Abstract
Hierarchical Bayesian models often capture distributions over a very large number of distinct atoms. The need for these models arises when organizing huge amount of unsupervised data, for instance, features extracted using deep convnets that can be exploited to organize abundant unlabeled images. Inference for hierarchical Bayesian models in such cases can be rather nontrivial, leading to approximate approaches. In this work, we propose Canopy, a sampler based on Cover Trees that is exact, has guaranteed runtime logarithmic in the number of atoms, and is provably polynomial in the inherent dimensionality of the underlying parameter space. In other words, the algorithm is as fast as search over a hierarchical data structure. We provide theory for Canopy and demonstrate its effectiveness on both synthetic and real datasets, consisting of over 100 million images.

1. Introduction
Fast nearest-neighbor algorithms have become a mainstay of information retrieval (Beygelzimer et al., 2006; Liu et al., 2007; Indyk & Motwani, 1998). Search engines are able to perform virtually instantaneous lookup among sets containing billions of objects. In contrast, inference procedures for latent variable models (Gibbs sampling, EM, or variational methods) are often problematic even when dealing with thousands of distinct objects. This is largely because, for any inference methods, we potentially need to evaluate all probabilities whereas search only needs the best instance.

While the above is admittedly an oversimplification of matters (after all, we can use Markov-Chain Monte Carlo methods for inference), it is nonetheless nontrivial to perform exact sampling for large state spaces. In the current work, we propose Canopy, an inference technique to address this issue by marrying a fast lookup structure with an adaptive rejection sampler. This leads to a surprisingly simple design for a plethora of sampling-based inference algorithms. Moreover, we provide runtime guarantees for Canopy that depend only on the inherent dimensionality of both parameter and data distributions. The expected depth for lookups is never worse than logarithmic in the number of atoms and the characteristic length scale at which models can be sufficiently well distinguished. Furthermore, we can parallelize Canopy for hierarchical Bayesian models using stochastic cellular automata (ESCA) (Zaheer et al., 2016), thus leading to an extremely scalable and efficient system design.

Most latent variable models, e.g., Gaussian mixture models (GMM), latent Dirichlet allocation (Blei et al., 2002), hidden Markov models, Dirichlet process clustering (Neal, 1998), or hierarchical generative models (Adams et al., 2010), have the structure of the form:

$$p(x) = \sum_z p(z)p(x|\theta_z)$$  \hspace{1cm} (1)

where $x$ denotes observed variables, $z$ latent variables, and $\theta_z$ parameters of the conditional. Often the conditional distribution $p(x|\theta_z)$ belongs to the exponential family, which we assume to be the case as well. The inference procedure on these models using either Gibbs sampling, stochastic variation methods, or ESCA would require to draw $z \sim p(z|x)$ repeatedly. Naively producing these draws would be expensive, especially when the number of latent classes is huge. We aim to bring the per-iteration cost down from $O(mn)$ to $O(m + n)$, where $m, n$ are the number of latent classes and data points, respectively. For example, on GMM, the proposed method Canopy is much faster than EM or ESCA, while achieving the same accuracy as shown in Fig. 1.
Our approach is as follows: we use cover trees (Beygelzimer et al., 2006) to design an efficient lookup structure for \( p(x|\theta_z) \) and approximate the values of \( p(z|x) \) for a large number of \( \theta_z \). In combination with an efficient node summary for \( p(z) \), this allows us to design a rejection sampler that has an increasingly low rejection rate as we descend the tree. Moreover, for large numbers of observations \( x \), we use another cover tree to aggregate points into groups of similar points, perform expensive pre-computation of assignment probabilities \( p(z|x) \) only once, and amortize them over multiple draws. In particular, the alias method (Walker, 1977) allows us to perform sampling in \( O(1) \) time once the probabilities have been computed.

In summary, Canopy has three parts: construction of cover trees for both parameters and data (Sec. 3.1, 3.2), an adaptive rejection sampler at the top-level of the cover tree until the data representation is sufficiently high to exploit it for sampling (Sec. 3.2.1), and a rejection sampler in the leaves (Sec. 3.2.2), whenever the number of clusters is large. Most importantly, the algorithm becomes more efficient as we obtain larger amounts of data since they lead to greater utilization of the alias table in (Walker, 1977) as shown by theoretical analysis in Sec. 4. This makes it particularly well-suited to big data problems as demonstrated through experiments in Sec. 5.

2. Background

We briefly discuss latent variable models, cover trees, and the alias method needed to explain this work.

2.1. Latent Variable Models

The key motivation for this work is to make inference in latent variable models more efficient. As expressed in (1), we consider latent models which have mixtures of exponential family. The reasons for limiting to exponential families are two fold. First, most of the mixture models used in practice belong to this class. Second, assumptions used in practice need to explain this work. Moreover, the degree of all internal nodes is well controlled, thus giving guarantees for retrieval (as expressed in (1)).

2.2. Alias Sampler

A key component of Canopy is the alias sampler (Walker, 1977; Vose, 1991). Given an arbitrary discrete probability distribution on \( n \) outcomes, it allows for \( O(1) \) sampling once an \( O(n) \) preprocessing step has been performed. Hence, drawing \( n \) observations from a distribution over \( n \) outcomes costs an amortized \( O(1) \) per sample. Sec. A in appendix has more details.

2.3. Cover Trees

Cover Trees (Beygelzimer et al., 2006) and their improved version (Izbicki & Shelton, 2015) are a hierarchical data structure that allow fast retrieval in logarithmic time. The key properties are: \( O(n \log n) \) construction time, \( O(\log n) \) retrieval, and polynomial dependence on the expansion constant (Karger & Ruhl, 2002) of the underlying space, which we refer to as \( c \). Moreover, the degree of all internal nodes is well controlled, thus giving guarantees for retrieval (as exploited by (Beygelzimer et al., 2006)), and for sampling (as we will be using in this paper).

Cover trees are defined as an infinite succession of levels \( S_i \) with \( i \in \mathbb{Z} \). Each level \( i \) contains a (nested subset of) the tree with the following properties:

- Nesting property: \( S_i \subseteq S_{i-1} \).
- All \( x,x' \in S_i \) satisfy \( \|x-x'\| \geq 2^i \).
- All \( x \in S_i \) have children \( x' \in S_{i-1} \), possibly with \( x=x' \), with \( \|x-x'\| \leq 2^i \).

As a consequence, the subtree for any \( x \in S_i \) has distance at most \( 2^{i+1} \) from \( x \).

Please refer to appendix Sec. C for more details.
3. Our Approach

Now we introduce notation and explain details of our approach when the number of clusters is (a) moderate (Sec. 3.1) and (b) large (Sec. 3.2). In what follows, the number of data points and clusters are denoted with $n$ and $m$ respectively. The function $c(x)$ returns children of a node $x$ of any tree.

**Data tree ($T_D$):** Cover tree built with levels $S_j$ on all available data using the sufficient statistic $\phi(x)$, constructed once for our setup. We record ancestors at level $j$ as prototypes $\bar{x}$ for each data point $x$. In fact, we only need to construct the tree up to a fixed degree of accuracy $j$ in case of moderate number of clusters. A key observation is that multiple points can have the same prototype $\bar{x}$, making it a many-to-one map. This helps us amortize costs over points by re-using proposal computed with $\bar{x}$ (Sec. 3.1).

**Cluster tree ($T_C$):** Similarly, $T_C$ is the cover tree generated with cluster parameters $\bar{\theta}_z$. For simplicity, we assume that the expansion rates of clusters and data are both $c$.

### 3.1. Canopy I: Moderate number of clusters

We introduce our sampler, Canopy I, when the number of clusters is relatively small compared to the total number of observations. This addresses many cases where we want to obtain a flat clustering on large datasets. For instance, it is conceivable that one might not want to infer more than a thousand clusters for one million observations. In a nutshell, our approach works as follows:

1. Construct $T_D$ and pick a level $j \in \mathbb{Z}$ with accuracy $2^j$ such that the average number of elements per node in $S_j$ is $O(m)$.
2. For each of the prototypes $\bar{x}$, which are members of $S_j$, compute $p(z|x)$ using the alias method to draw from $m$ components $\theta_z$. By construction, this cost amortizes $O(1)$ per observation, i.e., a total cost of $O(n)$.
3. For each observation $x$ with prototype $\bar{x}$, perform Metropolis-Hastings (MH) sampling using the draws from $p(z|\bar{x}) =: q(z)$ as proposal. Hence we accept an MH move from $z$ to $z'$ with probability

$$\pi := \min \left(1, \frac{p(z'|x)p(z|\bar{x})}{p(z|x)p(z'|\bar{x})} \right).$$

The key reason why this algorithm has a useful acceptance probability is that the normalizations for $p(z|x)$ and $p(z|\bar{x})$, and mixing proportions $p(z)$ and $p(z')$ cancel out respectively. Only terms remaining in (4) are

$$\pi = \min \left(1, \exp \left(\langle \phi(x) - \phi(\bar{x}), \theta_{z'} - \bar{\theta}_z \rangle\right) \right) \ge e^{-2^{j+2} L}$$

for $\|\bar{\theta}_z\| \le L$. This follows from the Cauchy Schwartz inequality and the covering property of cover trees, which ensures all descendants of $\bar{x}$ are no more than $2^{j+1}$ apart from $\bar{x}$, i.e., $\|\phi(x) - \phi(\bar{x})\| \le 2^{j+1}$.

### 3.2. Canopy II: Large number of clusters

The key difficulty in dealing with many clusters is that it forces us to truncate $T_D$ at a granularity $x$ that is less precise than desirable in order to benefit from the alias sampler naively. In other words, for a given sampling complexity, a larger $m$ reduces the affordable granularity in $x$. The problem arises because we are trying to distinguish clusters at a level of resolution that is too coarse. A solution is to apply cover trees not only to observations but also to the clusters themselves, i.e., use both $T_D$ and $T_C$. This allows us to decrease the minimum observation-group size at the expense of having to deal with an aggregate of possible clusters.

Our method for large number of clusters operates in two phases: (a) Descend the hierarchy in cover trees while sampling (Sec. 3.2.1) (b) Sample for a single observation $x$ from a subset of clusters arranged in $T_C$ (Sec. 3.2.2), when appropriate conditions are met in (a). We begin with initialization and then elaborate each of these phases in detail.

**Initialize 1:** Construct $T_C$ and for each node $\theta_z$, assign $\alpha(i, z) = p(z)$, where $i$ is the highest level $S_i$ such that $z \in S_i$, else 0. Then perform bottom-up aggregation via

$$\beta(i, z) = \alpha(i, z) + \sum_{z' \in c(z)} \beta(i + 1, z')$$

This creates at most $O(m)$ distinct entries $\beta(i, z)$. Notice that aggregated value $\beta(i, z)$ captures the mixing probability of the node and its children in $T_C$.

**Initialize 2:** Partition both the observations and the clusters at a resolution that allows for efficient sampling and precomputation. More specifically, we choose accuracy levels $j$ and $i$ to truncate $T_D$ and $T_C$, so that there are $n'$ and $m'$ nodes respectively after truncation. These serve as partitions for data points and clusters such that $n' \cdot m' = O(m)$ is satisfied. The aggregate approximation error

$$\delta := 2^{i+1} L + 2^{j+1} R + 2^{i+j+2}$$

due to quantizing observations and clusters is minimized over the split, searching over the levels.

#### 3.2.1. Descending $T_D$ and $T_C$

Given $T_D$ and $T_C$ with accuracy levels $j$ and $i$, we now iterate over the generated hierarchy, as shown in Fig. 2. We recursively descend simultaneously in both the trees until the number of observations for a given cluster is too small. In that case, we simply default to the sampling algorithm described in Sec. 3.2.2 for each observation in a given cluster.

The reasoning works as follows: Once we have the partitioning into levels $j, i$ for data and clusters respectively with $n' \cdot m' = O(m)$, we draw from the proposal distribution

$$q(\bar{\xi}|x) \propto \beta(i, \bar{\xi}) \exp(\langle \phi(\bar{\xi}), \hat{\theta}_z \rangle - g(\theta_z))$$
for all the observations and clusters above the partitioned levels $j$ and $i$, respectively. That is, we draw from a distribution where both observations and clusters are grouped. We draw from the proposal for each $x$ in $T_D$ truncated at level $j$. Here, $\beta(i, z)$ collects the prior cluster likelihood from $z$ and all its children. As described earlier, we can use the alias method for sampling efficiently from (7).

Within each group of observations, drawing from (7) leads to a distribution over a (possibly smaller) subset of cluster groups. Whenever the number of observations per cluster group is small, we default to the algorithm described in Sec. 3.2.2 for each observation. On the other hand, if we have a sizable number of observations for a given cluster, which should happen whenever the clusters are highly discriminative for observations (a desirable property for a good statistical model), we repeat the strategy on the subset to reduce the aggregate approximation error (6). In other words, we descend the hierarchy to yield a new pair $(i', j')$ on the subset of clusters/observations with $i' < i$ and $j' < j$ and repeat the procedure.

The process works in a depth-first fashion in order to avoid using up too much memory. The sampling probabilities according to (7) are multiplied out for the path over the various hierarchy levels and used in a MH procedure. Each level of the hierarchy can be processed in $O(1)$ operations per instance, without access to the instance itself. Moreover, we are guaranteed to descend by at least one step in the hierarchy of observations and clusters, hence the cost is at most $O(c^2 \log n, \log m)$.

To summarize, we employ a MH scheme as before, with the aim of using a highly accurate yet cheap proposal. To overcome the loss in precision of Canopy I proposal due to large of clusters, we devise a proposal wherein we look at aggregates of both data and clusters at comparable granularity using both $T_D$ and $T_C$. Note that the acceptance probabilities are always at least as high as the bounds derived in Sec. 3.1 as the errors on the paths are log-additive. Instead of MH, a rejection sampler can also be devised. Details are omitted for the sake of brevity, since they mirror the single-observation argument of the following section.

### 3.2.2. Sampling for a single observation $x$

Let $x$ be the single observation for which we want to sample from possibly subset of clusters $z$ that are arranged in $T_C$. In this case, we hierarchically descend $T_C$ using each aggregate as a proposal for the clusters below. As before, we can either use MH sampling or a rejection sampler. To illustrate the effects of the latter, we describe one below, whose theoretical analysis is provided in Sec. 3.2.2 for each observation. On the other hand, if we group is small, we default to the algorithm described in Sec. 3.1 as the errors on the paths are log-additive.

Here, $\beta(i, z)$ is multiplied out for the path over the hierarchy and sample at a finer level, until we have few number of points per cluster. We then use Sec. 3.2.1 for rejection sampler.

...
4. Theoretical Analysis

The main concern is to derive a useful bound regarding the runtime required for drawing a sample. Secondary concerns are those of generating the data structure. We address each of these components, reporting all costs per data point.

**Construction** The data structure $T_D$ costs $O(c^6 \log n)$ (per data-point) to construct and $T_C$ costs $O(c^6 \log m)$ (per data-point, as $m < n$) — all additional annotations cost negligible time and space. This includes computing $\alpha$ and $\beta$, as discussed above.

**Startup** The first step is to draw from $S_i$. This costs $O(|S_i|)$ for the first time to compute all probabilities and to construct an alias table. Subsequent samples only cost 3 CPU cycles to draw from the associated alias table. The acceptance probability at this step is $e^{-2c}$. Hence the aggregate cost for the top level is bounded by $O \left( \frac{|S_i| + e^{2c+2} \| \phi(x) \|}{c} \log m \right)$. Here the first term is due to the upper bound on the acceptance probability, a term of $c^4$ arises from the maximum number of children per node and lastly the $c^2 \log m$ term quantifies the maximum depth. It is quite clear that this term would dominate all others. We now derive a more refined (and tighter) bound.

Essentially we will exploit the fact that the deeper we descend into the tree, the less likely we will have wasted computation later in the process. We use the following relations

$$e^x - 1 \leq xe^a \text{ for } x \in [0, a] \text{ and } \sum_{i=1}^\infty 2^{-i} = 1. \quad (11)$$

In expectation, the first step of the sampler requires $e^{2c} = e^{2c+2} \| \phi(x) \|$ steps in expectation until a sample is accepted. Thus, $e^{2c} - 1$ effort is wasted. At the next level below we waste at most $e^{2c+1} \| \phi(x) \| - 1$ effort. Note that we are less likely to visit this level commensurate with the acceptance probability. These bounds are conservative since any time we terminate above the very leaf levels of the tree we are done. Moreover, not all vertices have children at all levels, and we only need to revisit them whenever they do. In summary, the wasted effort can be bounded from above by

$$c^4 \sum_{i=1}^\infty \left( e^{2c-i} \| \phi(x) \| - 1 \right) \leq c^4 e^{2c} \| \phi(x) \| \sum_{i=1}^\infty 2^{-i} = c^4 e^{2c} \| \phi(x) \|.$$  

Here $c^4$ was a consequence of the upper bound on the number of children of a vertex. Moreover, note that the exponential upper bound is rather crude, since the inequality (11) is very loose for large $a$. Nonetheless we see that the rejection sampler over the tree has computational overhead independent of the tree size! This result is less surprising than it may seem. Effectively we pay for lookup plus a modicum for the inherent top-level geometry of the set of parameters.

**Theorem 1** The cover tree sampler incurs worst-case computational complexity per sample of

$$O \left( |S_i| + e^{C + 11} \frac{|\phi(x)|}{c} \log m + c^4 e^{2c+2} \| \phi(x) \| \right) \quad (12)$$

Note that the only data-dependent terms are $c, S_i, i$ and $\| \phi(x) \|$ and that nowhere the particular structure of $p(z)$ entered the analysis. This means that our method will work equally well regardless of the type of latent variable model we apply. For example, we can even apply the model to more complicated latent variable models like latent Dirichlet allocation (LDA). The aforementioned constants are all natural quantities inherent to the problems we analyze. The constant $c$ quantifies the inherent dimensionality of the parameter space, $\| \phi(x) \|$ measures the dynamic range of the distribution, and $S_i, i$ measure the “packing number” of the parameter space at a minimum level of granularity.

5. Experiments

We now present empirical studies for our fast sampling techniques in order to establish that (i) Canopy is fast (Sec. 5.1), (ii) Canopy is accurate (Sec. 5.2), and (iii) it opens new avenues for data exploration and unsupervised learning (Sec. 5.3), previously unthinkable. To illustrate these claims, we evaluate on finite mixture models, more specifically, Gaussian Mixture models (GMM), a widely used probabilistic models. However, the proposed method can be applied effortlessly to any latent variable model like topic modeling through Gaussian latent Dirichlet allocation (Gaussian LDA) (Das et al., 2015). We pick GMMs due to their wide-spread application in various fields spanning computer vision, natural language processing, neurobiology, etc.

**Methods** For each experiment, we compare our two samplers (Canopy I, Section 3.1 and Canopy II, Section 3.2) with both the traditional Expectation Maximization (EM) (Dempster et al., 1977) and the faster Stochastic EM through ESCA (ESCA) (Zaheer et al., 2016) using execution time, cluster purity, and likelihood on a held out TEST set.

**Software & hardware** All the algorithms are implemented multithreaded in simple C++ using a distributed setup. Within a node, parallelization is implemented using the work-stealing Fork/Join framework, and the distribution across multiple nodes using the process binding to a socket over MPI. We run our experiments on a cluster of 16 Amazon EC2 c4.8xlarge nodes connected through 10Gpbs Ethernet. There are 36 virtual threads per node and 60GB of memory. For purpose of experiments, all data and calculations are carried out at double floating-point precision.
Initialization Recall that speed and quality of inference algorithms depend on initialization of the random variables and parameters. Random initializations often lead to poor results, and so many specific initialization schemes have been proposed, like KMeans++ (Arthur & Vassilvitskii, 2007), K-MC2 (Bachem et al., 2016). However, these initializations can be costly, roughly $O(mn)$.

Our approach provides a good initialization using cover trees free of cost, as the construction of cover tree is at the heart of our sampling approach. The proposed initialization scheme relies on the observation that cover trees partition the space of points while preserving important invariants based on its structure. They thus help in selecting initializations that span the entirety of space occupied by the points, which is desired to avoid local minima. The crux of the approach is to descend to a level $l$ in $\mathbb{T}_D$ such that there are no more than $m$ points at level $l$. These points from level $l$ are included in set of initial points $I$. We then randomly pick a point from $I$ such that it belongs to level $l$ and replace it with its children from level $l+1$ in $I$. This is repeated until we finally have $m$ elements in $I$. The chosen $m$ elements are mapped to parameter space through the inverse link function $g^{-1}()$ and used as initialization. All our experiments use cover tree based initializations. We also make comparisons against random and KMeans++ in Sec. 5.2.

5.1. Speed

To gauge the speed of Canopy, we begin with inference on GMMs using synthetic data. Working with synthetic data is advantageous as we can easily vary parameters like number of clusters, data points, or dimensionality to study its effect on the proposed method. Note that, from a computational perspective, data being real or synthetic does not matter as all the required computations are data independent, once the cover tree has been constructed.

Synthetic Dataset Generation Data points are assumed to be i.i.d. samples generated from $m$ Gaussian probability distributions parameterized by $(\mu_i^*, \Sigma_i^*)$ for $i = 1, 2, \ldots, m$, which mix with proportions given by $\pi_i^*$. Our experiments operate on three free parameters: $(n, m, d)$ where $n$ is the total number of points, $m$ is the number of distributions, and $d$ is the dimensionality. For a fixed $(n, m, d)$, we randomly generate a TRAIN set of $n$ points as follows: (1) Randomly pick parameters $(\mu_i^*, \Sigma_i^*)$ along with mixing proportions $\pi_i^*$, for $i = 1, 2, \ldots, m$, uniformly random at some scale. (2) To generate each point, select a distribution based on $\{\pi_i^*\}$ and sample from the corresponding $d$-dimensional Gaussian pdf. Additionally, we also generate another set of points as TEST set using the same procedure. For all the four models (Canopy I, Canopy II, EM, ESCA), parameters are learnt using TRAIN and log-likelihood on the TEST set is used as evaluation.

Observations We run all algorithms for a fixed number of iterations and vary $n, m, d$ individually to investigate the respective dependence on performance of our approach as shown in Fig. 3. We make the following observations: (1) Overall, Fig. 3 is in line with our claim that the proposed method reduced the per iteration complexity from $O(nm)$ of EM/ESCA to $O(n + m)$. (2) To illustrate this further, we consider $n = O(m)$ and vary $m$ (shown in Fig. 3c). While EM and ESCA have per-iteration time of $O(nm)$, i.e., $O(m^2)$ in this case, our Canopy I and Canopy II show $O(m + n)$, i.e., $O(m)$. (3) However, there is no free lunch. The huge speed-up comes at the cost of increased memory usage (for storing the data-structures). For example, in the case of $n = 32$ mil, $m = 4096$, and $d = 1024$ (Fig. 1), a mere $2 \times$ increase in memory gets us a speed-up of $150 \times$.

5.2. Correctness

Next, we demonstrate correctness of Canopy using medium sized real world datasets with labels, i.e., ground truth grouping of the points are known. We setup an unsupervised classification task on these datasets and perform evaluation on both cluster purity and log-likelihood.
Datasets
We use two benchmark image datasets—MNIST8m (Loosli et al., 2007) and CIFAR-100 (Krizhevsky & Hinton, 2009). The former contains 8 million annotated handwritten digits of size 28 × 28, giving us data points of dimension 784. CIFAR-100, on the other hand, contains 50k images annotated with one of 100 object categories. Each image has 3 channels (RGB) and of size 32 × 32, resulting in a vector of dimension 3072.

Unsupervised Classification.
We run unsupervised classification on the above two datasets and evaluate using cluster purity and log-likelihood. Here, cluster purity is defined as the mean of accuracy across all clusters, where each cluster is assigned the class of majority of its members. In addition to using data points as is, we also experiment with unsupervised features learnt from a denoising autoencoder (Hinton & Salakhutdinov, 2006). We extract 30 and 256 dimensional features for MNIST8m and CIFAR-100 respectively. Details of our unsupervised feature extraction are in Appendix E. Further, we evaluate in multiple scenarios that differ in (a) number of clusters: \( m = 10, 100 \) for MNIST8m and \( m = 100, 500 \) for CIFAR-100, and (b) parameter initializations (Random, Kmeans++ and CTree).

Observations
For a qualitative assessment, we randomly pick four clusters and show four images (more in Appendix F) closest to the means in Fig. 5 (each cluster in a row). We highlight two important observations: (a) Though the underlying visual feature extractor, ResNet, is trained on 1000 semantic classes, our clustering is able to discover semantic concepts that go beyond. To illustrate, images from the first row indicate a semantic class of crowd even though ResNet never received any supervision for such a concept. (b) The keywords associated with these images do not necessarily collate with the semantic concepts in the image. For example, images in first row are associated with key words ‘heave’, ‘makeshift’, ‘bloodbath’, and ‘fulfillment’, respectively. It is not too surprising as the relatedness of retrieved images for a query key word generally decreases for lower ranked images. This suggests that pre-processing images to obtain more meaningful semantic classes could potentially improve the quality of labels used to learn models. Such a cleanup would definitely prove beneficial in learning deep image classification models from weakly supervised data.

5.3. Scalability - A New Hope
Finally, we demonstrate the scalability of our algorithm by clustering a crawled dataset having more than 100 million images that belong to more than 80,000 classes. We query Flickr\(^1\) with the key words from WordNet (Fellbaum, 1998) and downloaded the returned images for each key word, those images roughly belong to the same category. We extracted the image features of dimension 2048 with ResNet (He et al., 2015; 2016) – the state-of-the-art convolutional neural network (CNN) on ImageNet 1000 classes data set—using publicly available pre-trained model of 200 layers\(^2\). It takes 5 days with 20 GPUs to extract these features for all the images. We then use Canopy II to cluster these images with \( m = 64000 \), taking around 27 hours.

Observations
For a qualitative assessment, we randomly pick four clusters and show four images (more in Appendix F) closest to the means in Fig. 5 (each cluster in a row). We highlight two important observations: (a) Though the underlying visual feature extractor, ResNet, is trained on 1000 semantic classes, our clustering is able to discover semantic concepts that go beyond. To illustrate, images from the first row indicate a semantic class of crowd even though ResNet never received any supervision for such a concept. (b) The keywords associated with these images do not necessarily collate with the semantic concepts in the image. For example, images in first row are associated with key words ‘heave’, ‘makeshift’, ‘bloodbath’, and ‘fulfillment’, respectively. It is not too surprising as the relatedness of retrieved images for a query key word generally decreases for lower ranked images. This suggests that pre-processing images to obtain more meaningful semantic classes could potentially improve the quality of labels used to learn models. Such a cleanup would definitely prove beneficial in learning deep image classification models from weakly supervised data.

---

\(^1\)http://www.flickr.com/  
\(^2\)github.com/facebook/fb.resnet.torch
6. Discussion

We present an efficient sampler, Canopy, for mixture models over exponential families using cover trees that brings the per-iteration cost down from $O(mn)$ to $\tilde{O}(m + n)$. The use of cover trees over both data and clusters combined with alias sampling can significantly improve sampling time with no effect on the quality of the final clustering. We demonstrate speed, correctness, and scalability of Canopy on both synthetic and large real world datasets. To the best of our knowledge, our clustering experiment on a hundred million images is the largest to be reported. We conclude with some related works and future extensions.

Related works There has been work using nearest-neighbor search for guiding graphical model inference like kd-trees (Moore, 1999; Gray & Moore, 2000). But use of kd-trees is not scalable with respect to dimensionality of the data points. Moreover, kd-trees could be deeper (especially for small $c$) and do not have special properties like covering, which can be exploited for speeding up sampling. We observe this empirically when training kd-tree based methods using publicly available code\(^3\). The models fail to train for dimensions greater than 8, or number of points greater than few thousands. In contrast, our method handles millions of points with thousands of dimensions.

Further approximations From our experiments, we observe that using a simplified single observation sampling in Canopy II works well in practice. Instead of descending on the hierarchy of clusters, we perform exact proposal computation for $k$ closest clusters obtained through fast lookup from $T_C$. All other clusters are equally assigned the least out of these $k$ exact posteriors.

In the future, we plan to integrate Canopy with:

**Coresets** Another line of work to speed up mixture models and clustering involves finding a weighted subset of the data, called coreset (Lucic et al., 2016; Feldman et al., 2013). Models trained on the coreset are provably competitive with those trained on the original data set. Such approaches reduce the number of samples $n$, but perform traditional inference on the coreset. Thus, our approach can be combined with coreset for additional speedup.

**Inner product acceleration** In an orthogonal direction to Canopy, several works (Ahmed et al., 2012; Mussmann & Ermon, 2016) have used maximum inner product search to speed up inference and vice versa (Auvolat et al., 2015). We want to incorporate these ideas into Canopy as well, since the inner product is evaluated $m$ times each iteration, it becomes the bottleneck for large $m$ and $d$. A solution to overcome this problem would be to use binary hashing (Ahmed et al., 2012) as a good approximation and therefore a proposal distribution with high acceptance rate.

Combining these ideas, one could build an extremely scalable and efficient system, which potentially could bring down the per-iteration sampling cost from $O(mnd)$ to $\tilde{O}(m + n + d)$ or less!

---

\(^3\)http://www.cs.cmu.edu/~psand/
References


