Canopy — Fast Sampling with Cover Trees

Manzil Zaheer [1 2] **Satwik Kottur** [1] **Amr Ahmed** [3] **Alex Smola** [1 2]

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 clustering (Gibbs sampling, stochastic 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 'clusters' 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.

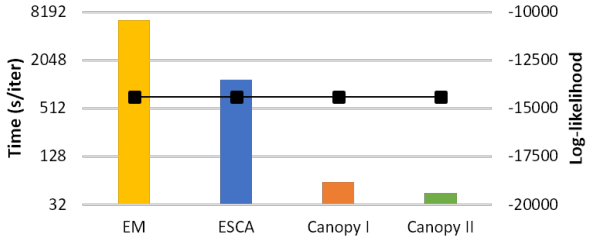


Figure 1. Canopy is much faster yet as accurate as other methods like EM or ESCA (Zaheer et al., 2016). The bar graph shows time per iteration while line plots the likelihood on held-out test set. Results shown are for inference of a Gaussian mixture model with 32 million points having 4096 clusters at 1024 dimensions.

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:

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

where x denotes observed variables, z latent variables, and θ_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. Naïvely 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 $\tilde{O}(m+n)$, 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.

[1]Carnegie Mellon University, Pittsburgh PA [2]Amazon Web Services [3]Google Inc, Mountain View CA. Correspondence to: Manzil Zaheer <manzil@cmu.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, 2017. JMLR: W&CP.

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 $p(x|\theta_z)$ for a large number of θ_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 on model structure, for instance exponential family, allows for efficient design of fast inference. In particular, we first assume that updates to $p(z)$ can be carried out by modifying $O(1)$ values at any given time. For instance, for Dirichlet process mixtures, the collapsed sampler uses $p(z_i = j|Z\setminus\{z_i\}) = n_j^{-i}/(n+\alpha-1)$. Here, n is the total number of observations, n_j^{-i} denotes the number of occurrences of $z_l = j$ when ignoring z_i, and α is the concentration parameter. Second, the conditional $p(x|\theta)$ in (1) is assumed to be a member of the exponential family, *i.e.*,

$$p(x|\theta) = \exp(\langle\phi(x), \theta\rangle - g(\theta)). \tag{2}$$

Here $\phi(x)$ represents the sufficient statistics and $g(\theta_z)$ is the (normalizing) log-partition function.

Trying to find a metric data structure for fast retrieval is not necessarily trivial for the exponential family. Jiang et al. (2012) and Cayton (2008) design Bregman divergence based methods for this problem. Unfortunately, such methods are costlier to maintain and have less efficient lookup properties than those using Euclidean distance, as computing and optimizing over Bregman divergences is less straightforward. For example, whenever we end up on the boundary of the marginal polytope, as is common with natural parameters associated with single observations, optimization becomes intractable. Fortunately, this problem can be avoided entirely by rewriting the exponential family model as

$$p(x|\theta) = e^{\langle(\phi(x),-1),\theta,g(\theta)\rangle} = e^{\left\langle\tilde{\phi}(x),\tilde{\theta}\right\rangle} \tag{3}$$
$$\text{where } \tilde{\phi}(x) := (\phi(x), -1) \text{ and } \tilde{\theta} := (\theta, g(\theta)).$$

In this case, being able to group similar $\tilde{\theta}$ together allows us to assess their contributions efficiently without having to inspect individual terms. Finally, we assume that $\left\|\tilde{\phi}(x_i)\right\| \leq R$ and $\left\|\tilde{\theta}_z\right\| \leq T$ for all i and for all $z \in \mathcal{Z}$ respectively.

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 data with the following properties:

- Nesting property: $S_{i-1} \subseteq S_i$.
- All $x, x' \in S_i$ satisfy $\|x - x'\| \geq 2^i$.
- All $x \in S_{i+1}$ have a parent in $x' \in S_i$, 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 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 $\mathsf{ch}(x)$ returns children of a node x of any tree.

Data tree ($\mathbb{T}_D$): 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 $\bar{j}$ in case of moderate number of clusters. A key observation is that multiple points can have a 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 ($\mathbb{T}_C$): Similarly, $\mathbb{T}_C$ is the cover tree generated with cluster parameters $\tilde{\theta}_z$. 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 $\mathbb{T}_D$ and pick a level $\bar{j} \in \mathbb{Z}$ with accuracy $2^{\bar{j}}$ such that the average number of elements per node in $S_{\bar{j}}$ is $O(m)$.
2. For each of the prototypes $\bar{x}$, which are members of $S_{\bar{j}}$, compute $p(z|\bar{x})$ using the alias method to draw from m components θ_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 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

$$\pi := \min\left(1, \frac{p(z'|x)p(z|\bar{x})}{p(z|x)p(z'|\bar{x})}\right). \tag{4}$$

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 likelihoods $p(z)$ and $p(z')$ cancel out respectively. Only terms remaining in (4) are

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

for $\|\theta_z\| \leq L$. This follows from the Cauchy Schwartz inequality and the nesting property of cover trees, that all children of $\bar{x}$ are no more than $2^{\bar{j}}$ apart from each other, *i.e.*, $\|\phi(x) - \phi(\bar{x})\| \leq 2^{\bar{j}}$.

3.2. Canopy II: Large number of clusters

The key difficulty in dealing with many clusters is that it forces us to truncate $\mathbb{T}_D$ at a granularity in 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 $\mathbb{T}_D$ and $\mathbb{T}_C$. 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 $\mathbb{T}_C$ (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 $\mathbb{T}_C$ and for each node θ_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

$$\beta(i, z) = \alpha(i, z) + \sum_{z' \in \mathsf{ch}(z)} \beta(i+1, z') \tag{5}$$

This creates m entries $\beta(z)$ as $\mathbb{T}_C$ has exactly m nodes. Notice that aggregated value $\beta(z)$ captures the probability of θ_z and its children in $\mathbb{T}_C$.

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 $\hat{\imath}$ and $\hat{\jmath}$ to truncate $\mathbb{T}_D$ and $\mathbb{T}_C$, 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

$$\delta := 2^{\hat{\imath}} L + 2^{\hat{\jmath}} R + 2^{\hat{\imath}+\hat{\jmath}+1} \tag{6}$$

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

3.2.1. Descending $\mathbb{T}_D$ and $\mathbb{T}_C$

Given $\mathbb{T}_D$ and $\mathbb{T}_C$ with accuracy levels $\hat{\imath}$ and $\hat{\jmath}$, 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 $\hat{\imath}, \hat{\jmath}$ for data and clusters respectively with $n' \cdot m' = O(m)$, we draw from the proposal distribution

$$q(\bar{z}|x) \propto \beta(\theta_{\bar{z}}) \exp\left(\langle \phi(\bar{x}), \theta_{\bar{z}} \rangle - g(\theta_{\bar{z}})\right) \tag{7}$$

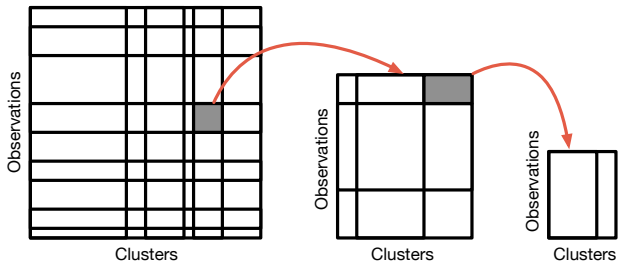


Figure 2. Hierarchical partitioning over both data observations and clusters. The accuracy at a particular level is indicated by rows for observations and columns for clusters. Once we sample clusters at a coarser level, we descend the hierarchy and sample at a finer level, until we have few number of points per cluster. We then default to Sec. 3.2.1 for rejection sampler.

for all the observations and clusters above the partitioned levels $\hat{\imath}$ and $\hat{\jmath}$, respectively. That is, we draw from a distribution where both observations and clusters are grouped. We draw from the proposal for each x in $\mathbb{T}_D$ truncated at level $\hat{\imath}$. Here, $\beta(\theta_{\bar{z}})$ collects the prior cluster likelihood from $\bar{z}$ 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 $i' < \hat{\imath}$ and $j' < \hat{\jmath}$ 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 Metropolis-Hastings 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 $O(c^2 \min(\log n, \log m))$.

Note that the acceptance probabilities are always at least as high as the bounds derived in Sec. 3.1 since the errors on the paths are log-additive. An alternative would be to use a rejection sampler. Details are omitted for the sake of brevity and 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 $\mathbb{T}_C$. In this case, we hierarchically descend $\mathbb{T}_C$ 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. 4. If we are able to approximate $p(x|\theta_z)$ by some q_z such that

$$e^{-\epsilon} p(x|\theta_z) \leq q_z \leq e^{\epsilon} p(x|\theta_z) \tag{8}$$

then it follows that a sampler drawing z from

$$z \sim \frac{q_z p(z)}{\sum_{z'} q_{z'} p(z')} \tag{9}$$

and then accepting with probability $e^{-\epsilon} q_z^{-1} p(x|\theta_z)$ will draw from $p(z|x)$ (see Appendix Sec. B for details). Moreover, the acceptance probability is at least $e^{-2\epsilon}$. We will obtain such a bound by successively approximating the set of θ_z via cover tree $\mathbb{T}_C$ using the sampler described below:

1. Choose approximation level $\hat{\imath}$ and set $e^{-\epsilon} = e^{-2^{\hat{\imath}} \|\tilde{\phi}(x)\|}$ as multiplier for the acceptance threshold of the sampler.
2. Compute normalization at accuracy level $\hat{\imath}$

$$\gamma := e^{\epsilon} \sum_{z \in S_{\hat{\imath}}} \beta(\hat{\imath}, z) \exp \left\langle \tilde{\theta}_z, \tilde{\phi}(x) \right\rangle \tag{10}$$

3. Draw $\xi \sim U[0, 1]$.
4. Draw a child $z \in S_{\hat{\imath}}$ with probability $\delta_z := e^{-\epsilon} \gamma^{-1} \beta(\hat{\imath}, z) \exp \left\langle \tilde{\theta}_z, \tilde{\phi}(x) \right\rangle$ and restrict ξ to fall into the interval $[0, \delta_z]$. Denote this child by $z_{\hat{\imath}}$.
5. Accept θ_z (we bail out at the current level $\hat{\imath}$) with probability $\gamma^{-1} p(z) \exp \left\langle \tilde{\theta}_z, \tilde{\phi}(x) \right\rangle$ and reduce ξ by this amount if we do not accept, for recycling ξ again.
6. For $i := \hat{\imath} - 1$ to $-\infty$ do

i. Set $e^{-\epsilon} = e^{-2^i \|\tilde{\phi}(x)\|}$ as the new accuracy level.

ii. Draw one of the children z of z_{i+1} with probability $\delta_z := \epsilon \gamma^{-1} \beta(i, z) \exp \left\langle \tilde{\theta}_z, \tilde{\phi}(x) \right\rangle$ and restrict ξ to fall into the interval $[0, \delta_z]$, i.e. we recycle the random variable ξ. Exit if we do not draw any of them (since $\sum_z \delta_z \leq 1$) and restart from step 3.

iii. Accept θ_z at the current level with $\gamma^{-1} p(z) \exp \left\langle \tilde{\theta}_z, \tilde{\phi}(x) \right\rangle$ and reduce ξ by this amount if we do not accept, for recycling ξ again. Do *not include* z_{i+1} in this setting since we may only accept θ_z the first time we encounter it.

The above describes a rejection sampler that keeps on upper-bounding the probability of accepting a particular cluster or any of its children. It is as aggressive as possible at retaining tight lower bounds on the *acceptance* probability such that not too much effort is wasted in traversing the cover tree to the bottom, *i.e.*, we attempt to reject as quickly as possible.

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 $\mathbb{T}_D$ costs $O(c^6 \log n)$ (per data-point) to construct and $\mathbb{T}_C$ costs $O(c^6 \log m)$ (per data-point, as $m < n$) — all additional annotations cost negligible time and space. This includes computing α and β, as discussed above.

Startup The first step is to draw from $S_{\hat{\imath}}$. This costs $O(|S_{\hat{\imath}}|)$ 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 ϵ. Hence the aggregate cost for the top level is bounded by $O\left(|S_{\hat{\imath}}| + e^{2^{\hat{\imath}}\left\|\tilde{\phi}(x)\right\|}\right)$.

Termination To terminate the sampler successfully, we need to traverse $\mathbb{T}_C$ at least once to its leaf in the worst case. This costs $O(c^6 \log m)$ if the leaf is at maximum depth.

Rejections The main effort of the analysis is to obtain useful guarantees for the amount of effort wasted in drawing from the cover tree. A brute-force bound immediately would yield $O\left(e^{2^{\hat{\imath}}\left\|\tilde{\phi}(x)\right\|} c^6 \log m\right)$. 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

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

In expectation, the first step of the sampler requires $\epsilon^{-1} = e^{2^{\hat{\imath}}\left\|\tilde{\phi}(x)\right\|}$ steps until a sample is accepted. Thus, $\epsilon^{-1} - 1$ effort is wasted. At the next level below we waste at most $e^{2^{\hat{\imath}-1}\left\|\tilde{\phi}(x)\right\|}$ 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

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

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*

$$O\left(|S_{\hat{\imath}}| + c^6 \log n + c^6 \log m + c^4 e^{2^{\hat{\imath}}\left\|\tilde{\phi}(x)\right\|}\right) \qquad (12)$$

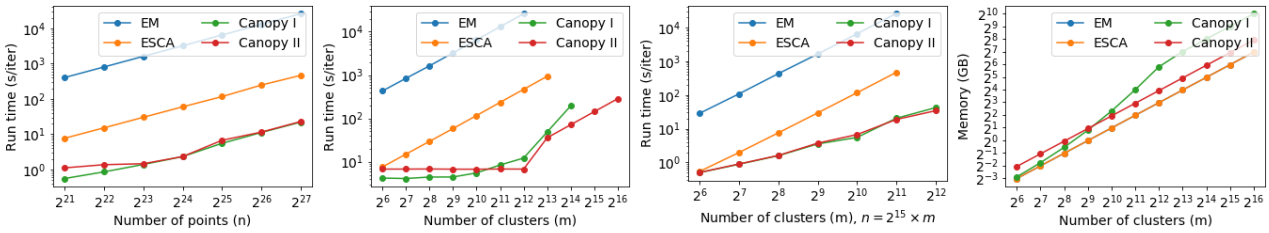
Note that the only data-dependent terms are $c, S_{\hat{\imath}}, \hat{\imath}$ and $\left\|\tilde{\phi}(x)\right\|$ and that nowhere the particular structure of $p(z)$ entered the analysis. This means that our method will work equally well regardless of the type of latent variable model we apply. For 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, $\left\|\tilde{\phi}(x)\right\|$ measures the dynamic range of the distribution, and $S_{\hat{\imath}}, \hat{\imath}$ 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 $C++11$ 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 10Gb/s 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.



(a) Varying n with fixed $(m, d) = (2^{10}, 2^{10})$

(b) Varying m with fixed $(n, d) = (2^{25}, 2^{10})$

(c) Varying m with $n = 2^{15}m$ and $d = 2^{10}$

(d) Varying m with fixed $(n, d) = (2^{25}, 2^{10})$

Figure 3. Showing scalability of per-iteration runtime of different algorithms with increasing dataset size. From Fig. 3a, 3b, and 3c we see that our approaches take orders of magnitude less time compared to the traditional EM and ESCA methods, while varying the number of points and clusters respectively. Note that we trade off memory for speed as seen from Fig. 3d. For instance, with $(n, m, d) = (32\text{mil}, 4096, 1024)$, we see that there is a speed-up of *150*$\times$ for a mere $2\times$ memory overhead.

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 $g^{-1}(\cdot)$ 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 (μ_i^*, Σ_i^*) for $i = 1, 2, \cdots, m$, which mix with proportions given by π_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 (μ_i^*, Σ_i^*) along with mixing proportions π_i^*, for $i = 1, 2, \cdots, m$, 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 Likelihood of 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 $\tilde{O}(n + m)$. (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 $O(mn)$, *i.e.*, $O(m^2)$ in this case, our Canopy I and Canopy II show $\tilde{O}(m + n)$, *i.e.*, $\tilde{O}(m)$. (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 loglikelihood.

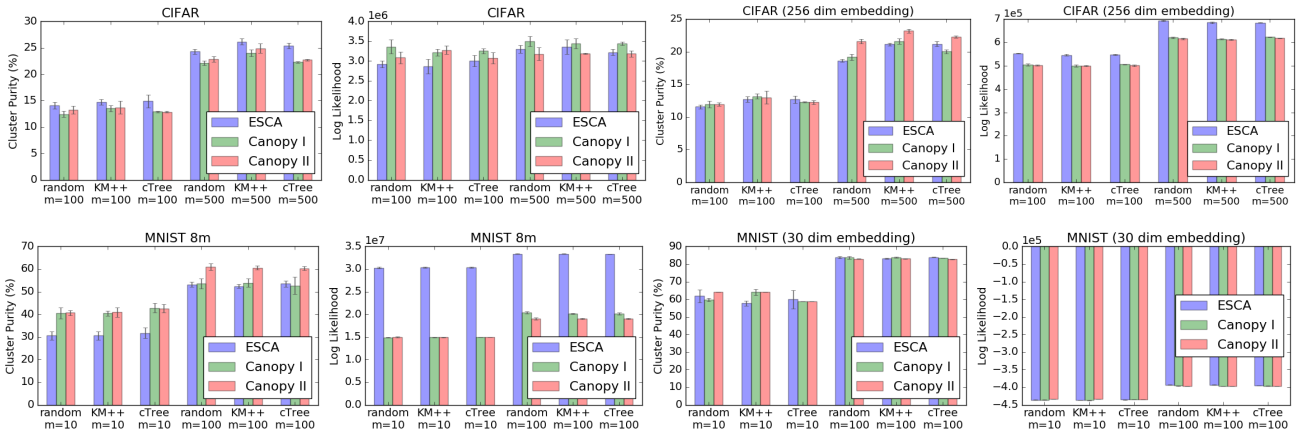


Figure 4. Plots of cluster purity and loglikehood of ESCA, Canopy I, and Canopy II on benchmark real datasets –MNIST8m and CIFAR-100. All three methods have roughly same performance on cluster purity. See Sec. 5.2 for more details.

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 loglikelihood. 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: Fig. 4 shows our results on MNIST8m ($m = 10, 100$) and CIFAR-100 ($m = 100, 500$), with error bars computed over 5 runs. Here are the salient observations: (1) All the methods (SEM, Canopy I, Canopy II) have roughly the same cluster purity with Canopy II outperforming in CIFAR-100 (256 dim) and MNIST8m by around 10% and 3% respectively. In CIFAR-100, SEM does slightly better than other methods by 2-3%. (2) Similar results are obtained for loglikelihood except for MNIST8m, where SEM heavily outperforms Canopy. However, note that loglikelihood results in an unsupervised task can be misleading (Chang et al., 2009), as evidenced here by superior performance of Canopy in terms of cluster purity.

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 'fullfillment', 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`

Figure 5. Illustration of concepts captured by clustering images in the ResNet (He et al., 2015; 2016) feature space. We randomly pick three clusters and show four closest images (one in each row), possibly denoting the semantic concepts of 'crowd', 'ocean rock scenery' and 'horse mounted police'. Our clustering discovers new concepts beyond the Resnet supervised categories (does not include 'crowd').

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 $\mathbb{T}_C$. 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

Adams, R., Ghahramani, Z., and Jordan, M. Tree-structured stick breaking for hierarchical data. In *Neural Information Processing Systems*, pp. 19–27, 2010.

Ahmed, A., Ravi, S., Narayanamurthy, S., and Smola, A.J. Fastex: Hash clustering with exponential families. In *Neural Information Processing Systems 25*, pp. 2807–2815, 2012.

Arthur, David and Vassilvitskii, Sergei. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pp. 1027–1035, 2007. URL http://dl.acm.org/citation.cfm?id=1283383.1283494.

Auvolat, Alex, Chandar, Sarath, Vincent, Pascal, Larochelle, Hugo, and Bengio, Yoshua. Clustering is efficient for approximate maximum inner product search. *arXiv preprint arXiv:1507.05910*, 2015.

Bachem, Olivier, Lucic, Mario, Hassani, S. Hamed, and Krause, Andreas. Approximate k-means++ in sublinear time. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pp. 1459–1467, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12147.

Beygelzimer, A., Kakade, S., and Langford, J. Cover trees for nearest neighbor. In *International Conference on Machine Learning*, 2006.

Blei, D., Ng, A., and Jordan, M. Latent dirichlet allocation. In Dietterich, T. G., Becker, S., and Ghahramani, Z. (eds.), *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

Cayton, L. Fast nearest neighbor retrieval for bregman divergences. In *International Conference on Machine Learning ICML*, pp. 112–119. ACM, 2008.

Chang, Jonathan, Gerrish, Sean, Wang, Chong, Boyd-graber, Jordan L., and Blei, David M. Reading tea leaves: How humans interpret topic models. In Bengio, Y., Schuurmans, D., Lafferty, J. D., Williams, C. K. I., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 22*, pp. 288–296. Curran Associates, Inc., 2009.

Das, Rajarshi, Zaheer, Manzil, and Dyer, Chris. Gaussian lda for topic models with word embeddings. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics*, 2015.

Dempster, A. P., Laird, N. M., and Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39(1):1–22, 1977.

Feldman, Dan, Schmidt, Melanie, and Sohler, Christian. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1434–1453. Society for Industrial and Applied Mathematics, 2013.

Fellbaum, C. *WordNet: An electronic lexical database*. The MIT press, 1998.

Gray, Alexander G and Moore, Andrew W. N-body'problems in statistical learning. In *NIPS*, volume 4, pp. 521–527. Citeseer, 2000.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

Hinton, Geoffrey and Salakhutdinov, Ruslan. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

Indyk, Piotr and Motwani, Rajeev. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613. ACM, 1998.

Izbicki, Mike and Shelton, Christian R. Faster cover trees. In *Proceedings of the Thirty-Second International Conference on Machine Learning*, 2015.

Jiang, K., Kulis, B., and Jordan, M. Small-variance asymptotics for exponential family dirichlet process mixture models. In *Neural Information Processing Systems NIPS*, pp. 3167–3175, 2012.

Karger, D. R. and Ruhl, M. Finding nearest neighbors in growth-restricted metrics. In *Symposium on Theory of Computing STOC*, pp. 741–750. ACM, 2002.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Liu, Ting, Rosenberg, Charles, and Rowley, Henry A. Clustering billions of images with large scale nearest neighbor search. In *Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on*, pp. 28–28. IEEE, 2007.

Loosli, Gaëlle, Canu, Stéphane, and Bottou, Léon. Training invariant support vector machines using selective sampling. In Bottou, Léon, Chapelle, Olivier, DeCoste, Dennis, and Weston, Jason (eds.), *Large Scale Kernel Machines*, pp. 301–320. MIT Press, Cambridge, MA., 2007.

Lucic, Mario, Bachem, Olivier, and Krause, Andreas. Strong coresets for hard and soft bregman clustering with applications to exponential family mixtures. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 1–9, 2016.

Moore, Andrew W. Very fast em-based mixture model clustering using multiresolution kd-trees. *Advances in Neural information processing systems*, pp. 543–549, 1999.

Mussmann, Stephen and Ermon, Stefano. Learning and inference via maximum inner product search. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2587–2596, 2016.

Neal, R. Markov chain sampling methods for dirichlet process mixture models. Technical Report 9815, University of Toronto, 1998.

Vose, Michael D. A linear algorithm for generating random numbers with a given distribution. *Software Engineering, IEEE Transactions on*, 17(9):972–975, 1991.

Walker, Alastair J. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3): 253–256, 1977.

Zaheer, M., Wick, M., Tristan, J.B., Smola, A. J., and Steele, G. L. Exponential stochastic cellular automata for massively parallel inference. In *Artificial Intelligence and Statistics*, 2016.

A. Alias Sampler

A key component is the alias sampler of (Walker, 1977). 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 a distribution over n outcomes costs an amortized $O(1)$ per sample. Given probabilities π_i with $\pi \in \mathcal{P}_n$ the algorithm proceeds as follows:

- Decompose $\{1, \dots n\}$ into sets L, H with $i \in L$ if $\pi_i < n^{-1}$ and $i \in H$ otherwise.
- For each $i \in L$ pick some $j \in H$.
  - Append the triple (i, j, π_i) to an array A
  - Set residual $\pi'_j := \pi_j + \pi_i - n^{-1}$
  - If $\pi'_j > n^{-1}$ return π'_j to H, otherwise to L.

Preprocessing takes $O(n)$ computation and memory since we remove one element at a time from L.

- To sample from the array pick $u \sim U(0, 1)$ uniformly at random.
- Choose the tuple (i, j, π_i) at position $\lfloor un \rfloor$.
- If $u - n^{-1}\lfloor un \rfloor < \pi_i$ return i, else return j.

This step costs $O(1)$ operations and it follows by construction that i is returned with probability π_i. Now we need a data structure that will allow us to sample many objects *in bulk* without the need to inspect each item individually. Cover trees satisfy this requirement.

B. Rejection Sampling

The proof for the proposed rejection sampler in case of sampling a cluster for a single observation x is as follows. If we approximate $p(x|\theta_z)$ by some q_z such that

$$e^{-\epsilon} p(x|\theta_z) \leq q_z \leq e^{\epsilon} p(x|\theta_z) \tag{13}$$

then it follows that a sampler drawing z from

$$z \sim \frac{q_z p(z)}{\sum_{z'} q_{z'} p(z')} \tag{14}$$

and then accepting with probability $e^{-\epsilon} q_z^{-1} p(x|\theta_z)$ will draw from $p(z|x)$. To prove this, we simply compute the probability of this sampler $r(z)$ to return a particular value z. The sample returns z when it (a) samples and accepts z, or (b) samples any value, rejects it to proceed to next iteration of sampling. Using $\gamma = \sum_{z'} q_{z'} p(z')$ and $\gamma_T = \sum_{z'} p(x|\theta_{z'}) p(z')$ to denote normalization for proposal and true posterior respectively, we have:

$$r(z) = \frac{q_z p(z)}{\gamma} e^{-\epsilon} q_z^{-1} p(x|\theta_z) + \sum_{z'} (1 - e^{-\epsilon} q_{z'}^{-1} p(x|\theta_{z'})) \frac{q_{z'} p(z')}{\gamma} r(z) \tag{15}$$

$$= \frac{e^{-\epsilon}}{\gamma} p(z) p(x|\theta_z) + \frac{r(z)}{\gamma} \sum_{z'} q_{z'} p(z') - r(z) \frac{e^{-\epsilon}}{\gamma} \sum_{z'} p(x|\theta_{z'}) p(z') \tag{16}$$

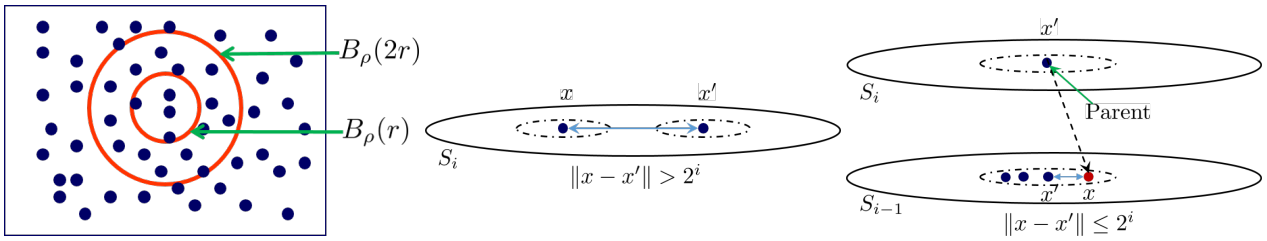
$$= \frac{e^{-\epsilon}}{\gamma} p(z) p(x|\theta_z) + r(z) - r(z) \frac{e^{-\epsilon}}{\gamma} \gamma_T \tag{17}$$

$$r(z) = \frac{p(z) p(x|\theta_z)}{\gamma_T} \tag{18}$$

Hence the procedure will draw from the true posterior $p(z|x)$.

C. Cover Trees

Cover Trees (Beygelzimer et al., 2006) and their improved version (Izbicki & Shelton, 2015) form a hierarchical data structure that allows fast retrieval in logarithmic time. The key properties for the purpose of this paper are that it allows for $O(n \log n)$ construction time, $O(\log n)$ retrieval, and that it only depends polynomially on the expansion rate (Karger &



(a) Expansion rate (b) Separation property (c) Covering property

Figure 6. Illustration of various properties of covering tree.

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 in (Beygelzimer et al., 2006)), and for sampling (as we will be using in this paper).

The expansion rate of a set, due to (Karger & Ruhl, 2002) captures several key properties.

Definition 2 (Expansion Rate) *Denote by $B_\rho(r)$ a ball of radius of r centered at ρ. Then a set S has a (l, c) expansion rate iff all $r > 0$ and $\rho \in S$ satisfy*

$$|B_\rho(r) \cap S| \geq l \Longrightarrow |B_\rho(2r) \cap S| \leq c\,|B_\rho(r) \cap S|\,. \tag{19}$$

In the following we set $l = O(\log |S|)$, thus referring to c simply as the expansion rate of S.

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 data with the following properties:

- Nesting property: $S_{i-1} \subseteq S_i$.
- Separation property: All $x, x' \in S_i$ satisfy $\|x - x'\| \geq 2^i$.
- All $x \in S_{i-1}$ have a parent in $x' \in S_i$, 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 from x.

Clearly we need to reperesent each x only once, namely in terms of S_i with the largest i for which $x \in S_i$ holds. This data structure has a number of highly desirable properties, as proved in (Beygelzimer et al., 2006). We list the most relevant ones below:

- The depth of the tree in terms of its explicit representation is at most $O(c^2 \log n)$.
- The maximum degree of any node is $O(c^4)$.
- Insertion & removal take at most $O(c^6 \log n)$ time.
- Retrieval of the nearest neighbor takes at most $O(c^{12} \log n)$ time.
- The time to construct the tree is $O(c^6 n \log n)$.

The fast lookup of cover tree is built upon the implicit assumption in terms of the distinguishability of parameters θ_z, which we also borrow in Canopy. This is related to the issue that if we had many choices of θ_z that, a-priori, all looked quite relevant yet distinct, we would have no efficient means of evaluating them short of testing all by brute force. Note that this could be achieved, e.g. by using the fast hash approximation of a sampler in (Ahmed et al., 2012). This is complementary to the present paper.

D. Theoretical Analysis

Some more conclusions we can make about the algorithm:

Remark 3 (Rejection Sampler) *The same reasoning yields a rejection sampler since*

$$\frac{p(z|\bar{x})}{p(z|x)} \geq e^{-\|\phi(x)-\phi(\bar{x})\|\|\theta_z\|} \geq e^{-2^{\bar{\jmath}+1}L}. \tag{20}$$

Here we may bound each term (and the normalization) in computing $p(z|x)$ appropriately.

Remark 4 *The efficiency of the sampler increases as the sample size m increases. In particular, an increase of m by $O(c^4)$ is guaranteed to decrease $\bar{\jmath}$ by 1, thus increasing the acceptance probability π from π to $\sqrt{\pi}$. This follows from the fact that each node in the cover tree has at most $O(c^4)$ children.*

Remark 5 *There is no need to build a cover tree to a level beyond $\bar{\jmath}$ since we do not exploit the improvement. This could be used to remove the logarithmic dependence $O(n \log n)$ in constructing the cover tree and reduce it to $O(n\bar{\jmath})$.*

E. Feature Extraction

E.1. Denoising Autoencoder for MNIST

The autoencoder consists of an encoder with fully connected layers of size (28x28)-1000-500-250-30 and a symmetric decoder. The thirty units in the code layer were linear and all the other units were logistic. The network was trained on the 8 million images using mean square error loss.

E.2. Denoising Autoencoder for CIFAR100

The autoencoder consists of an encoder with convolutional layers of size (3x32x32)-(64, 5, 5)-(32, 5, 5)-(16, 4, 4) and having a 2x2 max pooling after each convolutional layer. The decoder is symmetric with max pooling replaced by upsampling. The 256 units in the code layer were linear and all the other internal units were RelU while the final layer was sigmoid. The network was trained on the 50 thousand images using mean square error loss.

E.3. ResNet for ImageNet

We use the state of the art deep convolutional neural network (DCNN), based on the ResNet ("Residual Network") architecture (He et al., 2015; 2016). ResNet consists of small building blocks of layers which learn the residual functions with reference to the input. It is demonstrated that ResNet is able to train networks that are substantially deeper without the problem of noisy backpropagation gradient. For feature extraction We use a 200 layer ResNet that is trained on a task of classification on ImageNet. In the process, the network learned which high-level visual features (and combinations of those features) are important. After training the model, we remove the final classification layer of the network and extract from the next-to-last layer of the DCNN, as the representation of the input image which is of dimension 2048.

F. Further Experimental Results

MNIST8m - Direct										
			Random I		Random II		KMeans++		CoverTree	
Clusters	**Method**	**s/iter**	**LLH**	**Purity**	**LLH**	**Purity**	**LLH**	**Purity**	**LLH**	**Purity**
10	EM	39.588 ± 1.801	3.04×10^7	32.39%	3.05×10^7	30.76%	3.04×10^7	30.81%	3.05×10^7	30.50%
	SEM	7.124 ± 0.241	3.04×10^7	32.33%	3.03×10^7	30.65%	3.04×10^7	30.61%	3.04×10^7	31.69%
	Canopy I	7.453 ± 0.255	1.49×10^7	42.12%	1.49×10^7	40.51%	1.49×10^7	40.41%	1.50×10^7	42.84%
	Canopy II	7.534 ± 0.320	1.49×10^7	42.85%	1.49×10^7	40.69%	1.49×10^7	40.95%	1.50×10^7	42.59%
100	EM	512.185 ± 13.295	3.27×10^7	53.20%	3.26×10^7	53.24%	3.28×10^7	52.45%	3.32×10^7	53.10%
	SEM	10.085 ± 0.162	3.34×10^7	53.19%	3.34×10^7	53.21%	3.34×10^7	52.42%	3.33×10^7	53.52%
	Canopy I	6.882 ± 0.174	2.02×10^7	53.39%	2.04×10^7	53.53%	2.01×10^7	53.88%	2.02×10^7	52.69%
	Canopy II	6.483 ± 0.298	1.91×10^7	60.19%	1.90×10^7	61.09%	1.90×10^7	60.61%	1.90×10^7	60.29%
MNIST8m - Embedding										
			Random I		Random II		KMeans++		CoverTree	
Clusters	**Method**	**s/iter**	**LLH** ($\times 10^7$)	**Purity**	**LLH** ($\times 10^7$)	**Purity**	**LLH** ($\times 10^7$)	**Purity**	**LLH** ($\times 10^7$)	**Purity**
10	EM	6.595 ± 0.230	-4.35×10^5	58.43%	-4.36×10^5	63.14%	-4.35×10^5	63.19%	-4.34×10^5	63.22%
	SEM	0.943 ± 0.037	-4.35×10^5	58.43%	-4.35×10^5	62.05%	-4.36×10^5	61.44%	-4.35×10^5	60.58%
	Canopy I	0.932 ± 0.027	-4.35×10^5	58.78%	-4.36×10^5	61.61%	-4.35×10^5	64.46%	-4.35×10^5	58.78%
	Canopy II	1.008 ± 0.053	-4.35×10^5	58.78%	-4.35×10^5	62.30%	-4.36×10^5	61.69%	-4.35×10^5	58.78%
100	EM	56.640 ± 1.060	-3.93×10^5	83.95%	-3.94×10^5	82.33%	-3.94×10^5	83.44%	-3.94×10^5	82.77%
	SEM	4.006 ± 0.050	-3.93×10^5	83.99%	-3.93×10^5	83.37%	-3.94×10^5	83.05%	-3.95×10^5	83.44%
	Canopy I	1.220 ± 0.025	-3.96×10^5	83.44%	-3.96×10^5	83.20%	-3.97×10^5	83.48%	-3.96×10^5	83.22%
	Canopy II	1.015 ± 0.029	-3.97×10^5	82.77%	-3.97×10^5	83.21%	-3.97×10^5	82.66%	-3.97×10^5	82.66%
CIFAR100 - Direct										
			Random I		Random II		KMeans++		CoverTree	
Clusters	**Method**	**s/iter**	**LLH**	**Purity**	**LLH**	**Purity**	**LLH**	**Purity**	**LLH**	**Purity**
100	EM	78.019 ± 10.702	2.86×10^6	14.27%	3.03×10^6	13.31%	3.09×10^6	13.84%	3.09×10^6	14.19%
	SEM	1.055 ± 0.095	2.93×10^6	14.08%	2.93×10^6	14.12%	2.86×10^6	14.75%	3.00×10^6	14.90%
	Canopy I	1.027 ± 0.095	3.20×10^6	12.98%	3.36×10^6	12.43%	3.21×10^6	13.55%	3.25×10^6	12.91%
	Canopy II	1.190 ± 0.099	2.99×10^6	12.87%	3.08×10^6	13.23%	3.28×10^6	13.72%	3.08×10^6	12.87%
500	EM	407.764 ± 18.160	3.37×10^6	25.19%	3.31×10^6	24.70%	3.27×10^6	26.03%	3.31×10^6	25.59%
	SEM	6.486 ± 0.613	3.39×10^6	25.14%	3.30×10^6	24.33%	3.36×10^6	26.16%	3.22×10^6	25.39%
	Canopy I	2.745 ± 0.225	3.38×10^6	22.35%	3.50×10^6	22.14%	3.45×10^6	24.03%	3.44×10^6	22.31%
	Canopy II	1.908 ± 0.152	3.17×10^6	22.68%	3.18×10^6	22.83%	3.19×10^6	24.91%	3.19×10^6	22.71%
CIFAR100 - Embedding										
			Random I		Random II		KMeans++		CoverTree	
Clusters	**Method**	**s/iter**	**LLH**	**Purity**	**LLH**	**Purity**	**LLH**	**Purity**	**LLH**	**Purity**
100	EM	12.589 ± 0.255	5.45×10^5	12.38%	5.50×10^5	12.14%	5.50×10^5	12.25%	5.46×10^5	12.59%
	SEM	0.491 ± 0.022	5.46×10^5	12.21%	5.53×10^5	11.57%	5.45×10^5	12.72%	5.47×10^5	12.68%
	Canopy I	0.315 ± 0.014	5.01×10^5	12.34%	5.04×10^5	11.96%	4.99×10^5	13.16%	5.06×10^5	12.30%
	Canopy II	0.313 ± 0.124	5.00×10^5	12.50%	5.02×10^5	11.97%	4.99×10^5	13.01%	5.02×10^5	12.29%
500	EM	62.520 ± 1.135	6.94×10^5	19.17%	6.96×10^5	18.93%	6.86×10^5	21.13%	6.86×10^5	21.05%
	SEM	2.276 ± 0.112	6.92×10^5	18.97%	6.93×10^5	18.64%	6.85×10^5	21.16%	6.85×10^5	21.20%
	Canopy I	0.963 ± 0.061	6.25×10^5	20.07%	6.21×10^5	19.19%	6.14×10^5	21.57%	6.24×10^5	20.04%
	Canopy II	0.333 ± 0.101	6.20×10^5	22.26%	6.16×10^5	21.61%	6.12×10^5	23.18%	6.18×10^5	22.25%

Table 1. Comparison of ESCA, Canopy I and Canopy II on cluster purity and loglikelihood on real, benchmark datasets–MNIST8m and CIFAR-100. Additionally, standard deviations are shown for 5 runs.

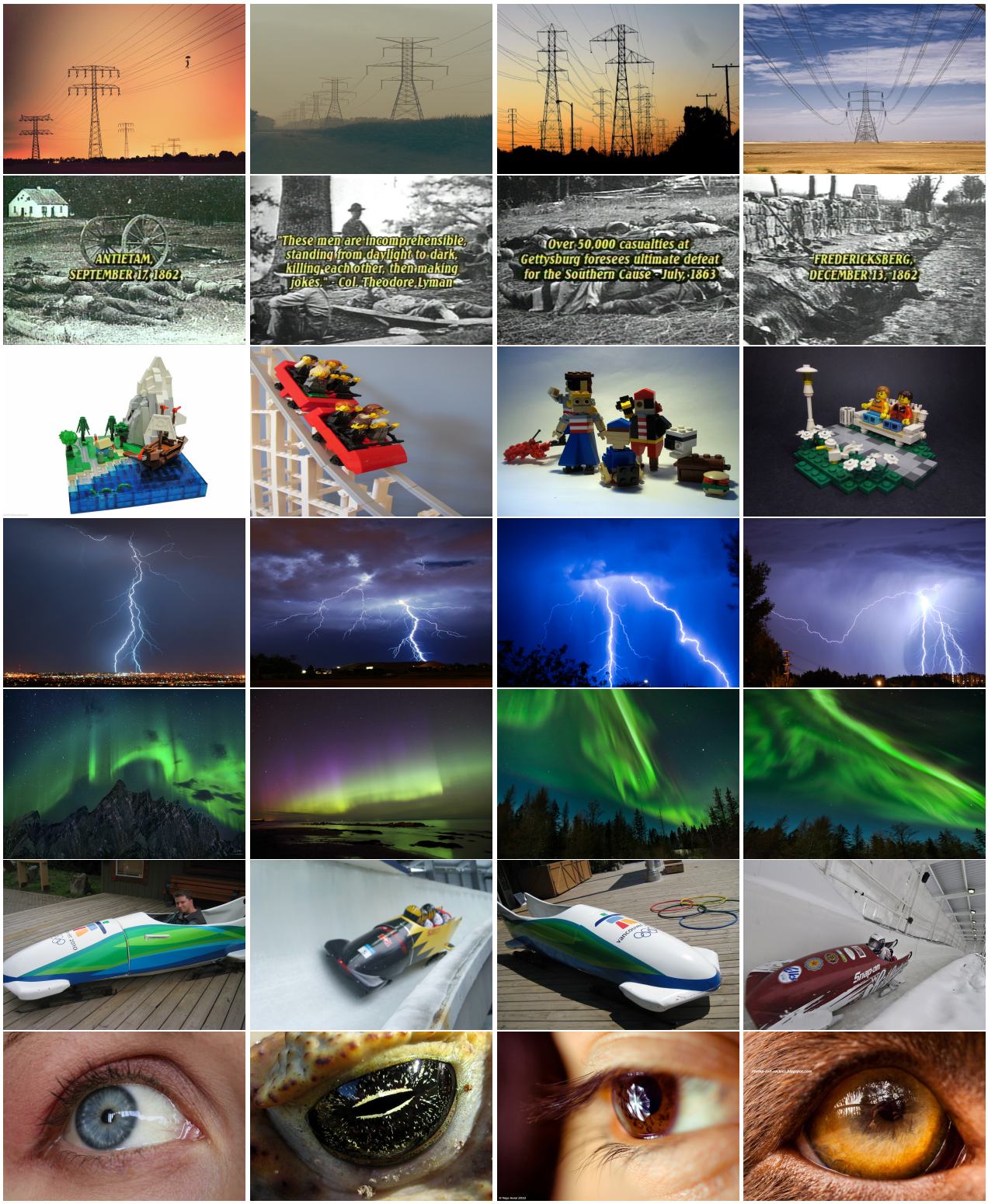


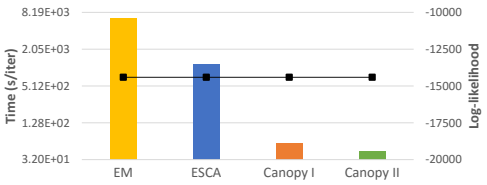
Figure 7. Illustration of concepts captured by clustering images in the feature space extracted by ResNet (He et al., 2015; 2016). Figure shows four closest images of seven more randomly selected clusters (one in each row) possibly denoting the semantic concepts of 'electrical transmission lines', 'image with text', 'lego toys', 'lightening', 'Aurora', 'buggy' and 'eyes'. Few of the concepts are discovered by clustering as Resnet received supervision only for 1000 categories (for example does not include label 'lightening', 'thunder', or 'storm'). Full set of 1000 imagenet label can be seen at `http://image-net.org/challenges/LSVRC/2014/browse-synsets`.

G. Graphical Explanation

We now present graphical intuitions about our approach.

Motivation

- Latent variable models (LVM), such as Mixture Models, Latent Dirichlet Allocation, are popular tools in statistical data analysis.
- They are used in diverse fields ranging from text, images, to user modelling and content recommendations.
- Inference is often slow



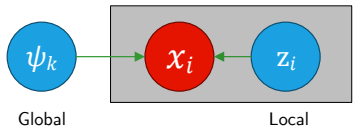
Inference Strategy

- Inference using Gibbs sampling, stochastic EM, or stochastic variational methods requires drawing from

$$p(z_i|x_i, \psi) \propto p(z_i)p(x_i|z_i, \psi)$$

- Assume exponential family, i.e.

$$p(x_i|z_i, \psi) = \exp(\langle \phi(x), \psi_{z_i} \rangle - g(\psi_{z_i}))$$

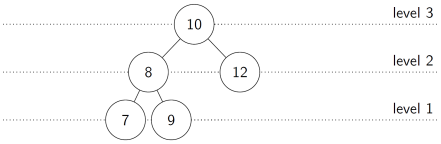


Insights

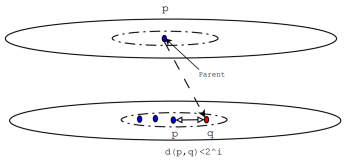
- For example assume we have following data:

- Two key observations
  - Points close by will have similar posteriors
  - No need to consider clusters far away
- Two tools to exploit the observations
  - Cover trees
  - MH sampling

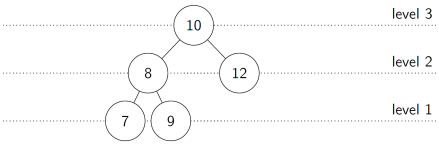
Cover Tree



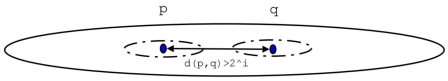
- Cover tree is a hierarchical data structure
- Covering property: $d(p, q) < 2^{\text{level}(p)}$



Cover Tree



- Cover tree is a hierarchical data structure
- Covering property: $d(p, q) < 2^{\text{level}(p)}$
- Separating property: $d(p, q) > 2^{\text{level}(p)-1}$



Computational Cost of Cover Trees

	Cover Tree	Ball Tree
Construction space	$O(n)$	$O(n)$
Construction time	$O(c^6 n \log n)$	$O(n^2)$
Insertion time (1 pt)	$O(c^6 \log n)$	$O(n)$
Query time (1 pt)	$O(c^{12} \log n)$	$O(n)$
Query time (n pts)	$O(c^{16} n)$	$O(n^2)$
	Beygelzimer *et. al.*, 2006	Omohundro, 1989

- Does not depend dimension of the data
- c: Expansion rate of data or Hausdorff dimension (special case of fractal dimension)

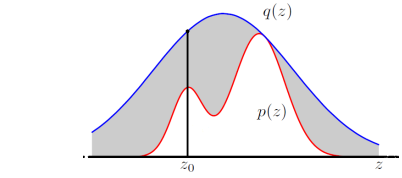
Insights

- For example assume we have following data:

- Two key observations
  - Points close by will have similar posteriors
  - No need to consider clusters far away
- Two tools to exploit the observations
  - Cover trees
  - Metropolis Hasting sampling

Metropolis Hasting Sampling

- Enables us to construct sound sampler that incorporates our intuitions



$q(\cdot)$ → Accept/ Reject → Sample from p

Only need to look at a few probabilities!

An easy to draw distribution

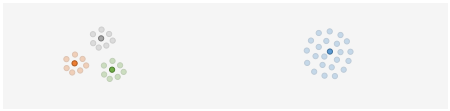
Acceptance probability $= \frac{p(z^+)q(z)}{p(z)q(z^+)}$

How to Design a Good Proposal?

- For example assume we have following data:

How to Design a Good Proposal?

- For example assume we have following data:

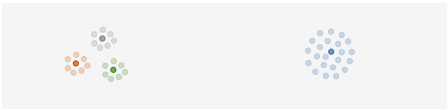


- Suppose we for each point x we can find surrogates $\bar{x}$

$$\|x - \bar{x}\| < \delta \Rightarrow |p(x|\psi) - p(\bar{x}|\psi)| < \epsilon$$

How to Design a Good Proposal?

- For example assume we have following data:



- Suppose we for each point x we can find surrogates $\bar{x}$

$$\|x - \bar{x}\| < \delta \Rightarrow |p(x|\psi) - p(\bar{x}|\psi)| < \epsilon$$

- Then $p(\bar{x}|\psi)$ becomes a good proposal for $p(x|\psi)$
  - Compute alias table and re-use for many points
  - Cost for sampling from proposal given alias table is O(1)

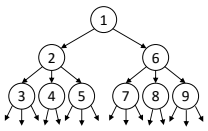
Outline

Canopy – Method I

- Build a cover tree on data points – Cost $O(N \log N)$

Data:

Cover Tree:

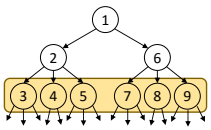


Canopy – Method I

- Build a cover tree on data points – Cost $O(N \log N)$
- Pick an accuracy level $\bar{j}$ having $\bar{N} = O(K)$ elements

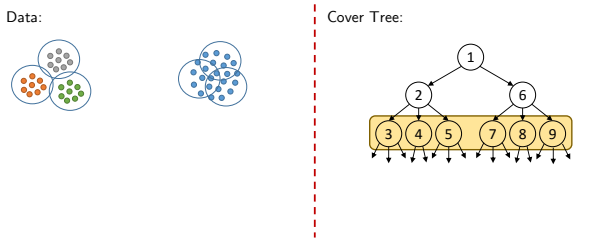
Data:

Cover Tree:



Canopy – Method I

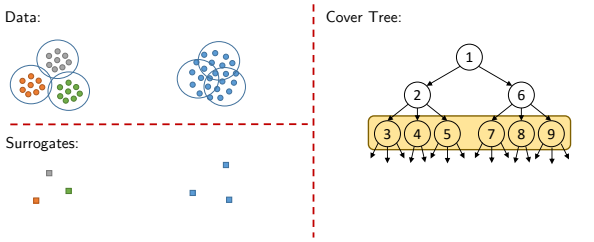
- Build a cover tree on data points – Cost $O(N \log N)$
- Pick an accuracy level $\bar{j}$ having $\bar{N} = O(K)$ elements



Carnegie Mellon University

Canopy – Method I

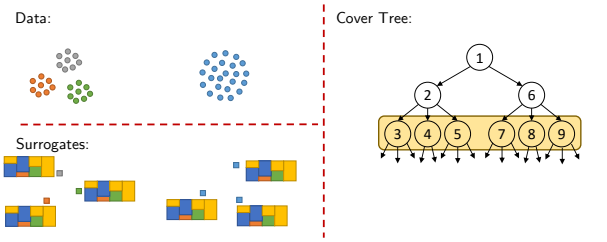
- Build a cover tree on data points – Cost $O(N \log N)$
- Pick an accuracy level $\bar{j}$ having $\bar{N} = O(K)$ elements



Carnegie Mellon University

Canopy – Method I

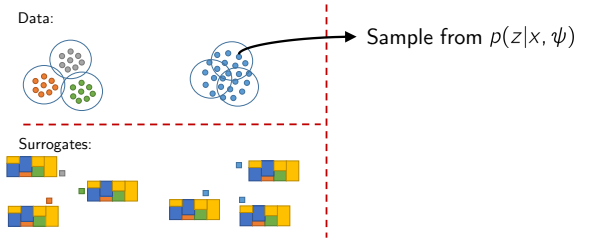
- Build a cover tree on data points – Cost $O(N \log N)$
- Pick an accuracy level $\bar{j}$ having $\bar{N} = O(K)$ elements
- Build alias tables for $p(z|\bar{x}, \psi)$ – Cost $O(K^2)$



Carnegie Mellon University

Canopy – Method II

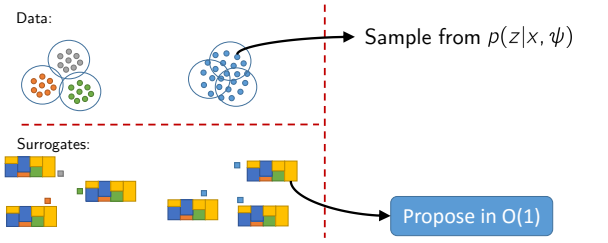
- For each observation x perform Metropolis-Hastings



Carnegie Mellon University

Canopy – Method II

- For each observation x perform Metropolis-Hastings

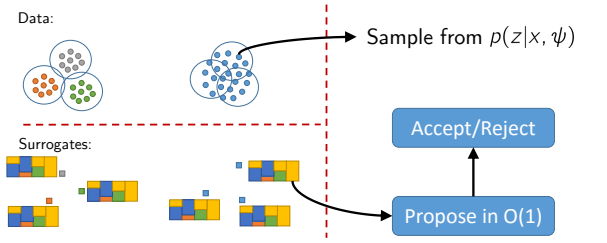


Carnegie Mellon University

Canopy – Method II

- For each observation x perform Metropolis-Hastings

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

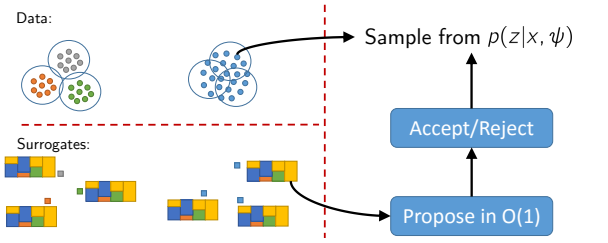


Carnegie Mellon University

Canopy – Method II

- For each observation x perform Metropolis-Hastings

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



Carnegie Mellon University

Canopy – Method II

- For each observation x perform Metropolis-Hastings
  - For exponential families:

$$\pi = \min\{1, \exp\{\langle \phi(x) - \phi(\bar{x}), \psi_{z'} - \psi_z \rangle\}} \geq e^{-2^{\bar{i}+2}L}$$