Incremental Learning of Nonparametric Bayesian Mixture Models

Ryan Gomes* Max Welling† Pietro Perona*

*Dept. of Computation and Neural Systems
California Institute of Technology
Pasadena, CA 91125 USA
{gomes,perona}@vision.caltech.edu

†Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92697 USA
welling@ics.uci.edu

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.

1. Introduction

Discovering visual categories automatically with minimal human supervision is perhaps the most exciting current challenge in machine vision [21, 16]. 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 [13, 20, 5, 7, 10]. 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 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

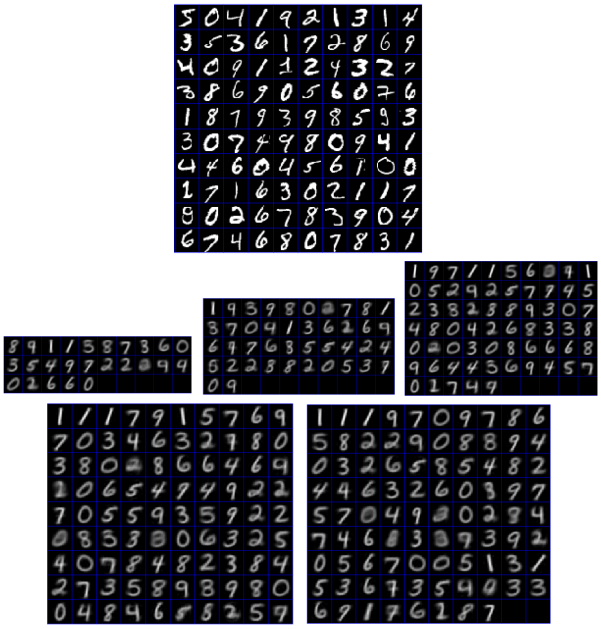

Figure 1. Top: A sample of the inputs to the incremental learning process. Middle: Cluster means discovered by incremental algorithm after 6000, 12000, and 30000 digits processed. As expected, the model complexity increases as data arrives. The computational burden per model update is not a function 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 4 for a description of the algorithm and section 5 for more information about the experimental results.

978-1-4244-2243-2/08/$25.00 ©2008 IEEE

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 nonparametric 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 2 provides background on the Dirichlet Process mixture model and sufficient statistics. Section 3 briefly describes existing approaches to the problem and section 4 explains our approach. Section 5 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 6.

2. Background

We start by reviewing the Dirichlet Process mixture model (DPMM) [1].

2.1. 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:

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

where ϕ_k are component parameter vectors and π_k are a set of mixing weights that sum to 1. During inference, the mixing weights and the component parameter vectors are treated as random quantities. A probabilistic structure known as the Dirichlet Process [8] defines a prior on these random variables.

The component parameters ϕ_k are assumed to be independent samples from a probability distribution H. The mixture weights π_k may be constructed from a countably infinite set of *stick breaking* random variables V_k [15] according to

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

The stick breaking variables are distributed independently according to $V_k \sim Beta(1, \alpha)$, where $\alpha > 0$ is the concentration parameter of the Dirichlet Process. When α 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_t. $z_t \in \mathbb{N}$ designates the mixture component that generated data point x_t. 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 α are known.

2.2. Exponential Family and Sufficient Statistics

We will restrict ourselves to component distributions that are members of the exponential family [3], because they have a number of well known properties that admit efficient inference algorithms. Exponential family distributions have the form:

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

where $F(x)$ is a fixed length vector *sufficient statistic*, ϕ is the *natural parameter* vector, and $a(\phi)$ is a scalar valued function of ϕ 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 [3]. It must be of the form:

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

η and ν 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 ϕ_k is determined by

$$S = \sum_{x_i \in X} F(x_i), \tag{5}$$

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.

3. 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.

3.1. Online Variational Bayes

Sato [14] 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.

3.2. Particle Filters

Fearnhead [6] 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.

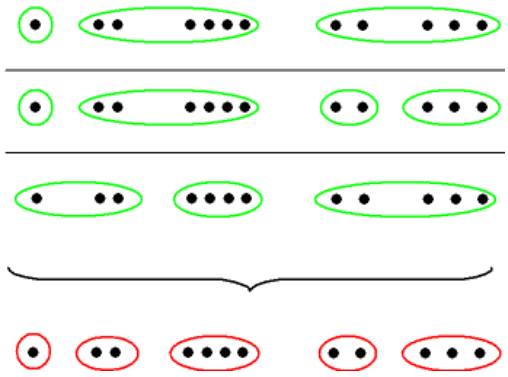

Figure 2. A schematic depiction of a one-dimensional clustering problem. Alternative solutions are displayed in green. The set of clump constraints consistent with all solutions are shown in red.

4. 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 clustering hypotheses. In the language of set theory, the set of clumps C is the greatest lower bound or infimum of the alternative clustering hypotheses.

A single optimization procedure done under the clump constraints will yield the best clustering mode (modulo lo-

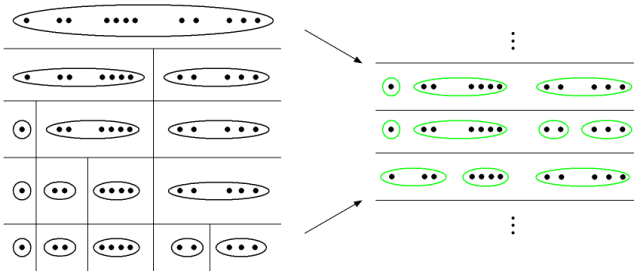

Figure 3. On the left, the problem is decomposed into (approximately) independent subproblems. The bottom level represents the set of clump constraints consistent with all identifed groups. On the right, the implicit ensemble of solutions that can be composed using the clumps.

cal 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 fig. 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.

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 4.1. Then we carry out a compression phase (explained in section 4.2) 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 sec. 2.2), 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. The algorithm is summarized in figure 4.

4.1. Model Building Phase

Learning rounds begin by computing a current best estimate mixture model using Variational Bayes (VB) [2]. 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 [4] extended this technique to the DPMM.

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

$$\mathcal{F}(X^T; q) = \int_{dW} q(V, \Phi, Z^T) \log \frac{p(V, \Phi, Z^T, X^T | \eta, \nu, \alpha)}{q(V, \Phi, Z^T)}, \tag{6}$$

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

$$q(V, \Phi, Z^T) = \prod_{k=1}^{K} q(V_k; \xi_{k,1}, \xi_{k,2}) q(\phi_k; \zeta_{k,1}, \zeta_{k,2}) \prod_{t=1}^{T} q(z_t) \tag{7}$$

are products of Beta distributions for the stick breaking variables (with hyperparameters ξ), component distributions (with hyperparameters ζ), 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*. [11] 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 $\mathcal{F}(X^T; q)$ 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:

Proposition 1 *The MB-VDP model-building phase optimizes $\mathcal{F}(X^T; q)$ 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: $\mathcal{F}_{MB}(X^T; q) \leq \max_q \mathcal{F}(X^T; q)$.*

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 Free Energy under the clump constraints can be derived that depend only on the sufficient statistics of the points in each

clump. The equations are given in [11] as equations 16-18, and they are omitted here for reasons of space. Kurihara *et al*. augment DPMM learning with a kd-tree in order to speed up inference (also [19] 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 [11] and [19] always have the option of working with individual data points if it leads to improvement in a Free Energy bound.

4.2. 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 scaling the current data sample to the target size N. This is equivalent to using the empirical distribution of the data seen at time T as a predictive model of the unseen future data. The following modified Free Energy is used during the compression phase:

$$\begin{aligned}\mathcal{F}_C = & -\sum_{k=1}^{K} KL(q(v_k)||p(v_k|\alpha)) - \sum_{k=1}^{K} KL(q(\phi_k)||p(\phi_k|\lambda)) \\ & + \frac{N}{T}\sum_s n_s \log \sum_{k=1}^{K} \exp(S_{sk})\end{aligned} \tag{8}$$

This is similar to the Free Energy given in [11] but modified with a data magnification factor $\frac{N}{T}$. The corresponding update equations for this cost function are

$$\begin{aligned}\xi_{k,1} &= 1 + \frac{N}{T}\sum_s n_s q(z_s = k) \\ \xi_{k,2} &= \alpha + \frac{N}{T}\sum_s n_s \sum_{j=k+1}^{K} q(z_s = j) \\ \zeta_{k,1} &= \eta + \frac{N}{T}\sum_s n_s q(z_s = k)\langle F(x)\rangle_s \\ \zeta_{k,2} &= \nu + \frac{N}{T}\sum_s n_s q(z_s = k) \\ q(z_s = k) &\sim \exp(S_{sk}) \\ S_{sk} &= E_{q(V,\phi_k)} \log\{p(z_s = k|V)p(\langle F(x)\rangle_s|\phi_k)\}\end{aligned} \tag{9}$$

where $\langle F(x)\rangle_s$ and n_s are the average sufficient statistics and number of data points summarized in clump s, respectively. These update rules also differ from those used during model building (eqs. 16-18 in [11]) by the data magnification factor $\frac{N}{T}$. However, the equations for the responsibilities $q(z_s)$ are unchanged, and we need not compute responsibilities for unseen future data.

As indicated in fig. 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, which is given by the assignment distributions $q(z_s = k)$. We then hard assign each clump or data point to the partition with the highest responsibility:

$$r_s = \arg\max_k q(z_s = k). \tag{10}$$

The variables r_s indicate which partition the clump (or data point) s belongs to in the compression process. We then proceed through each partition and split it along the principal component defined by the clumps in the partition. We iterate the update equations (eqs. 9) for the points in the partition in order to refine this split. Note that these updates involve only the clumps in the partition, and they may be assigned only to the two subpartitions. After this update process converges, the clumps are then hard assigned to one of the candidate subpartitions.

Each potential partition split is then ranked according to the resulting change in the data magnified Free Energy (eq. 8). We then greedily choose the split that results in the largest change. The process then repeats itself, with the new partitions ranked in the same way described above. We cache the results of each split evaluation in order to prevent redundant computation.

Proposition 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 Proposition 1. Each time the Compression Phase increases the number of clumps by

while *There is more data to collect* **do**
  Collect E data points from the world;
  Model building phase according to sec. 4.1;
  Initialize compression phase (eq. 10);
  while $MC < M$ *(eq. 11)* **do**
    for $k = 1$ *to* K **do**
      if $evaluated(k) = FALSE$ **then**
        Split partition k and refine (eqs. 9);
        $\Delta BFE(k) =$ change in eq. 8;
        $evaluated(k) = TRUE$;
      end
    end
    Split partition $\arg\max_k \Delta BFE(k)$;
    $K = K + 1$;
  end
  Retain clumps into next round;
  Discard summarized data points;
end

Figure 4. Memory Bounded Variational DPMM

one, it is because a previously existing partition has been split into two. The space of feasible solutions in the Model 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 can not 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 DPMM's 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

$$MC = \left(\frac{d^2 + 3d}{2} + 1\right) N_c + dN_s, \qquad (11)$$

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 CB-VDP algorithm is summarized in figure 4. The time required for the algorithm to learn the entire data set is typically less than the batch variational DPMM approach outlined in [11]. 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 CB-VDP algorithm requires only $O(K(N_c+N_s+E))$ for an iteration during the model building phase. The time required during the compression phase is quite modest when compared to the model building phase, because the compression phase only entails restricted variational updates that involve subsets of the data.

Vasconcelos and Lippman [18] 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`

5. Experimental Results

We test our algorithm with three experiments. The first experiment compares our algorithm against the particle filter in [6] on a small image clustering task of four categories from Caltech 256. The second experiment compares our algorithm against [11] 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 [9] that are projected to a 20 dimensional feature space using Kernel PCA with the Spatial Pyramid Match Kernel of Lazebnik *et al*. [12]. 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 α 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

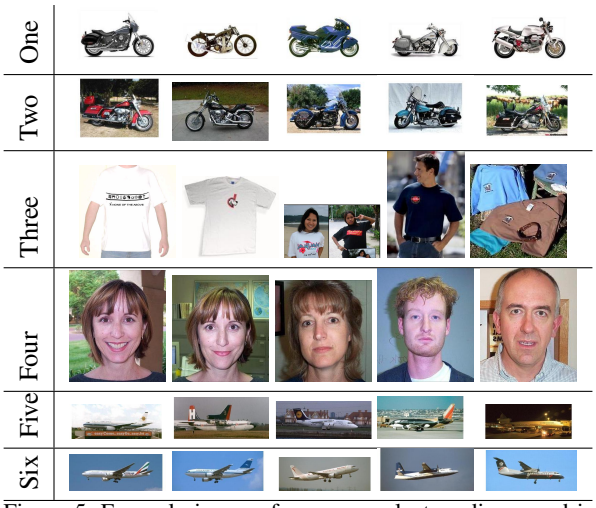

Figure 5. Example images from some clusters discovered in the T-Shirts, Airplanes, Faces, and Motorbike categories from Caltech 256. The clusters typically do not mix images from different categories and the algorithm discovers relevant distinctions within categories. For example, the Airplanes category is split into airplanes in the sky and on the ground, and the Motorbikes category is split into segmented motorbikes and motorbikes in clutter.

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 remaing 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 progesses 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 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 [11]. 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).

Particles	Ave Predictive Log-Likelihood	Runtime
100	4.99 ± 0.34	9.9 min
1000	5.43 ± 0.28	47.6 min
10000	5.80 ± 0.22	6.9 hr

Table 1. Particle filtering predictive performance and runtime.

Memory	Ave Predictive Log-Likelihood	Runtime
200	6.37 ± 0.32	73.3 s
400	6.93 ± 0.32	57.08 s
600	6.99 ± 0.31	57.76 s

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

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 [17], 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 [11] 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 [11] 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

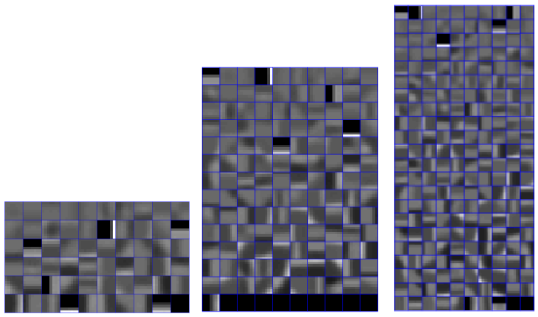

Figure 6. Cluster centers from the Corel patch experiment after 30K, 150K, and 330K patches.

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.

6. Discussion and Conclusions

We have introduced an incremental clustering algorithm with a number of favorable properties. The key idea (summarized by fig. 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 alternatives. 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.

7. Acknowledgements

This material is based on work supported by the National Science Foundation under grant numbers 0447903 and 0535278, the Office of Naval Research under grant numbers 00014-06-1-0734 and 00014-06-1-0795, and The National Institutes of Health Predoctoral Training in Integrative Neuroscience grant number T32 GM007737.

References

[1] C. Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The Annals of Statistics- Institute of Mathematical Statistics*, 1974.

[2] H. Attias. A variational baysian framework for graphical models. In *NIPS*, pages 209–215, 1999.

[3] J. M. Bernardo and A. F. M. Smith. *Bayesian Theory.* Wiley, 1994.

[4] D. M. Blei and M. I. Jordan. Variational inference for dirichlet process mixtures. *Journal of Bayesian Analysis*, 1(1):121–144, 2005.

[5] G. Dorko and C. Schmid. Selection of scale-invariant parts for object class recognition. In *International Conference on Computer Vision*, pages 634–640, 2003.

[6] P. Fearnhead. Particle filters for mixture models with an unknown number of components. *Journal of Statistics and Computing*, 14:11–21, 2004.

[7] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *CVPR*, pages 524–531. IEEE Computer Society, 2005.

[8] T. S. Ferguson. A bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1973.

[9] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007.

[10] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. In *ICCV*, pages 604–610. IEEE Computer Society, 2005.

[11] K. Kurihara, M. Welling, and N. Vlassis. Accelerated variational dirichlet process mixtures. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR (2006)*, pages 2169–2178, 2006.

[13] T. Leung and J. Malik. Recognizing surfaces using three-dimensional textons. In *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV-99)*, volume II, pages 1010–1017, Los Alamitos, CA, Sept. 20–27 1999. IEEE.

[14] M. Sato. Online model selection based on the variational bayes. *Neural Computation*, 13(7):1649–1681, 2001.

[15] J. Sethuraman. A constructive definition of dirichlet priors. *Statist. Sinica*, 4:639–650, 1994.

[16] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their localization in images. In *ICCV*, pages 370–377, 2005.

[17] N. Ueda, R. Nakano, Z. Gharamani, and G. Hinton. Smem algorithm for mixture models, 1999.

[18] N. Vasconcelos and A. Lippman. Learning mixture hierarchies. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 606–612, Cambridge, MA, USA, 1999. MIT Press.

[19] J. J. Verbeek, J. Nunnink, and N. A. Vlassis. Accelerated em-based clustering of large data sets. *Data Min. Knowl. Discov.*, 13(3):291–307, 2006.

[20] M. Vidal-Naquet and S. Ullman. Object recognition with informative features and linear classification. In *ICCV*, pages 281–288. IEEE Computer Society, 2003.

[21] M. Weber, M. Welling, and P. Perona. Towards automatic discovery of object categories. In *CVPR*, pages 2101–, 2000.