunately, it is difficult to apply the speedup tricks mentioned above to these algorithms without introducing approximations. One of the main goals in this letter is to propose a modification of VB clustering that combines model selection with the possibility of fast implementation.

The technique we propose is an instance of a new class of maximization-expectation (ME) algorithms that reverse the roles of expectation and maximization in the expectation-maximization (EM) algorithm. Alternatively, it can be viewed as a special case of the VB framework where expectation over hidden variables is replaced with maximization. We show that convergence is guaranteed by deriving the objective that is optimized by these iterations.

In the context of clustering, we discuss an ME algorithm that is very similar to $k$-means but uses a full covariance and an upgraded “distance” to penalize overly complex models. We also derive an alternative agglomerative clustering algorithm. Both algorithms can be implemented efficiently using kd-trees and conga lines respectively. Experimentally, we see no performance drops relative to VB, but at the same time, we demonstrate significant speedup factors. (Our software is publicly available at http://mi.cs.titech.ac.jp/kurihara/bkm.html.)

2 Maximization-Expectation Algorithms

Consider a probabilistic model, $P(x, z, \theta)$, with observed random variables (RVs), $x$, hidden RVs, $z$, and parameters, $\theta$ (which are assumed random as well). We adopt the convention that RVs are called hidden if there is a hidden RV for each data case (e.g., cluster assignment variables) and parameters if their number is fixed or needs to be inferred from the data (e.g., cluster centroids).

Given a data set $\mathcal{D}$, a typical task in machine learning is to compute posterior or marginal probabilities such as $p(z \mid \mathcal{D})$, $p(\theta \mid \mathcal{D})$, or $p(\mathcal{D})$. The former can be used to determine, for instance, a clustering of the data, when $z$ represents cluster assignment variables. It is not untypical that exact expressions for these quantities cannot be derived and approximate methods become necessary. We now review a number of approaches based on alternating model estimation.

A standard approach is to represent the distribution using samples, the canonical example being a Gibbs sampler, which alternatingly samples

$$\theta \sim p(\theta \mid z, \mathcal{D}) \leftrightarrow z \sim p(z \mid \theta, \mathcal{D}). \quad (2.1)$$

Instead of sampling, one can fit factorized variational distributions to the exact distribution: $p(\theta, z \mid \mathcal{D}) \approx q(\theta)q(z)$. This variational Bayesian (VB) approximation (Attias, 2000; Ghahramani & Beal, 2000) alternatingly estimates these distributions by minimizing the Kullback-Leibler (KL) divergence,
Figure 1: Four faces of alternating model learning: EE, EM, ME, and MM algorithms.

\[ K L [q(\theta)q(z) \mid \mid p(\theta, z \mid D)] \], between the approximation and the exact distribution. This results in the updates

\[
q(\theta) \propto \exp(\mathbb{E}[\log p(\theta, z, D)_{q(z)}]) \leftrightarrow q(z) \propto \exp(\mathbb{E}[\log p(\theta, z, D)_{q(\theta)}]).
\]

(2.2)

Instead of maintaining a distribution over parameters, one could decide to estimate a maximum a posterior (MAP) value for \( \theta \). To find this MAP value, \( \theta^* \), one can use the EM algorithm to alternatingly compute

\[
\theta^* = \arg\max_{\theta} \mathbb{E}[\log p(\theta, z, D)_{q(z)}] \leftrightarrow q(z) = p(z \mid \theta^*, D).
\]

(2.3)

This turns out to be a special case of the VB formalism by approximating the posterior as \( q(\theta) = \delta(\theta, \theta^*) \), where \( \delta(\cdot) \) is a delta function. Finally, one can also choose to use point estimates for both \( \theta \) and \( z \), which is known as iterative conditional modes (ICM). In this case, we iteratively maximize the posterior distributions, \( p(\theta \mid z, D) \) and \( p(z \mid \theta, D) \), which is equivalent to

\[
\theta^* = \arg\max_{\theta} p(z^*, \theta, D) \leftrightarrow z^* = \arg\max_{z} p(z, \theta^*, D).
\]

(2.4)

Viewing VB as an extension of EM, we could classify VB as an expectation-expectation (EE) algorithm while ICM should be interpreted
as a maximization-maximization (MM) algorithm. This letter is about a new class of algorithms, which we call maximization-expectation algorithms, where we maximize over hidden variables but take expectations over parameters

$$q(\theta) = p(\theta | z^*, D) \leftrightarrow z^* = \arg\max_z E[\log(p(\theta, z, D))]_{q(\theta)}.$$  \hspace{1cm} (2.5)

In analogy to the relation between EM and VB, we can interpret the ME algorithm as an approximation of the VB formalism where we use $q(z) = \delta(z, z^*)$. The main motivation behind this idea is to propose a class of algorithms that is computationally efficient but has the necessary ingredients for model selection. In particular, for the clustering example under consideration, we show that efficient data structures such as kd-trees can be employed with relative ease. Although this letter focuses on the Bayesian $k$-means algorithm, our ideas are rather more general than this.

### 3 Bits-Back Coding

An elegant interpretation for the approximations discussed in the previous section is provided by the bits-back argument (Hinton & van Camp, 1993). Imagine that we want to communicate the data to some receiver as compactly as we can. To that end, we first learn a probabilistic model of the data—say, a mixture of Gaussians MoG model. We use the internal representation of the hidden (or latent, unobserved) variables to encode the data vectors. For instance, the cluster assignments in an MoG can be used to vector-quantize the data vectors. To communicate the data, we first send the parameters of the model that we learned. Next, we send the hidden representation for each data vector. The receiver will predict the data vectors from this information, but since the code we send is “lossy,” we still need to send error corrections (which we hope are small). Note that the sender and receiver need to agree on a quantization level (or resolution) with which they measure parameters, code vectors, and data vectors. Given such a quantization level, it takes fewer bits to encode small error terms than to encode all of the data vectors themselves, resulting in the desired compression.

The reason that the optimal compression results in model selection follows from the fact that the number of parameters sent is constant (they have to be sent only once), while the hidden representation and the error terms scale with the number of data vectors. Hence, when we have a few data vectors to communicate, it does not pay to train a complicated model: the bits needed to send the model parameters may be more expensive than sending the data vectors directly. Using a complicated model to encode the data is warranted if we have to communicate lots of data vectors (assuming
there is structure to be found in the data). This delicate interplay determines the complexity of the model that results in optimal compression.

According to Shannon (1948), we need $\log_2 p(\theta | M)$ bits to encode the parameters, $\log_2 p(z | \theta, M)$ bits to encode the hidden representation $z$ (e.g., cluster assignments), and $\log_2 p(x | z, \theta, M)$ to encode the error terms for a model $M$. Although one is inclined to think that the optimal model and encoding results from minimizing the sum of the three terms over $z$ and $\theta$ (called energy), one can in fact do better. Consider, for instance, the case that two choices for $z$ both minimize the energy. Since both options are optimal, one can use this additional freedom to encode some side information (or future data vectors). In effect, some bits are refunded. More generally, we can use a distribution $q(z, \theta)$ to choose which $z$ and $\theta$ we use for the encoding. This will result in an average energy term of

$$E[q] = \int d\theta \sum_z q(z, \theta) (\log_2 P(x | z, \theta, M) + \log_2 P(z | \theta, M) + \log_2 P(\theta | M)).$$

However, the amount of information contained in our stochastic choices of the code and model parameters is

$$H[q] = -\int d\theta \sum_z q(z, \theta) \log_2 q(z, \theta).$$

(3.1)

Hence, the total number of bits needed to communicate the data becomes $F[q] = E[q] - H[q]$, which is equal to the negative of the logarithm of the marginal likelihood (or evidence).

We are now ready to interpret the approximations made by the four algorithms in Figure 1. The EM algorithm uses stochastic choices for the code vectors $z$ but maximizes over the parameters $\theta$, declining the refund possible from random choices. The VB formalism uses stochastic choices for both $z$ and $\theta$ but uses a factorized distribution $q(z, \theta) = q(z)q(\theta)$ to choose them, which is also suboptimal. (Note that one should in fact expect strong dependencies between $z$ and $\theta$.) ICM will not receive any bits back from either $z$ or $\theta$ because it maximizes over both of them. Finally, in the ME formalism, we receive bits back from the parameters $\theta$, but since we optimize over the encoding $z$, we decline a potential refund that we would have obtained through stochastic choices. The argument we are making in this letter is that this sacrifice will help us develop computationally efficient algorithms.

4 The Large $N$ Limit

In the previous sections, we have viewed the ME procedure as an approximation to either Bayesian model selection or the bits-back coding principle.
In this section, we derive the result that for large $N$, we recover the well-known Bayesian information criterion (BIC) or, equivalently, the maximum description length (MDL) penalty.

The variational bound on the negative marginal log likelihood is given by the following expression in the case of the ME approximation:

$$
\mathcal{F}(K) = \min_z \mathcal{F}(z, K) 
$$

$$
\mathcal{F}(z, K) = -\mathbb{E}[\log p(D, \theta, z')]_{q(\theta)q(z')} - H[q(\theta)]
$$

$$
= -\log p(D) + KL[q(\theta) || \text{p}(\theta, z' | D)]
$$

$$
\geq -\log p(D) \; \forall z, \; \Rightarrow \mathcal{F}(K) \geq -\log(D). \quad (4.2)
$$

We can rewrite $\mathcal{F}(z, K)$ as follows:

$$
\mathcal{F}(z, K) = \int d\theta \left[ -q(\theta) \log p(D, z | \theta) - q(\theta) \log p(\theta) + q(\theta) \log q(\theta) \right].
$$

Since we fixed $z$, we can treat it as observed data alongside $D$. As $N$ grows sufficiently large, $p(D, z | \theta)$ will concentrate on the maximum likelihood solution for the extended observations $\{D, z\}$. Since this is not the maximum likelihood solution, $\theta^{ML}$, for the original model with $z$ integrated out, $p(D | \theta)$, we will denote this as $\theta^*$. We also know that asymptotically, the posterior distribution over parameters (given $z$) converges to a gaussian distribution with a covariance that scales as $O(1/N)$.

Expanding $\log p(D, z | \theta)$ up to second order around its maximum at $\theta^*$, we find

$$
\log p(D, z | \theta) \approx \log p(D, z | \theta^*) + \frac{N}{2} (\theta - \theta^*)^T I_F (\theta - \theta^*), \quad (4.4)
$$

with $I_F$ the Fisher information. Using that the Fisher information scales as $O(1/N)$ and retaining only terms that grow with $N$, we find for the first term of equation 4.3, we find

$$
\log p(D, z | \theta) \approx \log p(D, z | \theta^*). \quad (4.5)
$$

Using a similar approximation for the second term in equation 4.3 but observing that the Hessian for $\log p(\theta)$ does not grow with $N$, we find that it can be ignored. The third term represents the entropy of $q(\theta)$, which asymptotically becomes

$$
H(\theta) \approx -\frac{M}{2} \log N, \quad (4.6)
$$
where $M$ is the number of parameters in the model. Putting everything together, we find

$$
\mathcal{F}(z, K) \approx \log p(D, z | \theta^*) - \frac{M}{2} \log N \tag{4.7}
$$

for every value of $z$. In particular, we can choose the value of $z$ that minimizes $\mathcal{F}(z, K)$, so we also have

$$
\mathcal{F}(K) \approx \log p(D, z^* | \theta^*) - \frac{M}{2} \log N. \tag{4.8}
$$

This limit thus reveals a BIC or MDL penalty term in addition to a likelihood term where marginalization is replaced with minimization. We do not claim that asymptotically, this ME-free energy converges to the true negative log-marginal likelihood since the first (dominant) term will typically not converge to the true likelihood. The reason is that even for very large $N$, the posterior distributions $p(z | \theta^{ML}, D)$ for data cases at the intersection of overlapping clusters will be “soft,” and not deterministic, as in the ME approximation. So $\mathcal{F}(K)$ should be interpreted as a computationally efficient proxy for the true negative log-marginal likelihood, even for very large $N$. Note that for small $N$, the ME approximation is in fact different from the BIC or MDL penalty because it retains all orders in $N$.

5 The Hard Assignment Approximation for Bayesian Clustering

In this section, we consider a specific example of an ME algorithm in the context of clustering.

Let $x_n, \{n = 1, \ldots, N\}$ be a vector of independent and identically distributed continuous-valued observations in $D$ dimensions. For each data case, we define a cluster assignment variable, $z_n \in \{1, \ldots, K\}$. We assume that the data have been generated following a mixture of gaussians, and we have placed the standard conjugate priors on the parameters

$$
p(x_n, z_n, \theta) = \mathcal{N}(x_n | z_n, \mu, \Omega) \mathcal{M}(z_n | \alpha) \mathcal{D}(\alpha | \phi_0) \mathcal{N}(\mu | m_0, \xi_0 \Omega) \times \mathcal{W}(\Omega | \eta_0, B_0), \tag{5.1}
$$

where $\mu$ are the cluster means, $\Omega$ the inverse cluster covariances, and $\alpha$ the mixing coefficients. $\mathcal{N}(\cdot)$ represents a normal distribution, $\mathcal{M}(\cdot)$ a multinomial (or discrete) distribution, $\mathcal{D}(\cdot)$ a Dirichlet distribution, and $\mathcal{W}(\cdot)$ a Wishart distribution. The priors depend on some (nonrandom) hyperparameters, $m_0, \phi_0, \xi_0, \eta_0, B_0$. Uninformative priors are covered in appendix A.

For clustering, we are interested in the posterior distribution $p(z | D)$, where $D$ denotes the data set and $z$ are binary assignment variables for each
data case. In the approximation we propose, we replace the full posterior with the following factorized form,

$$p(z, \theta | D) \approx q(\theta) \delta(z, z^*)$$

(5.2)

that is, we maintain a distribution over parameters, but settle for a point estimate of the assignment variables. Minimizing the KL divergence between this approximate posterior and the exact posterior, $KL[q(\theta)\delta(z, z^*) || p(z, \theta | D)]$, over $q(\theta)$, we find,

$$q(\mu, \Omega, \alpha) = \left[ \prod_{c=1}^{K} \mathcal{N}(\mu_c | m_c, \xi_c \Omega_c) \mathcal{W}(\Omega_c | \eta_c, B_c) \right] D(\alpha | \{\phi_c\})$$

(5.3)

where

$$\xi_c = \xi_0 + N_c$$

$$m_c = \frac{N_c \bar{x}_c + \xi_0 m_0}{\xi_c}$$

$$\eta_c = \eta_0 + N_c$$

$$B_c = B_0 + N_c S_c + \frac{N_c \xi_0}{\xi_c} (\bar{x}_c - m_0)(\bar{x}_c - m_0)^T$$

$$\phi = \{\phi_1, \ldots, \phi_K\}$$

$$\phi_c = \phi_0 + N_c$$

(5.4)

and where $N_c$ denotes the number of data cases in cluster $c$, $\bar{x}_c$ is the sample mean of cluster $c$ and $S_c$ is its sample covariance.

The main simplification that has resulted from the approximation in equation 5.2 is that this distribution now factorizes over the clusters. Using this and equation 4.1, we derive the following bound on the log evidence (or “free energy”),

$$\mathcal{F}(z, K) = \sum_{c=1}^{K} \left[ \frac{D N_c}{2} \log \pi + \frac{D}{2} \log \frac{\xi_c}{\xi_0} + \frac{\eta_c}{2} \log \det(B_c) - \frac{\eta_0}{2} \log \det(B_0) \right.$$

$$- \log \frac{\Gamma_D(\frac{\eta_c}{2})}{\Gamma_D(\frac{\eta_0}{2})} + \frac{1}{K} \log \frac{\Gamma(N + K \phi_0)}{\Gamma(K \phi_0)} - \log \frac{\Gamma(\phi_c)}{\Gamma(\phi_0)} \left] \right.$$

(5.5)

where $\Gamma_D(x) = \pi^{(D-1)/2} \prod_{i=1}^{D} \Gamma(x + \frac{1-i}{2})$ and $\Gamma(\cdot)$ the gamma function. We observe that this objective also nicely factorizes into a sum of terms—one for each cluster. Unlike the maximum likelihood criterion, this objective automatically penalizes overly complex models. Our task is to minimize equation 5.5 jointly over assignment variables $z_n$, $n = 1 \ldots N$ and $K$, the number of clusters.

6 The ME Algorithm for Clustering: Bayesian $k$-Means

In this letter, we study two approaches to minimizing the cost of equation 5.5. The one that we discuss in this section is a Bayesian variant of $k$-means.
Bayesian $k$-Means (BKM), which is an instance of the more general class of ME algorithms. In section 7, we discuss an agglomerative clustering algorithm.

From equation 2.5, we see that the ME algorithm iteratively maximizes $E[\log(p(\theta, z, D))]_{q(\theta)}$, where $p(\theta, z, D)$ can be derived from equation 5.1 and $q(\theta)$ is given by equation 5.3. This is alternated with resetting $q(\theta)$ to the distribution $p(\theta | z^*, D)$, equation 5.3, with $z^*$ the new values for $z$ computed in the previous phase. This leads (after some algebra) to the following iterative labeling cost,

$$C_{\text{BKM}} = \sum_n d_{z_n}(x_n) \quad \text{with}$$

$$d_{z_n}(x_n) = \frac{\eta_{z_n}}{2} (x_n - m_{z_n})^T B_{z_n}^{-1} (x_n - m_{z_n}) + \frac{1}{2} \log \det(B_{z_n}) + \frac{D}{2\xi_{z_n}} - \frac{1}{2} \sum_{d=1}^{D} \left( \frac{\eta_{z_n} + 1 - d}{2} \right) - \Psi(\phi_{z_n}),$$

where $\Psi(\cdot) = \frac{\partial \log \Gamma(\cdot)}{\partial \cdot}$ is the “digamma” function. We recognize $d_{z_n}(x_n)$ as the Mahalanobis distance plus some constant correction term. Due to this extra constant, $d_{z_n}(x_n)$ cannot be interpreted as a proper distance.

The ME algorithm thus alternates updating assignment variables $z_n$ to minimize $C_{\text{BKM}}$ and recalculating the quantities $\{B_c, \xi_c, \eta_c, \phi_c, m_c\}$, which in turn are simple functions of the sufficient statistics $\{N_c, \bar{x}_c, S_c\}$ for each cluster (see equation 5.4). This algorithm is similar to the classical $k$-means algorithm (see algorithm 1). The change in the labeling cost, however, has an essential function in terms of model selection, because unlike the classical $k$-means algorithm, it penalizes overly complex models. To search over different numbers of clusters, we need to introduce cluster split-and-merge operations. We have followed the approach taken in the split-and-merge expectation-maximization (SMEM) algorithm (Ueda, Nakano, Ghahramani, & Hinton, 2000), but note that in our case, we do not have to balance the number of splits and merges to retain the same number of clusters. When we split, we initialize the new clusters with the heuristic that Hamerly and Elkan (2003) used.

The performance of the algorithm depends on the settings of the hyperparameters $\{B_0, \xi_0, \eta_0, \phi_0, m_0\}$ that express our prior expectations. We have used the following data-informed settings for the hyperparameters throughout all our experiments involving BKM,

$$\xi_0 = 0.1 \quad \eta_0 = D$$

$$\phi_0 = 2 \quad B_0 = \frac{d_{\text{small}}^2}{D} \frac{S}{\text{trace}(S)},$$

where $\bar{x}$ is the sample mean of the data, $D$ the dimensionality of the problem, $S$ the sample covariance of the data, and $d_{\text{small}}$ is computed by determining
Algorithm 1: Top-Down Bayesian $k$-Means.

```
1 Initialize:
1i Set hyperparameters \{B_0, \xi_0, \eta_0, \phi_0, \mathbf{m}_0\} using equation 6.2.
1ii Assign all data cases to a single cluster \{z_n\} = 1.
2 Repeat split operations until no more improvement is possible:
2i Use heuristic equation D.1 to rank clusters for splitting.
2ii Split highest-ranked cluster using the procedure in appendix D, section D.3.
2iii Run Bayesian $k$-means updates until convergence:
2iiia Update quantities \{B_c, \xi_c, \eta_c, \phi_c, \mathbf{m}_c\} using equation 5.4.
2iiib Update assignment variables using labeling cost equation 6.1.
2iv Accept or reject split
2iva If free energy in equation 5.5 has decreased: accept split and goto 2i.
2ivb If free energy has increased: reject split, remove candidate from list, and goto 2ii.
3 Merge two clusters into one cluster:
3i Use heuristic equation D.4 to rank pairs of clusters for merging.
3ii Merge highest-ranked cluster.
3iii Run Bayesian $k$-means updates until convergence (see 2iiia and 2iiib above)
3iv Accept or reject merge
3iva If free energy in equation 5.5 has decreased: accept merge and goto 2.
3ivb If free energy has increased: reject merge, remove candidate from list, and goto 3ii.
```

Algorithm 2: Agglomerative Bayesian Clustering.

```
1 Initialize:
1i Set hyperparameters \{B_0, \xi_0, \eta_0, \phi_0, \mathbf{m}_0\} using equation 7.1.
1ii Assign all data cases to a separate cluster \{z_n\} = n.
2 Repeat merge operations until no more improvement is possible:
2i Update quantities \{B_c, \xi_c, \eta_c, \phi_c, \mathbf{m}_c\} using equation 5.4.
2ii Find the pair of clusters that generates the largest decrease in the free energy in equation 5.5.
2iii Merge closest pair by assigning their data cases to a single cluster, and goto 2i.
```

the closest neighbor for 10% of the data cases and averaging over the three closest pairs.

7 Agglomerative Bayesian Clustering

For those interested in the hierarchical structure of the data, bottom-up agglomerative approaches to clustering become interesting. Since we have an explicit objective function in the form of the free energy in equation 5.5, implementing a naive bottom-up clustering algorithm is trivial (see algorithm 2). One starts with assigning a separate cluster to every data case. At each iteration, we search for the best pair of clusters to merge in the sense that they maximally decrease the objective. At some point, the
Bayesian $k$-Means

objective may start to increase, at which point we may continue and choose merges that minimally increase the objective. The point where the objective is minimal has a special status in the sense that it indicates optimal model complexity.

Unfortunately, the settings of the hyperparameters given in equation 6.2 did not perform satisfactorily in obtaining good solutions and good model complexity. The reason is presumably that the agglomerative algorithm is too greedy in trying to optimize the free energy. In particular, in the initial stages of optimization when very few data cases are in each cluster, the priors are overpowering the likelihood term, resulting in suboptimal clusterings that cannot be "undone" in later stages. To diminish the influence of the prior, we have used the following hyperparameter settings in all our agglomerative clustering experiments,

$$
\begin{align*}
\xi_0 &= 0.01 \\
\mathbf{m}_0 &= \bar{x} \\
\eta_0 &= D \\
\phi_0 &= 2 \\
B_0 &= 0.01 d_{\text{small}}^2 I_D,
\end{align*}
$$

(7.1)

where $I_D$ is a $D \times D$ identity matrix.

We emphasize that compensating an overly greedy algorithm with a change in the prior in order to guide the bottom-up optimization to better optima is somewhat of an unprincipled "hack." It would, for instance, be preferable to use look-ahead schemes to obtain better solutions, but this was beyond the scope of what we intended to study for this letter. As a result, we cannot claim this approach to be a principled method to estimate model complexity. It is still useful, because unlike BKM, it returns a full clustering hierarchy instead of a single optimal solution.

8 Clustering for Infinite Mixtures

Dirichlet process mixture models have gained considerable attention in recent years. In this section, we briefly comment on the infinite limit for our model.

In taking the infinite limit, we need to switch from assignment vectors to partitions (see, e.g., Griffiths & Ghahramani, 2006). For example, assignments $(z_1, z_2, z_3) = (1, 1, 2)$ mean $x_1, x_2,$ and $x_3$ are assigned to clusters 1, 1, and 2, respectively. If 1 and 2 are just labels, $(1, 1, 2)$ would be equal to $(2, 2, 1)$. A partition denotes a set of equivalent assignments. In this case, partition $[z]$ is equal to the set $\{(1, 1, 2), (2, 2, 1)\}$; it collapses the space to the much smaller space of assignments equivalent under relabeling the clusters.

Setting $\phi_0$ to $\phi_0/K$, adding the appropriate counting factors necessary in switching from assignments to partitions (see Griffiths & Ghahramani,
2006), and taking the limit $K \to \infty$, we obtain the following free energy,

$$
\mathcal{F}([z], K_+) = \sum_{c=1}^{K_+} \left[ \frac{DN_c}{2} \log \pi + \frac{D}{2} \log \frac{\xi_c}{\xi_0} + \frac{\eta_c}{2} \log \det(B_c)
\right.

- \frac{\eta_0}{2} \log \det(B_0) - \log \frac{\Gamma_D(\frac{\eta}{2})}{\Gamma_D(\frac{\eta_0}{2})} + \frac{1}{K_+} \log \frac{\Gamma(N + \phi_0)}{\Gamma(\phi_0)}

- \log(N_c) - \log \phi_0 \right],

(8.1)
$$

where $K_+$ is the number of clusters that have at least one data case.

Similarly, the labeling cost becomes

$$
C_{BKM} = \sum_n d_{z_n}(x_n)
$$

with

$$
d_{z_n}(x_n) = \frac{\eta_{z_n}}{2} (x_n - m_{z_n})^T B_{z_n}^{-1} (x_n - m_{z_n})

+ \frac{1}{2} \log \det(B_{z_n}) + \frac{D}{2\xi_{z_n}} - \frac{1}{2} \sum_{d=1}^D \Psi \left( \frac{\eta_{z_n} + 1 - d}{2} \right) - \Psi(N_c).
$$

Equation 8.1 is in fact equivalent to $-\log P(D, [z])$ for a Dirichlet proc (DP) mixture. Note, however, that instead of sampling $\{z_n\}$ sequentially, we jointly maximize over $\{z_n\}$. However, compared to equation 5.5, not much has changed, and we do not expect the infinite limit to behave radically different from the finite Bayesian $k$-means algorithm.

9 Efficient Implementations

One of the main motivations behind our hard assignment Bayesian clustering algorithms is to combine model selection with efficiency. The most impressive speedups for $k$-means (at least in low dimensions) are based on special data structures such as kd-trees to avoid redundant distance calculations (Pelleg & Moore, 1999; Moore, 1998; Zhang et al., 1996; Verbeek et al., 2003). Similarly, efficient data structures such as the “conga line” were proposed by Eppstein (1998) to reduce the time complexity of agglomerative clustering methods. In the following, we report on implementing and adapting these techniques to the algorithms under study.

9.1 KD-Trees. In our implementation, we used kd-trees in a similar fashion as reported in Pelleg and Moore (1999), but the details are slightly different due to the fact that our distance function in equation 6.1 is different. (Note that equation 6.1 does not in fact define a distance but that this fact is irrelevant for the workings of the algorithm.)
A kd-tree is a data structure where every node is associated with a hyperrectangle \( h \) containing a subset of data cases, as shown in Figure 2. Each node stores the sufficient statistics \( N_h, t_h, M_h \) of the data cases in its corresponding hyperrectangle, where

\[
N_h = \#(\text{data-cases in } h), \quad t_h = \sum_{x_n \in h} x_n, \quad M_h = \sum_{x_n \in h} x_n x_n^T. \tag{9.1}
\]

The tree is constructed recursively by dividing the hyperrectangles in two smaller hyperrectangles and associating them (and their sufficient statistics) with the child nodes of the kd-tree. The construction is done only once, at the beginning of the Bayesian \( k \)-means algorithm.

To see how we can improve efficiency, we first note that the updates for the Bayesian \( k \)-means algorithm in equation 5.4 depend on only the sufficient statistics of the data, \( \{N_c, \bar{x}_c, S_c\} \) for each cluster \( c \). Instead of recomputing these sufficient statistics after every update by considering each data case separately, we can obtain them using the statistics stored at the nodes of the kd-tree,

\[
N_c = \sum_{h \in \mathcal{H}_c} N_h, \quad \bar{x}_c = \frac{1}{N_c} \sum_{h \in \mathcal{H}_c} t_h, \quad S_c = \frac{1}{N_c} \left( \sum_{h \in \mathcal{H}_c} M_h \right) - \bar{x}_c \bar{x}_c^T. \tag{9.2}
\]

where \( \mathcal{H}_c \) is a set of hyperrectangles whose data cases are all assigned to cluster \( c \). Equation 9.2 provides an efficient computation of the sufficient statistics. For example, the computational complexity of \( \bar{x}_c \) in equation 9.2 is \( O(|\mathcal{H}_c|) \), whereas without the kd-tree, \( \bar{x}_c \equiv \sum_{n=1}^N \delta(z_n, c) x_n \) requires \( O(N) \).

To utilize equation 9.2, a hyperrectangle needs to be associated with a single cluster, that is, the cluster must “own” all the data cases in that hyperrectangle. To associate hyperrectangles to clusters, we start at the root and descend down the tree until we find an owner (see Figure 2). Clearly, the deeper we go, the more inefficient the algorithm will be. The decision
problem of “cluster $c$ is the owner of hyperrectangle $h$” can be defined as follows (where $d_c(x)$ is defined by the labeling cost in equation 6.1):

$$h \text{ is owned by } c \iff d_c(x) < d_c(x) \text{ for all } x \in h, c' \neq c$$

(9.3)

$$\leq d_c^{\max}(h) < d_c^{\min}(h) \text{ for all } c' \neq c$$

(9.4)

$$\leq d_c^{\max}(h) < d_c^{\min}(h) \text{ for all } c' \neq c$$

(9.5)

and

$$d_c^{\max}(h) \equiv \max_{x \in h} d_c(x), \quad d_c^{\min}(h) \equiv \min_{x \in h} d_c(x)$$

$$d_c^{\max}(h) \equiv \max_{x \in h} d_c^+(x), \quad d_c^{\min}(h) \equiv \min_{x \in h} d_c^-(x).$$

$d_c^+$ and $d_c^-$ are the upper and lower bounds of $d_c$, respectively. Note that the use of upper and lower bounds still leads to exact speedups. The only impact will be that we sometimes have to go a bit deeper in the kd-tree to find nodes that are owned by clusters. This slight inefficiency is traded off against the improved efficiency of maximizing and minimizing the bounds instead of computing the exact values for $d_c^{\max}$ and $d_c^{\min}$.

First, we compute $d_c^{\min}(h), d_c^-$ is derived from equation 6.1 by replacing $\frac{n}{2} B_c^{-1}$ with $\lambda_c^{\min} I$, where $\lambda_c^{\min}$ is the smallest eigenvalue of $\frac{n}{2} B_c^{-1}$,

$$d_c^-(x) = \lambda_c^{\min} (x - m_c)^T (x - m_c) + a_c,$$

where $a_c = \frac{1}{2} \log \det(B_{zn}) + \frac{D}{2} \sum_{d=1}^{D} \psi\left(\frac{\eta_+ + 1 - d}{2}\right) - \psi(\phi_{zn})$. Although $\min_{x \in h} d_c(x)$ was originally a constrained convex quadratic program, it is easier to solve $d_c^{\min}(h) = \min_{x \in h} d_c^-(x)$ instead. We can in fact find that $d_c^{\min}(h)$ has the following solutions,

$$d_c^{\min}(h)$$

$$= \begin{cases} a_c & \text{if } m_c \in h \\ \lambda_c^{\min} \sum_{d=1}^{D} \min (\left(\mathbf{u}_h^{(d)} - \mathbf{m}_c^{(d)}\right)^2, \left(\mathbf{l}_h^{(d)} - \mathbf{m}_c^{(d)}\right)^2) + a_c & \text{otherwise} \end{cases}$$

where $\mathbf{u}_h$ and $\mathbf{l}_h$ are the upper and lower boundaries of the hyperrectangle $h$, respectively, and superscript $(d)$ denotes the $d$th dimension of a vector.

Next, we give a solution of $d_c^{\max}(h) = \max_{x \in h} d_c(x)$. The solution to the maximization problem can be shown to lie on one of the corners of the hyper-rectangle. Checking all corners has complexity $2^D$, which is manageable until $D = 8$. After that, we solve $d_c^{\max}(h) = \max_{x \in h} d_c^+(x)$ instead. $d_c^+(x)$ is defined in a similar way to the minimization problem. We replace $\frac{n}{2} B_c^{-1}$ in equation 6.1 with $\lambda_c^{\max} I$, where $\lambda_c^{\max}$ is the largest eigenvalue of $\frac{n}{2} B_c^{-1}$. The
Algorithm 3: Calculation of Sufficient Statistics Using a kd-Tree.

\[ \{N_c, t_c, M_c\}_{c=1}^{K} = \text{descend}(\text{root-node}, \text{empty blacklist}) \]

\begin{function}[N_h^c, t_h^c, M_h^c]_{c=1}^{K} = \text{descend}(\text{node } h, \text{blacklist list}) \]
\text{blacklist any cluster } c' \text{ to list s.t. } \tilde{d}_{c}^{\text{max}}(h) < \tilde{d}_{c'}^{\text{min}}(h) \]
\text{where } c = \arg \min_{c \in \text{blacklist}} d_{c}^{\text{min}}(h) \]
\text{set } \{N_h^c, t_h^c, M_h^c\}_{c=1}^{K} \text{ to zero.} \]
\text{if } |\{c \mid c \notin \text{list}\}| = 1 \text{ (i.e., } h \text{ has an owner cluster } c) \]
\[ N_h^c = h.N_h, \quad t_h^c = h.t_h, \quad M_h^c = h.M_h. \]
\text{else if node is a leaf} \]
\text{calculate } N_h^c, t_h^c, M_h^c \text{ for every } c \text{ according to assignments given by equation 6.1.} \]
\text{else} \]
\[ \{N_h^c, t_h^c, M_h^c\}_{c=1}^{K} = \text{descend}(h.\text{child}_1, \text{list}) + \text{descend}(h.\text{child}_2, \text{list}). \]
\text{end if} \]
\text{end function} \]

Solution can be found as
\[ \tilde{d}_{c}^{\text{max}}(h) = \gamma_c \max_{d=1}^{D} \max ((u_h^{(d)} - m_c^{(d)})^2, (l_h^{(d)} - m_c^{(d)})^2) + a_c. \]

When these solutions of equation 9.5, are used the decision problem of finding cluster \( c \) that is the owner of hyperrectangle \( h \) can deterministically be solved with computational complexity \( O(DK) \) given hyperrectangle \( h \) and cluster \( c \).

To expedite the process of finding owners, we implemented the “blacklisting algorithm” (Pelleg & Moore, 1999). We maintain a list of clusters that could not own the data cases in each hyperrectangle while going deeper into the kd-tree. As we go deeper, more and more clusters get “blacklisted.” This is continued until we find a hyperrectangle small enough so that all its data cases are owned by a single cluster. Deeper hyperrectangles are ignored at that point (see Figure 2). At hyperrectangle \( h \), we blacklist\(^1\) any clusters \( c' \), s.t. \( \tilde{d}_{c}^{\text{max}}(h) < \tilde{d}_{c'}^{\text{min}}(h) \) where \( c = \arg \min_{c \in \text{blacklist}} d_{c}^{\text{min}}(h) \). These calculations are repeated (for nonblacklisted clusters) until all hyperrectangles are owned by a single cluster, or we may stop at any prespecified level (defined as a leaf), where we compute the statistics for every cluster \( c \) separately.

We have summarized the calculation of the sufficient statistics in the pseudocode in algorithm 3. \( t_c \) and \( M_c \) are equal to \( \sum_{h \in \mathcal{H}_c} t_h \) and \( \sum_{h \in \mathcal{H}_c} M_h \) in equation 9.2, respectively. By calling \( \text{descend}(\text{root-node, empty blacklist}) \) the sufficient statistics are thus collected. Note that \( \text{descend}(\text{root-node, empty blacklist}) \) has to be repeated for every hyperparameter update.

\(^1\)We do not need to fix \( c \) as \( c = \arg \min_{c \in \text{blacklist}} d_{c}^{\text{min}}(h) \). In other words, we can blacklist any \( c' \) s.t. \( \tilde{d}_{c}^{\text{max}}(h) < \tilde{d}_{c'}^{\text{min}}(h) \) for “any” \( c \). But this requires \( O(K^2) \) in the worst case. We just fix \( c \) for efficiency.
9.2 Conga Lines. Conga line data structures were proposed by Eppstein (1998) to speed up the agglomerative clustering algorithm from $O(N^3)$ to $O(N^2 \log(N))$. The basic idea is to maintain a changing partition of the data set, where each partition is associated with a directed graph consisting of a collection of paths. There are $O(N)$ edges, and the closest pair will always be represented by some edge in this collection. The data structure gets updated; new partitions and graphs are formed if we remove two clusters and insert the newly merged cluster. It can be shown that these basic operations take $O(N \log(N))$ time instead of the usual $O(N^2)$ for naive implementations. Experimental results on speedup factors are reported in section 11.

10 Other Instances of the ME Algorithm

So far we have described the Bayesian $k$-means algorithm and agglomerative Bayesian clustering. However, it is straightforward to extend the ideas to clustering with discrete attributes. In section 11 we report on some experiments with multinomial distributions and the product of Bernoulli distributions in addition to gaussian distributions.

The derivation of the relevant quantities is detailed in appendixes B and C. The top-down and agglomerative algorithms of these models are the same as the algorithms described in sections 6 and 7, with the free energy and the labeling costs replaced by the ones for these models. Although the labeling costs, equations B.6 and C.6, are in fact different from equation 6.1, it is still possible to apply kd-tree data structures to speed up the algorithm. Empirical speed-up factors are shown in section 11.

11 Experimental Results

In this section, we show experimental results on a number of synthetic and real data sets. We also show the speedup between naive ME algorithms and ME algorithms using kd-trees and conga lines.

11.1 Synthetic Data. In our first experiment, we compared Bayesian $k$-means (BKM) with G-means (software obtained from Hamerly & Elkan, 2003), $k$-means + BIC (see also Pelleg & Moore, 2000), mixture of gaussians (MoG) + BIC, variational Bayesian learning for mixtures of gaussians (VBMG) (Attias, 2000; Ghahramani & Beal, 2000), and a collapsed Gibbs sampler for Dirichlet process gaussian mixtures (MacEachern & Müller, 1998). K-means + BIC, mixtures of gaussians + BIC, and VBMG use the same split-and-merge strategy as BKM but different

---

2The model of VBMG and the collapsed Gibbs sampler is exactly the same as BKM (see equation 5.1).
penalty terms to control the number of clusters. These algorithms were tested on a number of synthetic data sets that were generated similar to Hamerly and Elkan (2003). For each synthetic data set, we sampled $K$ centroids and standard deviations such that no two clusters are closer than $\tau \times (\text{the sum of two standard deviations})/2$. After we generated 10 data sets of size $N$, we multiplied the data within each cluster with a random matrix to give them full covariance structure. Figure 3 shows some typical results for BKM on three random data sets with $\tau = 0.1, 0.5$ and 2.

Table 1 shows the results of this experiment. We set $K$ to 10. Therefore, the closer to 10, the better a result is.3 It should come as no surprise that BKM, VBMG, MoG $+$ BIC, and the collapsed sampler outperform G-means and $k$-means $+$ BIC since the latter assume isotropic covariance. Among BKM, VBMG, and MoG $+$ BIC, it seems hard to discern significant differences in performance in these experiments. The collapsed Gibbs sampler sometimes performed worse than BKM, VBMG, and MoG $+$ BIC. This is because the collapsed Gibbs sampler does not use split-and-merge operations. Thus, it was often trapped at local optima.

We note, however, that the speedup for BKM is exact, while the speedup for MoG $+$ BIC is approximate (Moore, 1998) and no speed-up is currently available for VBMG. The performance of the algorithms VBMG and BKM also depends on the setting of the hyperparameters. The freedom to choose the hyperparameters is an additional source of flexibility. Certainly when good priors can be specified, this should be helpful. In the experiments reported here, we did not make an attempt to improve results by specifying priors on a case-by-case basis but instead used the empirical Bayesian rule of thumb (see equations 6.2 and 7.1) for all experiments. Note that BIC can be derived as the “large-$N$” limit of both VBMG and BKM (see section 4).

---

3Note that since the data were sampled from the (gaussian mixture) model, we should expect a consistent estimator for $K$ to return $K = 10$ if $N$ is sufficiently large.
Table 1: Number of Clusters Estimated on Synthetic Data.

<table>
<thead>
<tr>
<th>τ</th>
<th>N</th>
<th>D</th>
<th>G-Means</th>
<th>k-Means + BIC</th>
<th>MoG + BIC</th>
<th>BKM</th>
<th>VBMG</th>
<th>CGibbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>100</td>
<td>2</td>
<td>4.50 ± 2.42</td>
<td>2.00 ± 1.41</td>
<td>3.80 ± 2.04</td>
<td>2.00 ± 0.82</td>
<td>4.30 ± 2.16</td>
<td>6.32 ± 1.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>7.40 ± 3.86</td>
<td>1.00 ± 0.00</td>
<td>1.70 ± 0.48</td>
<td>2.50 ± 4.74</td>
<td>1.00 ± 0.00</td>
<td>2.60 ± 0.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>13.10 ± 5.47</td>
<td>1.00 ± 0.00</td>
<td>1.00 ± 0.00</td>
<td>13.30 ± 1.06</td>
<td>1.30 ± 0.48</td>
<td>1.70 ± 0.48</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td></td>
<td>32.10 ± 12.33</td>
<td>3.50 ± 1.90</td>
<td>7.80 ± 1.32</td>
<td>7.40 ± 1.96</td>
<td>8.80 ± 0.79</td>
<td>15.11 ± 1.84</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>39.10 ± 10.07</td>
<td>1.10 ± 0.32</td>
<td>5.60 ± 1.65</td>
<td>7.70 ± 1.16</td>
<td>9.90 ± 0.32</td>
<td>7.20 ± 1.14</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>49.40 ± 22.51</td>
<td>1.00 ± 0.00</td>
<td>3.60 ± 1.43</td>
<td>5.70 ± 4.79</td>
<td>2.90 ± 0.74</td>
<td>6.50 ± 0.97</td>
</tr>
<tr>
<td>5000</td>
<td>2</td>
<td></td>
<td>108.90 ± 15.07</td>
<td>3.80 ± 1.62</td>
<td>9.20 ± 2.70</td>
<td>8.10 ± 4.72</td>
<td>9.90 ± 0.32</td>
<td>15.23 ± 2.11</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>130.90 ± 46.17</td>
<td>2.70 ± 1.49</td>
<td>3.90 ± 2.64</td>
<td>10.00 ± 0.00</td>
<td>9.80 ± 0.42</td>
<td>7.90 ± 0.57</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>104.30 ± 51.93</td>
<td>1.30 ± 0.67</td>
<td>2.10 ± 0.32</td>
<td>10.00 ± 0.00</td>
<td>10.00 ± 0.00</td>
<td>7.20 ± 1.55</td>
</tr>
<tr>
<td>0.5</td>
<td>100</td>
<td>2</td>
<td>4.80 ± 2.49</td>
<td>2.50 ± 1.72</td>
<td>5.20 ± 2.35</td>
<td>4.10 ± 3.60</td>
<td>4.40 ± 2.12</td>
<td>6.73 ± 1.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>7.40 ± 4.06</td>
<td>1.00 ± 0.00</td>
<td>2.00 ± 0.00</td>
<td>14.80 ± 5.35</td>
<td>1.00 ± 0.00</td>
<td>3.07 ± 0.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>10.60 ± 6.55</td>
<td>1.00 ± 0.00</td>
<td>1.00 ± 0.00</td>
<td>13.80 ± 0.92</td>
<td>1.40 ± 0.52</td>
<td>1.70 ± 0.48</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td></td>
<td>21.30 ± 4.55</td>
<td>4.20 ± 1.87</td>
<td>9.20 ± 1.32</td>
<td>4.30 ± 3.65</td>
<td>9.40 ± 0.84</td>
<td>14.52 ± 2.45</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>25.60 ± 10.31</td>
<td>1.20 ± 0.42</td>
<td>9.20 ± 0.63</td>
<td>9.20 ± 2.10</td>
<td>9.90 ± 0.31</td>
<td>7.20 ± 0.79</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>37.90 ± 9.72</td>
<td>1.60 ± 1.08</td>
<td>3.80 ± 0.84</td>
<td>11.40 ± 6.15</td>
<td>3.80 ± 1.03</td>
<td>6.10 ± 1.37</td>
</tr>
<tr>
<td>5000</td>
<td>2</td>
<td></td>
<td>82.00 ± 14.91</td>
<td>4.20 ± 1.32</td>
<td>9.10 ± 2.76</td>
<td>8.10 ± 3.57</td>
<td>9.80 ± 0.42</td>
<td>12.00 ± 1.76</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>74.00 ± 32.40</td>
<td>2.60 ± 0.97</td>
<td>8.90 ± 1.45</td>
<td>9.90 ± 0.32</td>
<td>10.00 ± 0.00</td>
<td>7.40 ± 1.26</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>74.30 ± 31.72</td>
<td>2.50 ± 1.72</td>
<td>9.60 ± 0.52</td>
<td>9.70 ± 0.48</td>
<td>9.80 ± 0.42</td>
<td>7.20 ± 1.69</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>2</td>
<td>6.80 ± 3.26</td>
<td>2.10 ± 1.45</td>
<td>6.50 ± 1.96</td>
<td>8.30 ± 1.89</td>
<td>7.20 ± 2.15</td>
<td>5.96 ± 0.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>11.90 ± 1.20</td>
<td>3.30 ± 2.21</td>
<td>2.40 ± 0.00</td>
<td>10.70 ± 5.17</td>
<td>1.00 ± 0.00</td>
<td>4.75 ± 0.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>9.90 ± 3.87</td>
<td>2.20 ± 2.20</td>
<td>1.00 ± 0.00</td>
<td>13.80 ± 0.63</td>
<td>1.00 ± 0.00</td>
<td>2.96 ± 0.68</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td></td>
<td>18.30 ± 5.29</td>
<td>4.30 ± 1.57</td>
<td>9.50 ± 1.65</td>
<td>9.60 ± 1.26</td>
<td>9.80 ± 0.42</td>
<td>8.26 ± 1.63</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>13.90 ± 2.08</td>
<td>6.80 ± 2.94</td>
<td>7.40 ± 1.90</td>
<td>9.40 ± 1.26</td>
<td>9.00 ± 0.94</td>
<td>7.50 ± 0.53</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>13.50 ± 3.66</td>
<td>8.60 ± 2.41</td>
<td>3.00 ± 0.67</td>
<td>12.60 ± 3.31</td>
<td>4.00 ± 1.03</td>
<td>6.40 ± 0.97</td>
</tr>
<tr>
<td>5000</td>
<td>2</td>
<td></td>
<td>48.70 ± 14.22</td>
<td>5.30 ± 2.50</td>
<td>9.60 ± 1.58</td>
<td>10.00 ± 0.00</td>
<td>9.40 ± 1.43</td>
<td>9.33 ± 1.12</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td></td>
<td>15.00 ± 5.79</td>
<td>9.50 ± 1.84</td>
<td>7.90 ± 1.10</td>
<td>10.00 ± 0.00</td>
<td>10.00 ± 0.00</td>
<td>7.50 ± 0.85</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td></td>
<td>11.60 ± 2.07</td>
<td>9.90 ± 2.12</td>
<td>8.40 ± 0.97</td>
<td>10.00 ± 0.00</td>
<td>10.00 ± 0.00</td>
<td>6.60 ± 1.58</td>
</tr>
</tbody>
</table>

Notes: The true number of clusters is 10 for each data set. Results are averaged over 10 runs of the algorithm.
Table 2: Dendrogram Purities on Real Data Sets for Agglomerative Clustering Algorithms.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>ABC</th>
<th>Single Linkage</th>
<th>Complete Linkage</th>
<th>Average Linkage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pendigits</td>
<td>0.730 ± 0.059</td>
<td>0.691 ± 0.055</td>
<td>0.653 ± 0.055</td>
<td>0.707 ± 0.054</td>
</tr>
<tr>
<td>CEDAR</td>
<td>0.465 ± 0.077</td>
<td>0.297 ± 0.051</td>
<td>0.402 ± 0.055</td>
<td>0.403 ± 0.039</td>
</tr>
<tr>
<td>MNIST</td>
<td>0.410 ± 0.078</td>
<td>0.316 ± 0.056</td>
<td>0.358 ± 0.040</td>
<td>0.389 ± 0.048</td>
</tr>
<tr>
<td>Spambase</td>
<td>0.742 ± 0.057</td>
<td>0.632 ± 0.026</td>
<td>0.703 ± 0.036</td>
<td>0.686 ± 0.028</td>
</tr>
<tr>
<td>20 newsgroups</td>
<td>0.448 ± 0.033</td>
<td>0.300 ± 0.008</td>
<td>0.433 ± 0.035</td>
<td>0.356 ± 0.011</td>
</tr>
</tbody>
</table>

Note: ABC was modeled with a mixture of gaussians on handwritten digits, Pendigits, CEDAR, and MNIST, and with a product of Bernoullis on spambase and 20 newsgroups.

### 11.2 Real Data

Next, we evaluated agglomerative Bayesian clustering (ABC) against the following traditional agglomerative clustering methods: single, complete, and average linkages. We ran these algorithms on real data sets: Pendigits, CEDAR data set, MNIST data set, UCI spambase data set, and 20 newsgroups data set. The Pendigits, CEDAR, and MNIST data sets are handwritten digits (0–9) data, and they have 16, 64, and 784, dimensions, respectively. Following Hamerly and Elkan (2003), we applied a random projection to the MNIST data set to reduce the dimension to 50. From the 20 newsgroups data set, we used only rec.autos, rec.sport.baseball, rec.sport.hockey, and sci.space categories and chose 50 features using mutual information. The spambase and the 20 newsgroups data sets were binarized. ABC used an MoG on the handwritten digits and a product of Bernoullis on the spambase and 20 newsgroups data sets.

We evaluate results with the dendrogram purity.\(^4\) Table 2 shows the results. For each data set, we created 10 random subsets. Each of the handwritten digits data sets contains 100 data cases. Spambase and 20 newsgroups contain 200 and 240 data cases respectively. On each data set, ABC built dendrograms that have the highest dendrogram purity.

Figure 4 is an illustrative example of a dendrogram for a small subset of spambase. We used 50 data cases only (25 spam cases, “N,” and 25 nonsam cases, “S”) so that every label can be plotted on the x-axis. The purity was 0.816 for this example. The height of each linkage represents the negative free energy. Since a valid dendrogram accepts positive heights, we added 11,000 to the negative free energy.

---

\(^4\)Dendrogram purity is defined as follows. Pick a leaf node \(i\) and another leaf node \(j\) assigned to the same class randomly. Measure the fraction of leaf nodes in the smallest subtree containing both \(i\) and \(j\) that are assigned to the same class as \(i\) and \(j\). The expected value of this fraction is the dendrogram purity. The overall tree purity is 1 if and only if all leaves in each class are contained within some pure subtree. This quantity can be efficiently computed in a single bottom-up pass through the tree.
Figure 4: An illustrative example of a dendrogram built by ABC on the spam-base data set using a product of Bernoulli’s distribution. The purity was 0.816. Labels N and S denote nonspam and spam. The number of data cases is 50, where 25 data cases are spam and 25 are nonspam. Heights represents the negative free energy. Since a valid dendrogram accepts positive heights, we added 11,000 to the negative free energy.

11.3 Efficient Implementations. Finally, we compared naive ME algorithms with efficient implementations using kd-trees and conga lines. For the BKM + kd-trees experiment, we varied one parameter at a time and used default values of $N = 20,000$, $D = 2$, $\tau = 3$, and $K = 5$ to sample data. A node in a kd-tree is declared a leaf if the number of data cases in the node is less than 1000 (the remaining points are treated individually). In the experiments with ABC + conga lines, we vary $N$ and use $D = 2$, $\tau = 3$, and $K = 10$. For the multinomial ME (MME) + kd-trees, we set default values, $N = 10,000$ and $D = 8$, and varied $N$ and $D$.

Figure 5 shows the speedup factors. For BKM, speed-up increases very fast with $N$ (factor of 67 at $N = 80,000$) and moderately fast with $\tau$ (less overlap helps to blacklist competing clusters) and decreases surprisingly slowly with dimension (factor of 3.6 in 256 dimensions at $N = 20,000$). ABC using the conga lines is only slightly faster than the naive implementation due to large overhead, but this clearly becomes more significant with growing $N$. MME also achieved speedup using kd-trees, but it was not as significant as BKM.
Bayesian k-Means

Figure 5: Speedup factors using fast data structures, kd-trees, and conga lines. For Bayesian k-means (BKM) + kd-trees, the number of data cases, dimensions, and τ was varied. The result of agglomerative Bayesian clustering (ABC) + conga lines shows two plots; × and ∗ denote overall speedup factor and speedup factor per iteration, respectively. For the multinomial ME (MME) + kd-trees, the number of data cases and dimensions was varied.

12 Conclusion

The contributions of this letter are twofold. First, a new class of algorithms is introduced that reverses the roles of expectation and maximization in the traditional EM algorithm. This combines the potential for model selection (because we integrate over the parameters) with fast implementations (because hard assignments are well suited for efficient data structures such
as kd-trees). Second, we have implemented and studied one possible application of this idea in the context of clustering. Explicit algorithms were derived and experimentally tested for bottom-up Bayesian agglomerative clustering and top-down Bayesian $k$-means clustering.

The experimental results have verified that for data sampled from a gaussian mixture, the BKM algorithm is competitive with alternative approaches in finding the correct number of clusters. We have also verified that the agglomerative Bayesian clustering algorithm scored high on dendrogram purity relative to some standard competitors in the literature. Finally, and perhaps most significant, we found strong speedup results for the gaussian model as we increase the number of data cases. We also observed a surprisingly graceful decrease of speedup factors as we increased the dimensionality of the data.

It is well known that kd-trees break down in high dimensions. Therefore, for high-dimensional data, we recommend running the proposed algorithms on random (or informed) projections of the data. Strong theoretical and empirical results exist that show that surprisingly little information about the cluster structure is lost in these projections Dasgupta (2000).

Although we have explored these ideas in the simplest possible context, namely clustering, the proposed techniques seem to readily generalize to more complex models such as hidden Markov models. Whether the efficiency gains can also be achieved in this setting remains to be investigated.

**Appendix A: Bayesian K-Means with Uninformative Priors**

In this appendix, we derive the free energy of Bayesian $k$-means with uninformative priors. First, we set uninformative priors on the parameters,

$$p(\mu_c) = \prod_{d=1}^{D} \frac{1}{(x_{\text{max}}^{(d)} - x_{\text{min}}^{(d)})}, \quad p(\Omega_c) = \det(\Omega_c)^{-(D+1)/2}, \quad p(\alpha) = \Gamma(K),$$

(A.1)

where $x_{\text{max}}^{(d)} = \max_n x_n^{(d)}$, $x_{\text{min}}^{(d)} = \min_n x_n^{(d)}$, and $x_n^{(d)}$ denote the $d$th dimension value of $x_n$.

As we see in section 5, we find an approximate posterior:

$$q(\mu, \Omega, \alpha) = \left[ \prod_{c=1}^{K} \mathcal{N}(\mu_c | \bar{x}_c, \bar{\Omega}_c)\mathcal{W}(\Omega_c | N_c - 1, N_c S_c) \right] \mathcal{D}(\alpha | N_c + 1).$$

(A.2)
Bayesian k-Means

From this posterior and equation 4.1, we derive the free energy:

\[
\mathcal{F}(z, K) = \sum_{c=1}^{K} \left[ \frac{D(N_c - 1)}{2} \log \pi + \frac{1}{2} \log N_c \\
+ \frac{N_c - 1}{2} \log \det(N_c S_c) - \log \Gamma_D \left( \frac{N_c - 1}{2} \right) \\
+ \frac{1}{K} \log \frac{\Gamma(N + K)}{\Gamma(K)} - \log \Gamma(N_c + 1) + \sum_{d=1}^{D} \log (x_{\text{max}}^{(d)} - x_{\text{min}}^{(d)}) \right].
\]

(A.3)

We notice that this free energy is singular if there exists a cluster \( c \) such that \( N_c \leq (D - 1)/2 \). This singularity leads to problems in agglomerative clustering because \( N_c \) is equal to 1 for each class \( c \) in the first step of agglomerative clustering.

The labeling cost is given by

\[
C_{\text{BKM}} = \sum_n d_{zn}(x_n) \quad \text{with} \quad d_{zn}(x_n) = \frac{N_{zn}(N_{zn} - 1)}{2} (x_n - \bar{x}_{zn})^T S_{zn}^{-1} (x_n - \bar{x}_{zn}) \\
\quad + \frac{1}{2} \log \det(N_{zn} S_{zn}) + \frac{D}{2N_{zn}} \frac{1}{2} \sum_{d=1}^{D} \Psi \left( \frac{N_{zn} - d}{2} \right) - \Psi(N_{zn} - 1).
\]

(A.5)

This labeling cost shows that we can apply kd-trees in the same way as in section 9.1.

Appendix B: Multinomial ME Algorithm

In this section, we derive an ME algorithm modeled with multinomial. Let \( x_n \) be a vector, \( (x_{n1}, \ldots, x_{nD}) \), where \( x_{nd} \) denotes the number of occurrences of event \( d \) and \( \sum_{d=1}^{D} x_{nd} = W \).

The joint probability of \( x_n \) and \( z_n \) is described as

\[
p(x_n, z_n | \alpha, \beta) = M(z_n | \alpha) M(x_n | \beta_{zn}),
\]

(B.1)

where \( \alpha \) and \( \beta \) are the parameters of multinomials. We place conjugate priors on the parameters; then the marginal probability is given as

\[
p(x_n, z_n, \theta) = M(z_n | \alpha) M(x_n | \beta_{zn}) D(\alpha | \phi_0) D(\beta_{zn} | \psi_0).
\]

(B.2)
where $\phi_0$ and $\psi_0$ are hyperparameters. In our experiments, we set $\phi_0$ and $\psi_0$ to 1 and 0.1, respectively.

Similar to equation 5.3, we derive an approximate posterior,

$$q(\alpha, \beta) = D(\alpha | \phi) \prod_{c=1}^{K} D(\beta_c | \psi_c).$$

where

$$\phi = \{\phi_1, \ldots, \phi_K\} \quad \phi_c = \phi_0 + N_c$$

$$\psi_c = \{\psi_{c1}, \ldots, \psi_{cD}\} \quad \psi_{cd} = \psi_0 + \sum_{n_c=1}^{N_c} x_{n_c d}.$$  \hfill (B.4)

Using equation 4.1, we derive the free energy:

$$\mathcal{F}(z, K) = \sum_{c=1}^{K} \left[ \log \frac{\Gamma(\phi_0)}{\Gamma(\phi_c)} + \log \frac{\Gamma(\sum_{d=1}^{D} \psi_{cd})}{\Gamma(\sum_{d=1}^{D} \psi_0)} + \sum_{d=1}^{D} \log \frac{\Gamma(\psi_0)}{\Gamma(\psi_{cd})} + \frac{1}{K} \log \frac{\Gamma(K \phi_0 + N)}{\Gamma(K \phi_0 + N)} \right] + \sum_{n=1}^{N} \log \frac{\Gamma(\sum_{d=1}^{D} (x_{n d} + 1))}{\Gamma(W + 1)}.$$ \hfill (B.5)

The labeling cost takes the same general form as in section 6,

$$\mathcal{C}_{\text{MME}} = \sum_{n} d_{zn}(x_n),$$  \hfill (B.6)

where

$$d_{zn}(x_n) = a_{zn}^T x_n + b_{zn}$$

$$a_{zn} = [a_{z1}, \ldots, a_{zD}]^T \quad a_{zd} = -\Psi(\psi_{zd}) + \Psi\left(\sum_{d=1}^{D} \psi_{zd}\right)$$

$$b_{zn} = -\Psi(\phi_{zn}).$$

Note that this labeling cost is linear in $x_n$, whereas BKM’s labeling cost is quadratic. Therefore, it is in fact easier to apply kd-trees to BKM with multinomial distributions than with gaussian distributions.
Appendix C: Product of Bernoulli’s ME Algorithm

In this section, we apply the ME algorithm to a product of Bernoulli’s model. Let \( x_n \) be a vector, \((x_{n1}, \ldots, x_{nD})\), where \( x_{nd} \in \{0, 1\} \).

The joint probability of \( x_n \) and \( z_n \) is described as

\[
p(x_n, z_n | \alpha, \lambda) = \mathcal{M}(z_n | \alpha) \prod_{d=1}^{D} B(x_{nd} | \lambda_{zd,d}),
\]

where \( B(\cdot) \) denotes a Bernoulli distribution, \( B(x_{nd} = 1 | \lambda_{zd,d}) = \lambda_{zd,d} \) and \( B(x_{nd} = 0 | \lambda_{zd,d}) = 1 - \lambda_{zd,d} \). When conjugate priors are used, the marginal probability is given as

\[
p(x_n, z_n, \theta) = \mathcal{M}(z_n | \alpha) \mathcal{D}(\alpha | \phi_0) \prod_{d=1}^{D} B(x_{nd} | \lambda_{zd,d}) \mathcal{D}(\lambda_{zd,d} | \omega_0),
\]

where \( \phi_0 \) and \( \omega_0 \) are hyperparameters. In our experiments, we set them to 1 and 0.1, respectively.

Similar to the preceding section, we derive the approximate posterior,

\[
q(\alpha, \lambda) = \mathcal{D}(\alpha | \phi) \prod_{c=1}^{K} \prod_{d=1}^{D} \mathcal{D}(\lambda_{cd} | \omega_{cd}),
\]

where

\[
\phi = \{\phi_1, \ldots, \phi_K\} \quad \phi_c = \phi_0 + N_c
\]

\[
\omega_{cd} = \{\omega_{cd0}, \omega_{cd1}\} \quad \omega_{cd0} = \omega_0 + \sum_{n_c=1}^{N_c} \delta(x_{n_c,d}, b) \quad b \in \{0, 1\}.
\]

By substituting this approximate posterior into equation 4.1, we find the free energy:

\[
\mathcal{F}(z, K) = \sum_{c=1}^{K} \sum_{d=1}^{D} \left\{ \log \frac{\Gamma(\omega_{cd0} + \omega_{cd1})}{\Gamma(2\omega_0)} - \log \frac{\Gamma(\omega_{cd0})}{\Gamma(\omega_0)} - \log \frac{\Gamma(\omega_{cd1})}{\Gamma(\omega_0)} \right\}
\]

\[
+ \frac{1}{K} \log \frac{\Gamma(K\phi_0 + N)}{\Gamma(K\phi_0)} - \log \frac{\Gamma(\phi_c)}{\Gamma(\phi_0)}.
\]

The labeling cost is given by

\[
C_{BME} = \sum_{n} d_{zn}(x_n),
\]
where
\[ d_{zn}(x_n) = a_{zn}^T x_n + b_{zn} \]
\[ a_{zn} = [a_{zn1}, \ldots, a_{znd}]^T \quad a_{znd} = -\Psi(\omega_{znd1}) + \Psi(\omega_{znd0}) \]
\[ b_{zn} = \sum_{d=1}^{D} \left\{ -\Psi(\omega_{znd0}) + \Psi(\omega_{znd0} + \omega_{znd1}) \right\} - \Psi(\phi_{zn}). \]

Equation C.6 is also linear in \( x_n \), facilitating the application of kd-trees.

**Appendix D: Split-and-Merge Procedure**

We have followed the split-and-merge procedure in the SMEM algorithm (Ueda et al., 2000). In this section, we describe split-and-merge criteria, \( J_{\text{split}} \) and \( J_{\text{merge}} \), and the initialization of newly created clusters.

**D.1 Split Criterion.** The split criterion \( J_{\text{split}} \) is defined as
\[ J_{\text{split}}(c) = \int dx \ f_c(x | \bar{\theta}) \log \frac{f_c(x | \bar{\theta})}{p(x | c, \bar{\theta})}, \]  
where
\[ f_c(x | \bar{\theta}) = \frac{\sum_{n=1}^{N} \delta(x - x_n) p(c | x_n, \bar{\theta})}{\sum_{n=1}^{N} p(c | x_n, \bar{\theta})}, \]
with \( \bar{\theta} = \int d\theta \ \theta \ q(\theta). \)

\( J_{\text{split}} \) is a KL divergence between the local empirical density \( f_c(x) \) around cluster \( c \), where each data case is weighted according to its responsibility under a mixture-of-gaussians model with average parameters \( \bar{\theta} \). A cluster that has the largest value for \( J_{\text{split}} \) should be split, because the large \( J_{\text{split}} \) means that the cluster has a poor estimate of its local density.

**D.2 Merge Criterion.** The merge criterion \( J_{\text{merge}} \) is a cosine between \( N \)-dimensional vectors \( P_{c1} \) and \( P_{c2} \),
\[ J_{\text{merge}}(c1, c2 | \bar{\theta}) = \frac{P_{c1}^T P_{c2}}{\|P_{c1}\| \|P_{c2}\|}, \]
where
\[ P_c = (p(c | x_1, \bar{\theta}), \ldots, p(c | x_N, \bar{\theta}))^T \]
represent the responsibilities. When these responsibility vectors are very similar, for two clusters that is, their $J_{\text{merge}}$ is large, the clusters are good candidates for a merge.

**D.3 Initialization of New Clusters.** Clusters created by split or merge are initialized as follows. When we split a cluster $c$, we run one iteration of the usual $k$-means algorithm with centroids $m \pm d$, where $m$ is the centroid of cluster $c$ and $d = s\sqrt{\lambda}$ with $s$ is the principal eigenvector of cluster $c$ and $\lambda$ is its eigenvalue.

After we merge clusters, we compute new hyperparameters using equation 5.4.

**Acknowledgments**

We are grateful to Moore (1998) and Hamerly and Elkan (2003) for sharing their code and to Katherine Heller for data sets. Also, special thanks to Y. W. Teh for suggesting infinite mixtures. Finally, we acknowledge Andrew Moore for answering questions about data structures for fast implementations. This material is based on work supported by the National Science Foundation under grant 0447903.

**References**


k-d trees. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), Advances in neural
information processing systems, 10. Cambridge, MA: MIT Press.

Moore, A. (2000). The anchors hierarchy: Using the triangle inequality to survive
high-dimensional data. In Proc. of the 12th Conf. on Uncertainty in Artificial Intelli-

reasoning. In Proc. of the 5th Int’l Conf. on Knowledge Discovery in Databases (pp.

tion of the number of clusters. In Proceedings of the 17th International Conference on
Machine Learning (pp. 727–734). San Francisco: Morgan Kaufmann.


mixture models. Neural Computation, 12(9), 2109–2128.

for gaussian mixtures (Tech. Rep.) Amsterdam: University of Amsterdam.

method for very large databases. In ACM-SIGMOD Int. Conf. Management of Data

Received December 16, 2006; accepted October 25, 2007.