1 Abstract

Clustering is a fundamental task in many vision applications. To date, most clustering algorithms work in a batch setting and training examples must be gathered in a large group before learning can begin. Here we explore incremental clustering, in which data can arrive continuously. We present a novel incremental model-based clustering algorithm based on nonparametric Bayesian methods, which we call Memory Bounded Variational Dirichlet Process (MB-VDP). The number of clusters are determined flexibly by the data and the approach can be used to automatically discover object categories. The computational requirements required to produce model updates are bounded and do not grow with the amount of data processed. The technique is well suited to very large datasets, and we show that our approach outperforms existing online alternatives for learning nonparametric Bayesian mixture models.

2 Introduction

Discovering visual categories automatically with minimal human supervision is perhaps the most exciting current challenge in machine vision [WWP00, SRE+05]. A related problem is quantizing the visual appearance of image patches, e.g., to build dictionaries of visual words in order to train recognition models for textures, objects, and scenes [LM99, VNU03, DS03, FFP05, JT05]. This second problem is easier, because the features (e.g., pixels, SIFT coordinates) have been agreed upon in advance and do not need to be discovered as part of the process. In both cases unsupervised clustering is an important building block of the system.

Unsupervised clustering is usually carried out in batch on the entire training set. Here we consider instead ‘incremental’ or ‘on line’ unsupervised clustering. There are two reasons why incremental clustering or category learning may be useful. First of all, an organism, or a machine, has a competitive advantage if it can immediately use all the training data collected so far — rather than wait for a complete training set. Second, incremental methods usually have smaller
Figure 1: Top: A sample of the inputs to the incremental learning process. Middle: Cluster means discovered by incremental algorithm after 6000, 12000, and 30000 digits processed. As expected, the model complexity increases as data arrives. The computational burden per model update is not a function of the number of data points processed; it grows more slowly with the number of clusters discovered. Bottom Left: Cluster centers produced by incremental algorithm after visiting all 60000 digits, with effective memory size of 6000 digits. Bottom Right: Cluster centers produced by batch algorithm. Clusters are ordered according to size, from top left to bottom right. Our incremental algorithm requires substantially less memory and is faster than the comparable batch algorithm. See Section 5 for a description of the algorithm and Section 6 for more information about the experimental results.
memory requirements: new training examples are used to update a ‘state’ and then the examples are forgotten. The state summarizes the information collected so far – it typically consists of a parametric model and it thus occupies a much smaller amount of memory than a full-fledged training set. So: when the system has to operate while learning, when the memory available is small (as in an embedded system), or when the training data are very voluminous, an incremental method is the way to go. It has to be expected that an on-line method is not as efficient in extracting information from the data as a batch method. This is because decisions must often be taken without the benefit of future information.

A challenge for clustering methods, one that is often swept under the rug, is determining the complexity of the final model: “How many clusters should I plan for?” Batch methods have the luxury of solving this question by trial-and-error: fit many models, from simple to complex, and pick the one that maximizes some criterion, e.g., the likelihood on a validation set. Estimating the complexity of the model is much harder for on-line methods. Furthermore, the complexity is likely to grow with time, as more training examples are acquired and stronger evidence is available for subtler distinctions.

We present a new approach for learning nonparametric Bayesian mixture models incrementally. Our approach has a number of desirable properties: it is incremental, it is non-parametric in the number of components of the mixture, its memory use is parsimonious and bounded. Empirically, we find that it makes good use of the information provided by the training set, almost as good as a batch method, while being faster and able to tackle problems the size of which a batch method is unable to approach.

Section 3 provides background on the Dirichlet Process mixture model and sufficient statistics. Section 4 briefly describes existing approaches to the problem and Section 5 explains our approach. Section 6 shows experimental results on an object recognition problem, clustering of MNIST digits, and a large image patch clustering experiment. Discussions and conclusions may be found in Section 7.

3 Background

We start by reviewing the Dirichlet Process mixture model (DPMM) [Ant74].

3.1 Dirichlet Process Mixture Model

The DPMM extends the traditional mixture model to have an infinite number of components. Data points \(x_t\) are assumed to be drawn i.i.d. from the distribution:

\[
p(x_t) = \sum_{k=1}^{\infty} \pi_k p(x_t | \phi_k),
\]

where \(\phi_k\) are component parameter vectors and \(\pi_k\) are a set of mixing weights that sum to 1. During inference, the mixing weights and the component param-
eter vectors are treated as random quantities. A probabilistic structure known as the Dirichlet Process [Fer73] defines a prior on these random variables.

The component parameters $\phi_k$ are assumed to be independent samples from a probability distribution $H$. The mixture weights $\pi_k$ may be constructed from a countably infinite set of stick breaking random variables $V_k$ [Set94] according to

$$\pi_k = V_k \prod_{i=1}^{k-1} (1 - V_i).$$  \hspace{1cm} (2)

The stick breaking variables are distributed independently according to $V_k \sim \text{Beta}(1, \alpha)$, where $\alpha > 0$ is the concentration parameter of the Dirichlet Process. When $\alpha$ is small, there is a bias towards a small number of large mixing weights (clusters), and when it is large there is a tendency to have many small weights. The mixing weights are guaranteed to sum to one, as required to make a well-defined mixture model.

It is convenient to introduce a set of auxiliary assignment variables $Z = \{z_1, \ldots, z_N\}$, one for each data point $x_i$. $z_i \in \mathbb{N}$ designates the mixture component that generated data point $x_i$. The assignment variables $Z$ specify a clustering or partition of the data.

In learning, we are interested in estimating the posterior $p(Z, \Phi, V | X, \alpha, H)$ given a set of observations $X = \{x_1, \ldots, x_N\}$. We assume that the component parameter prior $H$ and concentration parameter $\alpha$ are known.

### 3.2 Exponential Family and Sufficient Statistics

We will restrict ourselves to component distributions that are members of the exponential family [BS94], because they have a number of well known properties that admit efficient inference algorithms. Exponential family distributions have the form:

$$p(x|\phi) = g(x) \exp\{\phi^T F(x) + a(\phi)\},$$  \hspace{1cm} (3)

where $F(x)$ is a fixed length vector sufficient statistic, $\phi$ is the natural parameter vector, and $a(\phi)$ is a scalar valued function of $\phi$ that ensures that the distribution normalizes to 1. The exponential family includes the Gaussian, Multinomial, Beta, Gamma, and other common distributions.

We also require an additional restriction that the component prior distribution $H$ be conjugate to the component distributions [BS94]. It must be of the form:

$$H = p(\phi|\nu, \eta) = h(\eta, \nu) \exp\{\phi^T \nu + \eta a(\phi)\}. \hspace{1cm} (4)$$

$\eta$ and $\nu$ are the natural parameters for the conjugate prior distribution.

The following fact is fundamental to our approach: If a set of observations $X$ are all assigned to the same mixture component ($z_i = k$ for all $i$ such that $x_i \in X$), then the posterior distribution of the component parameter $\phi_k$ is determined by

$$S = \sum_{x_i \in X} F(x_i),$$  \hspace{1cm} (5)
which is the sum of the sufficient statistic vectors of each observation $x_i \in X$. The significance of this fact is that if a set of assignment variables are constrained to be equal (i.e., their corresponding observations are assumed to be generated by the same mixture component), their inferential impact can be fully summarized by $S$, a vector whose length does not increase with the number of observations.

4 Existing Approaches

We briefly review existing approaches for online learning of Bayesian Mixture Models. Existing approaches have in common that they explicitly consider a number of alternative clusterings or mixture models in parallel, and update each of these hypotheses independently as new data arrives.

4.1 Online Variational Bayes

Sato [Sat01] derives recursive update rules for Variational Bayesian learning of mixture models. The alternative models are stored in memory, and each data point is discarded after it is used to update each parallel hypothesis. A “forgetting factor” is used in order to decay the contribution of “old” data points, since they are likely to be incorrectly assigned to components. Empirically, the forgetting factor means that much more data is needed to learn a model when compared with a batch technique. This makes the forgetting factor undesirable from the standpoint of our requirement to have an incremental algorithm that outputs results substantially close to the results that a batch algorithm would output given the total data seen. Finally, model parameters must be stored for each alternative hypothesis, and this becomes prohibitively expensive as the number of models increases.

4.2 Particle Filters

Fearnhead [Fea04] developed a particle filter learning algorithm for the DPMM. This approach approximates $p(Z^T | X^T)$ with a set of $M$ weighted particles (clustering hypotheses). Upon arrival of a new data point, the $M$ particles are extended to include a new assignment $z_{T+1}$ and none of the assignments for the previous observations change. In order to prevent combinatorial explosion over time, only $M$ of these descendant particles are retained. In our experiments, this approach behaves poorly for large datasets. Unseen observations can have a drastic effect on the relative rankings of the assignments $Z^T$. The algorithm greedily keeps only the top ranked clusterings at time $T$, and those that it discards can never be considered in the future. No two particles are identical, but the assignments tend to vary from one another for only a small number of data points and so do not cover a wide enough set of hypotheses.
Figure 2: A schematic depiction of a two-dimensional clustering problem. Alternative clustering solutions are displayed on their own coordinate axes, and the model clusters are represented by green ellipses. The set of clump constraints consistent with all solutions are displayed as red ellipses. The implicit set of clustering solutions under consideration are those that can be composed of the clumps, which is much larger than other online clustering methods that explicitly enumerate alternative clustering hypotheses.

5 Our Approach

We observe that the chief difficulty with existing approaches is that they must explicitly enumerate and update a very large number of alternative clusterings in order to produce accurate results (the number of potential clusterings of $N$ points grows as the Bell number $B_N$). We wish to avoid this explicit enumeration, while at the same time keeping a large number of alternatives alive for consideration. Our approach must also require bounded time and space requirements to produce an update given new data: the computational requirements must not scale with the total number of data seen.

Figure 2 shows how multiple clustering hypotheses can be combined into a single set of assignment constraints. Rather than explicitly fixing the assignments in each parallel branch, the constraints now take the form of points that are grouped together in every alternative. We will call these groups of points “clumps”. We define sets of indices $C_s$ such that if $i \in C_s$ and $j \in C_s$ for some $s$, then data points $x_i$ and $x_j$ are assigned to the same component in all of the alternative clusterings. The sets $C_s$ are disjoint, meaning that no data point can exist in more than one clump. The collection of clumps $C$ is the partition with the fewest number of sets that can compose each of the alternative clus-
Figure 3: Top down compression: Clump constraints are discovered by recursive splitting in a top down fashion. This process is well suited to discovering groups of points that are likely to belong to the same model cluster across multiple plausible clustering alternatives.

tering hypotheses. In the language of lattice theory, the clump partition $C$ is the greatest lower bound or infimum of the alternative clustering hypotheses. A similar scheme was pursued in [BFR98] for scaling the k-means algorithm (where the number of clusters is assumed to be known) to large databases.

A single optimization procedure done under the clump constraints will yield the best clustering mode (modulo local minima issues) that is compatible with the implicit ensemble of alternatives inherent in the constraints. The implicit ensemble of alternatives is very large: it is composed of every possible grouping of the clumps, and is much larger than could be explicitly modeled.

This raises the question: How can these clump constraints be computed without first explicitly computing a number of plausible solutions? We observe that alternative models, while distinct, have considerable redundancy. The reason is that the clustering of data points in one region of space has little impact on the clustering assignments of data in a distant part of space. Any two alternatives will tend to vary from one another only for a subset of data points. Our approach is to partition the clustering problem into a series of independent subproblems. This is carried out in a top down fashion, as illustrated in Figure 3. This forms a tree of possible groupings of data, and the bottom level of this tree defines our clump constraints. Variational Bayes techniques provide a convenient framework for carrying out this procedure (see Section 5.2).

Our algorithm processes data in small batches which we refer to as epochs, each one of which contains $E$ data points. We first compute the current best estimate mixture model as described in Section 5.1. Then we carry out a compression phase (explained in Section 5.2) in which clump constraints are computed in a top down recursive fashion, and this phase halts when a stopping criterion is met. Data points that belong to the same clump are summarized by their average sufficient statistics (see Section 3.2), and the $E$ individual data points are discarded. The clumps are each given an assignment variable $z_s$ and can be treated in the same way as data points in the next round of learning. We bound the computational time and space requirements in each learning round by controlling the number of clumps discovered during the compression phase.
Figure 4: A sketch of the algorithm. Data arrives in small batches or epochs, and the current best estimate mixture model is computed in the model building phase (here, the model is represented by green ellipses). Next, clump constraints are computed in the compression phase, and summarized by their associated sufficient statistics (represented by red ellipses). The summarized data are discarded and the process continues.

The algorithm is summarized in Figure 4.

5.1 Model Building Phase

Learning rounds begin by computing a current best estimate mixture model using Variational Bayes (VB) [Att99]. In the Variational Bayes approach, intractable posterior distributions are approximated with simpler proxy distributions that are chosen so that they are tractable to compute. Blei and Jordan [BJ05] extended this technique to the DPMM.

Given the observed data \(X^T\), the batch VB algorithm optimizes the variational Free Energy functional:

\[
\mathcal{F}(X^T; q) = \int_{dW} q(V, \Phi, Z^T) \log \frac{p(V, \Phi, Z^T, X^T | \eta, \nu, \alpha)}{q(V, \Phi, Z^T)},
\]

which is a lower bound on the log-evidence \(\log p(X^T | \eta, \nu, \alpha)\). The proxy distributions

\[
q(V, \Phi, Z^T) = \prod_{k=1}^{K} q(V_k; \xi_{k,1}, \xi_{k,2}) q(\phi_k; \zeta_{k,1}, \zeta_{k,2}) \prod_{t=1}^{T} q(z_t)
\]

are products of beta distributions for the stick-breaking variables (with hyperparameters \(\xi\)), component distributions (with hyperparameters \(\zeta\)), and assignment variables, respectively. Update equations for each proxy distribution can be cycled in an iterative coordinate ascent and are guaranteed to converge to a local maximum of the free energy. The true DPMM posterior allows for an infinite number of clusters, but the proxy posterior limits itself to \(K\) components. Kurihara et al. [KWV07] showed that \(K\) can be determined by starting with a single component, and repeatedly splitting components as long as the free energy bound \(\mathcal{F}(X^T; q)\) improves.
Like the batch approach, our algorithm optimizes $F(X^T; q)$ during model building, but this optimization is carried out under the clump constraints discovered during previous learning rounds. The resulting Free Energy bound is a lower bound on the optimal batch solution. (In practice, the batch process itself may not achieve the optimal bound because of local optima issues.) Formally this can be expressed as:

Property 1. The MB-VDP model-building phase optimizes $F(X^T; q)$ subject to the constraints that $q(z_i) = q(z_j)$ for all $i \in C_s$ and all $j \in C_s$ and all clump constraints $C_s$. The resulting solution lower bounds the optimal batch solution: $\max_q F_{MB}(X^T; q) \leq \max_{q'} F(X^T; q')$.

The bound follows because solutions to the constrained problem are in the space of feasible solutions of the unconstrained optimization problem. Hyperparameter update equations that optimize the constrained Free Energy $F_{MB}$ are:

\[
\xi_{k,1} = 1 + \sum_s |C_s| q(z_s = k) \\
\xi_{k,2} = \alpha + \sum_s |C_s| \sum_{j=k+1}^K q(z_s = j) \\
\zeta_{k,1} = \eta + \sum_s |C_s| q(z_s = k) \langle F(x) \rangle_s \\
\zeta_{k,2} = \nu + \sum_s |C_s| q(z_s = k) \\
\]

where we define
\[
S_{sk} = E_{q(V, \phi_k)} \log \{p(z_s = k|V)p(\langle F(x) \rangle_s|\phi_k)\} \\
\langle F(x) \rangle_s = \frac{1}{|C_s|} \sum_{i \in C_s} F(x_i)
\]

which (critically) depend only on $\langle F(x) \rangle_s$, the sufficient statistics of the points in each clump.

After executing the update equations for $q(z_s = k)$, the constrained Free Energy may be expressed in the following form:

\[
F_{MB}(X^T; q) = F_{\text{likelihood}}(X^T; q(V), q(\Phi)) - F_{\text{complexity}}(q(V), q(\Phi)).
\]

We find that $F_{MB}$ decomposes into a likelihood term that measures the extent to which the current model fits the compressed data:

\[
F_{\text{likelihood}} = \sum_s n_s \log \sum_{k=1}^K \exp(S_{sk})
\]
as well as a complexity penalty

$$F_{\text{complexity}} = \sum_{k=1}^{K} KL(q(v_k)||p(v_k|\alpha)) + \sum_{k=1}^{K} KL(q(\phi_k)||p(\phi_k|\lambda))$$ (13)

which penalizes models according to the Kullback-Leibler divergence between the model parameters’ proxy posterior distributions and their respective prior distributions. The complexity penalty increases with the number of clusters $K$ expressed by the current model. Intuitively, the constrained Free Energy balances the goal of finding models that explain the observed data while preventing overfitting with overly complex models.

The constrained Free Energy $F_{MB}$ was first given in Kurihara et al. [KWV07], in which DPMM learning is augmented with a kd-tree in order to speed up inference (also [VNV06] for EM learning). Sufficient statistics of data points were cached at nodes of the kd-tree and used to perform approximate inference. Our approach differs from these algorithms in several ways. We do not use a kd-tree to compute clump constraints but instead build a tree by greedily splitting collections of data points according to a Free Energy-based cost function, as discussed in the next section. We process data in sequential rounds and recompute clump constraints after each round. We irreversibly discard individual data points that are summarized by clump statistics in order to maintain storage costs below a pre-assigned bound, whereas [KWV07] and [VNV06] always have the option of working with individual data points if it leads to improvement in a Free Energy bound.

### 5.2 Compression Phase

The goal of the compression phase is to identify groups of data points that are likely to belong to the same mixture component, no matter the exact clustering behavior of the rest of the data. Once these groups are summarized by their sufficient statistics, they are irreversibly constrained to have the same assignment distribution during future learning rounds. Therefore we must take into account unseen future data when making these decisions in order to avoid locking into suboptimal solutions. We must find collections of points that are not only likely to be assigned to the same component given the first $T$ data points, but also at some target time $N$, with $N \geq T$.

We estimate this future clustering behavior by using the empirical distribution of data seen so far (up to time $T$) as a predictive distribution for future data:

$$\hat{p}(x_{T+1}, \cdots, x_N) = \prod_{i=T+1}^{N} \frac{1}{T} \sum_{t=1}^{T} \delta(x_i - x_t)$$ (14)

and define the following modified Free Energy

$$F_C(X^T; r) = E_{\hat{p}(x_{T+1}, \cdots, x_N)} F_{MB}(X^N; r)$$ (15)
by taking the expectation of the constrained Free Energy $F_{MB}$ over the unobserved future data. We also define a new proxy distribution $r(V, \Phi, Z^N)$ used during the compression phase, which is identical in form to Eq. 7 estimated during the Model Building phase.

**Proposition 1.** Iteration of the following parameter update equations results in convergence to a local maximum of $F_c$:

$$
\xi_{k,1} = 1 + \frac{N}{T} \sum_s |C_s| r(z_s = k) \\
\xi_{k,2} = \alpha + \frac{N}{T} \sum_s |C_s| \sum_{j=k+1}^K r(z_s = j) \\
\zeta_{k,1} = \eta + \frac{N}{T} \sum_s |C_s| r(z_s = k) \langle F(x) \rangle_s \\
\zeta_{k,2} = \nu + \frac{N}{T} \sum_s |C_s| r(z_s = k) \\
S_{sk} = E_{r(V, \phi_k)} \log \{ p(z_s = k|V) p(\langle F(x) \rangle_s | \phi_k) \} \\
r(z_s = k) \sim \exp(S_{sk}).
$$

After performing the updates for $r(z_s = k)$, it holds that:

$$
F_C = \left( \frac{N}{T} \right) F_{\text{likelihood}}(X^T, r(V), r(\Phi)) - F_{\text{complexity}}(r(V), r(\Phi)).
$$

The above update equations differ from those in the model building phase by a data magnification factor $\frac{N}{T}$. The indices $s$ range over the current clump constraints; we need not compute assignment distributions $r(z_s = k)$ for unobserved future data. We also find that the compression phase objective can be interpreted as re-scaling the data likelihood term by $\frac{N}{T}$.

As indicated in Figure 3, we compute clump constraints in a top down fashion. We start the process with the clustering estimate determined during the preceding model building phase; that is, $r(z_s = k) = q(z_s = k)$. We then evaluate splitting each partition $k$ by first splitting it along the principal component defined by the clumps in the partition. We then iterate the update equations (Eqs. 16) in order to refine this split. Each potential partition split is then ranked according to the resulting change in $F_C$ (Eq. 15). We then greedily choose the split that results in the largest change. The process repeats itself, until a halting criterion is met (see below). We update the clump constraints according to $C_l = \{ s : \text{argmax}_k r(z_s = k) = l \}$.

**Property 2.** The maximum attainable Free Energy during the MB-VDP model building phase increases monotonically with the number of clump constraints discovered during the compression phase.

The reasoning is similar to Property 1. Each time the compression phase splits an existing partition into two, the space of feasible solutions in the model
Algorithm 1 Memory Bounded Variational DPMM

while There is more data do
    Model building phase according to sec. 5.1
    Initialize compression phase: \( r(z_s = k) = q(z_s = k) \)
    while \( MC < M \) (eq. 18) do
        for \( k = 1 \) to \( K \) do
            Split partition \( k \) and refine (eqs. 16)
            \( S(k) = \Delta F_C \) (change in eq. 17)
        end for
        Split partition \( \arg \max_k S(k) \)
        \( K = K + 1 \)
    end while
    \( C_l = \{ s : \arg \max_k r(z_s = k) = l \} \)
    Retain clump statistics \( \langle F(x) \rangle_l \) into next round
    Discard summarized data points
end while

building optimization problem has been increased, but the previous set of solutions (all data in the two new partitions constrained to have equal assignment distributions) is still available. Therefore, the maximum attainable Free Energy cannot decrease.

We must restrict the number of clumps that are retained in order to ensure that the time and space complexity is bounded in the next round of learning. A stopping criterion determines when to halt the top down splitting process. A number of criteria are possible, depending on the situation.

When learning DPMMs with full-covariance Gaussian components, each clump requires \( \frac{d^2 + 3d}{2} + 1 \) values to store sufficient statistics (mean, symmetric covariance matrix, and number of data points summarized). It is convenient to express the stopping criterion as a limit on the amount of memory required to store the clumps. From this perspective, it makes sense to replace a clump with its sufficient statistics if it summarizes more than \( \frac{d + 3}{2} \) data points. If a clump summarizes fewer points, then the individual data points are retained instead. We refer to these individual retained data points as singlets. The clump memory cost for mixture models with full covariance matrices is therefore

\[
MC = \left( \frac{d^2 + 3d}{2} + 1 \right) N_c + d N_s, \tag{18}
\]

where \( N_c \) is the number of clumps and \( N_s \) is the number of singlets. An upper limit on clump memory cost \( M \) is defined, and the compression phase halts when \( MC \geq M \).

The MB-VDP algorithm is summarized in Algorithm 1. The time required for the algorithm to learn the entire data set is typically less than the batch variational DPMM approach outlined in [KWV07]. This is because full variational updates in the batch procedure require \( O(KN) \), where \( K \) is the number of
clusters and $N$ is the number of data points. The MB-VDP algorithm requires only $O(K(N_c + N_s + E))$ for an iteration during the model building phase.

We have implemented a number of measures in the compression phase in order to reduce computational overhead. The first is to hard assign clumps to partitions, i.e., $r(z) = \delta(z - a_s)$ where $a_s = \arg\max_k S_{sk}$, rather than maintaining full assignment distributions. The second is to refine split partitions by optimizing $F_c$ restricted only to the data and parameters associated with the partition, rather than performing complete updates with all data and parameters (this also allows us to cache the candidate partition splits rather than re-computing them during each iteration). When these speed up heuristics are in place, the compression phase can no longer be interpreted as optimization of $F_C$, however experimental results in Section 6 show that the algorithm performs well and that the time required during the compression phase is quite modest when compared to the model building phase.

Vasconcelos and Lippman [VL99] learn a hierarchy of EM mixture model solutions using a bottom up procedure (although they did not investigate this approach in the context of incremental learning). We find that a bottom up approach to learn clump constraints is inappropriate in our situation. Variational updates for the DPMM are sensitive to initial conditions, and our top down method sidesteps this initialization problem.

Our implementation of MB-VDP may be found at: http://vision.caltech.edu/~gomes.

6 Experimental Results

We test our algorithm with three experiments. The first experiment compares our algorithm against the particle filter in [Fea04] on a small image clustering task of four categories from Caltech 256. The second experiment compares our algorithm against [KWV07] on the larger MNIST digit dataset. Finally, we demonstrate our approach on $330K$ image patches from the Corel image database, which was too large for the batch approach.

The first set of experiments compares the performance of our method with that of Fearnhead’s particle filter. The data set consists of four categories (Airplanes, Motorbikes, Faces, and T-Shirts) from Caltech 256 [GHP07] that are projected to a 20 dimensional feature space using Kernel PCA with the Spatial Pyramid Match Kernel of Lazebnik et al. [LSP06]. There are 1400 data points (images) in total. The hyperparameters for Normal Inverse Wishart prior on cluster parameters $(H)$ were chosen by hand, based on prior knowledge about the scale of the data, and the concentration parameter $\alpha$ was set to 1. The batch algorithm tends to find 12 to 15 clusters in this setting. The clusters discovered respect the categories, that is, very few objects from different classes are clustered together. This was tested by assigning labels to clusters by looking at five examples from each. Images from the training set were classified according to the label of the cluster with highest responsibility. Average classification performance was 98%. However, the algorithm divides each category into sub-
categories according to perceptually relevant differences. Figure 5 shows some example images from six of the discovered clusters.

The algorithms were judged quantitatively according to predictive likelihood. 1300 of the 1400 images were chosen at random as a training set, and the algorithm is trained on a complete pass through the data in random order. The average likelihood of the remaining data points was computed as a measure of generalization performance. The particle filter was tested at different numbers of particles. The amount of memory was varied for our algorithm. In our algorithm, the memory value represents the memory required to store both the clumps from earlier rounds of memory and the current small batch of points. In all cases, the data epoch size $E$ are chosen to be one-half of the memory size, so for an effective memory of 200, the algorithm progresses through the data in epochs of 100 points. Note that at memory of 200, the algorithm is unable to store all 12 to 15 clusters inherent in the data.

Table 1 shows the performance of the particle filter, and Table 2 shows the
<table>
<thead>
<tr>
<th>Particles</th>
<th>Ave Predictive Log-Likelihood</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>4.99 ± 0.34</td>
<td>5.94 × 10² s</td>
</tr>
<tr>
<td>1000</td>
<td>5.43 ± 0.28</td>
<td>2.856 × 10³ s</td>
</tr>
<tr>
<td>10000</td>
<td>5.80 ± 0.22</td>
<td>2.484 × 10⁴ s</td>
</tr>
</tbody>
</table>

Table 1: Particle filtering predictive performance and runtime

<table>
<thead>
<tr>
<th>Memory</th>
<th>Ave Predictive Log-Likelihood</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>6.37 ± 0.32</td>
<td>73.3 s</td>
</tr>
<tr>
<td>400</td>
<td>6.93 ± 0.32</td>
<td>57.08 s</td>
</tr>
<tr>
<td>600</td>
<td>6.99 ± 0.31</td>
<td>57.76 s</td>
</tr>
</tbody>
</table>

Table 2: MB-VDP predictive performance and runtime. Batch performance was 7.04 ± 0.28 with runtime 71.4s on 1300 data points.

performance of our algorithm. Our algorithm beats the particle filter in terms of generalization accuracy at all parameter values. Our algorithm produces generalization results that are close to the performance of the batch algorithm. The runtime advantage of our approach is very significant over that of the particle filter.

In the second experiment, our approach is compared against the batch algorithm of [KWV07]. The 60000 hand-written digits from the MNIST training set were reduced to 50 dimensions using PCA in a preprocessing step. Our algorithm was set to have a memory size equivalent to 6000 data points, which is an order of magnitude smaller than the size of the data set. Our algorithm processes data in epochs of 3000.

The second row of Figure 1 shows the cluster means discovered by our algorithm as it passes through more data. Since the DPMM is nonparametric, the model complexity increases as more data is seen. The bottom row of Figure 1 shows the cluster centers discovered by our approach after processing the entire data set compared to those produced by the batch algorithm. The clusters are qualitatively quite similar, and the two algorithms discover a comparable number of clusters (88 for the batch approach, 90 for our algorithm).

The run time for the batch algorithm was 31.5 hours, while for our approach it was 20 hours for a complete pass through. Note that we can likely achieve greater speedup by initializing each learning round with the previous round’s model estimate and using a split-merge procedure [UNGH99], although we did not pursue this here. We compare the free energy bounds produced by the two approaches. The ratio of these two values is 0.9756 meaning that our incremental algorithm produces a slightly worse lower bound on the likelihood. Our approach is more accurate than the kd-tree accelerated algorithm in [KWV07] which produced a free energy ratio of 0.9579 relative to the standard batch approach. Recognition was performed on 10000 MNIST test digits, in the same way as the Caltech 4 dataset but labels were assigned by observing only the cluster means.
Performance for the incremental algorithm was 88.5% and 91.2% for batch. Note that this approach only requires labeling of approximately 90 images, compared to 60000 training labels used by traditional approaches.

Finally, we demonstrate our algorithm on a clustering task of 330,000 7 pixel by 7 pixel image patches from the Corel image database. We preprocess the data by discarding patches with standard deviation below a threshold, and normalize all remaining patches to unit norm. We use Gaussian components with diagonal covariance matrices. The batch approach in [KWV07] was unable to cluster this data due to memory requirements. We use an effective memory size of 30000 data points. Cluster centers are shown in Figure 6 after 30K, 150K, and 330K patches were processed. As expected, the model complexity increases as more data is processed and the clusters represent greater diversity in the data. The total memory required by the incremental algorithm was 109 MB to store the best estimate model, the clumps, and the responsibilities. In contrast, the batch approach would require 773 MB. The incremental algorithm required approximately 2 hours per epoch of 15000 data points. Again this could be substantially reduced by initializing each round with the previous estimate, rather than beginning from scratch each time.

7 Discussion and Conclusions

We have introduced an incremental clustering algorithm with a number of favorable properties. The key idea (summarized by Figure 3) is to find clustering arrangements (clumps) that alternative models are likely to have in common, rather than to explicitly enumerate and independently update a set of alterna-
tives. This idea leads to an algorithm that outperforms other online approaches in terms of run time and accuracy, and is suitable for use on large datasets. Our algorithm’s nonparametric Bayesian framework allows for automatic determination of the number of clusters, and model complexity adjusts as more data is acquired. Future work includes extending these lessons to build systems capable of learning complex object categories incrementally and with little human supervision.

References


