Canopy – Fast Sampling using Cover Trees

Manzil Zaheer

Abstract

Background. The amount of unorganized and unlabeled data, specifically images, is growing. To extract knowledge or insights from the data, exploratory data analysis tools such as searching and clustering have become the norm. The ability to search and cluster the data semantically facilitates tasks like user browsing, studying medical images, content recommendation, and computational advertisements among others. However, large image collections are plagued by lack of a good representation in a relevant metric space, thus making organization and navigation difficult. Moreover, the existing methods do not scale for basic operations like searching and clustering.

Aim. To develop a fast and scalable search and clustering framework to enable discovery and exploration of extremely large scale data, in particular images.

Data. To demonstrate the proposed method, we use two datasets of different nature. The first dataset comprises of satellite images of seven urban areas from around the world of size 200GB. The second dataset comprises of 100M images of size 40TB from Flickr, consisting of variety of photographs from day to day life without any labels or sometimes with noisy tags ($\sim$1% of data).

Methods. The excellent ability of deep networks in learning general representation is exploited to extract features from the images. An efficient hierarchical space partitioning data structure, called cover trees, is utilized so as to reduce search and reuse computations.

Results. Using the visual search-by-example on satellite images runs in real-time on a single machine and the task of clustering into 64k classes converges in a single day using just 16 commodity machines. Furthermore, the method is able to discover semantic concepts and "patterns of interest" in an unsupervised manner from Flickr images and unlabeled satellite imagery respectively.

Conclusions. A new scalable search and clustering method to explore massive datasets has been demonstrated on real-world datasets. Speed and scalability is guaranteed by a runtime that is logarithmic in the number of data points & clusters and polynomial in the expansion rate of the underlying parameter space per sample.

Keywords: Clustering, nearest neighbour search, deep networks, latent variable models, Gibbs sampling

DAP Committee members:
Barnabas Poczos (Machine Learning Department);
Taylor Berg-Kirkpatrick (Language Technology Institute);

1 Introduction

The digitally connected world of today involves potentially valuable data growing at a tremendous pace, in various forms ranging from text to images & videos to detailed user activity logs and shopping histories. The value generation is in transforming unstructured data into usable information, which is well organized, searchable and understandable, so as to enable visualization, summarization, personalized recommendation (of articles to read, videos to watch, locations to visit, restaurants to dine at, events to attend), fraud detection, medical image analysis, or more informed decision making in general.

For converting data into useful information nearest-neighbour search and latent variable models (LVM), such as clustering, have become the staple. More recently, the aforementioned algorithms in conjunction with Bayesian nonparametrics and deep networks have gained a lot of popularity [1, 2]. They have become a popular tool for modelling, visualization, inference, and exploratory analysis of multivariate data as they possess a desirable property of discovering the hidden thematic structure of objects in an interpretable format unlike unparsable results from deep networks.

With terabytes and petabytes of data pouring in, brute-force search and traditional methods for LVM inference are not up to the challenge. In the past decade, frameworks such as stochastic gradient descent (SGD) [3] and map-reduce [4] have enabled deep learning algorithms to scale to larger datasets. However, these frameworks are not always applicable to latent variable models or Bayesian nonparametrics which typically involve rich statistical dependencies and intractable gradients. Variational methods [5] and Markov chain Monte-Carlo (MCMC) [6] have thus become the *sine qua non* to infer the posterior distribution in these models, but they are often sequential and involve dense computation.

To deal with real life problems in industry, these inference procedures need to scale to massive datasets in a reasonable time with modern computational resources which are racks of fast, cheap, heavily multicored and unreliable machines. Fast nearest-neighbour search, given a metric space, has become a mainstay of information retrieval [7, 8, 9]. 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 method, we potentially need to evaluate *all* probabilities whereas for search we only need to find the *best* instance.

The problem is further exacerbated in case of images for which a general purpose representation and similarity metric is not established. A lot of work has been done to encode images by a vector of hand-crafted visual features [10]. Such hand-crafted feature vectors do not necessarily preserve the accurate semantic similarities of image pairs, especially across multiple domains, e.g. satellite images and photographs from day-to-day life. Deep networks and Bayesian models with rich hierarchies easily extract useful information for downstream tasks like classification or other predictions [11]. Often these learnt representations by the aforementioned algorithms exhibit generality for tasks outside of the task it was trained for, thereby enabling transfer learning and domain adaption. Well known examples include word embedding such as word2vec [12], learned from large, unstructured corpora have been shown to be surprisingly effective at capturing semantic regularities in language, or image features extracted from deep convnets such as resnet [13] or inception [14]. We want to utilize the ability of deep networks to learn representations along with $\mathcal{L}_2$ Euclidean metric for performing search and clustering. An important point to note though is that such representations learnt from deep networks are high dimensional dense vectors. So operating on these vectors put a

lot of pressure on CPU-RAM and network bandwidth, hindering scalability.

Problem being solved. We propose an algorithm to speed-up inference on LVM, thereby bridging the gap between meeting the need for speed and understanding the data in a very generalized setting. In particular, for this project, we want to focus on the following two problems:

- **Search:** Implement a visual search-by-example system capable of returning similar images in real time using a single machine
- **Clustering:** Design and implement image clustering system capable of handling 100M-1B images within reasonable time and with reasonable use of resources.

Outline of the approach. The images are embedded into an Euclidean space using the learned representation of a deep network trained for a classification task on the noisy tags available for a subset of the data. Next, the space is partitioned using a hierarchical data structure, called Cover Trees [7]. The advantages of this partitioning are two fold. Firstly, it allows fast look-up by reducing the search-space as we descend the hierarchy. Secondly, for clustering it allows flexibility for visualizing the data points in terms of varying degrees of granularity during the inference. (For example, exact calculations must be carried out for the few clusters near to point in consideration but approximations can be made for many clusters that are far away, reducing computational costs.)

In particular, for the first problem of search we simply implement cover tree based nearest neighbor search on the feature vectors obtained by the deep networks and launch a real-time service called "Terrapattern" http://www.terrapattern.com. But for the second problem of clustering, as mentioned earlier, no scalable solution exists. We aim to address this by marrying the fast lookup structure, i.e., cover trees which is used for nearest neighbor search, with an adaptive rejection sampler. This leads to a surprisingly simple design, which we term as "Canopy", for a variety of sampling-based inference algorithms. Moreover, we can obtain runtime guarantees for the sampler that depend only on the inherent dimensionality of the 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, it has been shown that in hierarchical Bayesian models, the Gibbs update step can be carried out in parallel as a stochastic cellular automata (ESCA) while still ensuring convergence [15]. This work combines the stochastic cellular automata idea leading to an extremely scalable and efficient system design. For example, in the case of Gaussian mixture models, the proposed method, Canopy, is much faster than EM or ESCA while achieving same accuracy as shown in Fig. 1.

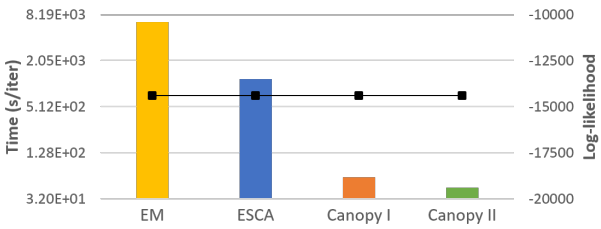


Figure 1: Canopy is accurate but much faster as compared to other methods like EM or ESCA [15]. The bar graph shows time per iteration while line plots the likelihood on held-out test set. Results shown are for inference of Gaussian mixture model with 32 million points having 4096 clusters at 1024 dimensions.

2 Related Works

There has been some work on speeding-up clustering using different approaches.

Using nearest neighbour search: Among the first works [16, 17] on accelerating inference on graphical models (like mixture models) employs nearest neighbour search like KD-trees for guiding the inference. But, KD-trees are not scalable with respect to dimensionality of the data points. This is because KD-trees could be very deep, even for datasets having small fractal dimensions. Also KD-trees do not provide any advantage and do not have special properties like covering.

Using coresets: Another line of work [18] to speed up mixture models and clustering involves finding a weighted subset of the data, called coreset. A coreset [19] is a weighted subset of the data such that the quality of any clustering instance evaluated on the coreset closely approximates the quality on the full data set. Models trained on the coreset are provably competitive with models trained on the original data set. Such an approach will reduce the effective number of samples, but then perform traditional inference on the coreset. However, the construction time of coreset is atleast $\Omega(nm)$, where n is the number of points in the dataset and m is the number of clusters, which is very expensive.

Commercial packages: Commercial frameworks like Spark [20] offer large scale clustering functionality as well.

In Table 1 we list the size of largest dataset reported to be used by each method listed above along with the size of the problem we are targeting in this work. We would like to mention that it is hard to compare the quality of clustering an algorithm achieves unless tested on the same dataset. As each of the reported work used a different dataset, we cannot provide a quality metric.

Paper	Dimension	Number of Points	Number of Clusters
Moore 1999 [16]	3	1,600,000	1,000
Spark 2015 [21]	10	2,000,000	-
Bachem 2015 [22]	57	11,620,300	2,000
Lucic 2016 [18]	74	145,751	50
This work	2048	100,000,000	64,000

Table 1: Comparison of the size of the largest dataset used by various works on clustering.

3 Background

In this section we provide a brief discussion about the cover tree data structure, upon which both of our systems are based, i.e. fast nearest neighbor search and clustering. Furthermore, the proposed technique for speeding up inference on latent variables depends crucially on a reformulation of the conditional probabilities, with which we begin our discussions.

3.1 Latent Variable Models

Most latent variable models, *e.g.* Gaussian mixture models, latent Dirichlet allocation [23], hidden Markov models, Dirichlet process clustering [24], or hierarchical generative models [25], have the structure of the form:

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

where x, z and θ_z denote observed variables, latent variables, and parameters of the conditional distribution respectively. Often the conditional distribution $p(x|\theta_z)$ belongs to the exponential family, which we assume to be the case in this work as well. Inference procedures for estimating z and θ_z on these models involving Gibbs sampling or 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 $\tilde{O}(m+n)$[1], where m is the number of latent classes and n is the number of data points.

The key motivation for this work is to make inference in mixtures of exponential families more efficient. The reasons for limiting to exponential families are two fold. First, most of the mixture models used in practice belongs to this class. Second, for speeding up one would require more structure which we obtain by assuming the form of distribution, i.e. exponential family. We make the following assumptions on $p(x|\theta_z)$ and $p(z)$. First, we 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 *alpha* is the concentration parameter. Second, we assume that the conditional model $p(x|\theta)$ in (1) is 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 $g(\theta_z)$ being the (normalizing) log-partition function.

Trying to find a metric data structure for fast retrieval is not necessarily trivial for the exponential family. In fact [26] and [27] design Bregman divergence based methods. Unfortunately they are more costly to maintain and have somewhat less efficient lookup properties. Finally, 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

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

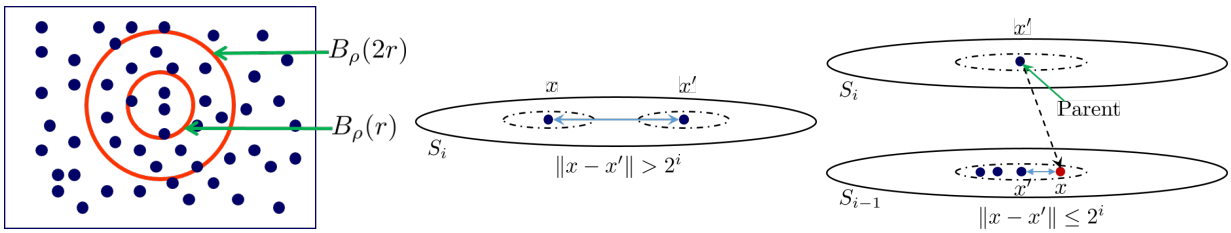
In this case, being able to group similar $\tilde{\theta}$ together allows us to assess their contributions efficiently without having to inspect individual terms. Further, we assume that $\left\|\tilde{\phi}(x_i)\right\| \leq R$ and $\left\|\tilde{\theta}_z\right\| \leq T$ for all i and for all $z \in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

3.2 Cover Trees

Cover Trees [7] and their improved version [28] 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 [29] 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 [7]), and for sampling (as we will be using in this paper).

The expansion rate of a set, due to [29] captures several key properties.

[1] It is a variant of big O notation that ignores logarithmic factors. $f(n) \in \tilde{O}(g(n))$ is shorthand for $\exists k : f(n) \in O(g(n) \log^k g(n))$.



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

Figure 2: Illustration of various properties of covering tree.

Definition 1 (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*

$$|B_\rho(r) \cap S| \geq l \Longrightarrow |B_\rho(2r) \cap S| \leq c\,|B_\rho(r) \cap S|\,. \tag{4}$$

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.

Cleary 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 [7]. 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 $O(c^6 n \log n)$.

We will make one final assumption in terms of the distinguishability of parameters θ_z. 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 [30]. This is complementary to the present paper.

4 Approach

In this section, we explain the details of our approach. The first step in solving either of the two problems of search and clustering is to embed the images into a metric space, i.e. feature extraction. After extraction of features, for the task of nearest neighbor search, we directly employ the cover tree based approach as described in [7, 28]. For the task of clustering, we explain details of our approach when the number of clusters is (a) moderate (b) large, after describing feature extraction and introducing some notation.

Feature extraction: We use the state of the art deep convolutional neural network (DCNN), based on the ResNet ("Residual Network") architecture [13, 31]. 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. The resnet is trained on a task of classification for a subset of images for which we have labels. In the process, our network learned which high-level visual features (and combinations of those features) are important for the classification of satellite imagery. 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.

Notation: 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.

Data tree ($\mathbb{T}_D$): Cover tree built with levels $S_\jmath$ on all available data using the sufficient statistic $\phi(x)$ for representation purposes. This costs at most $O(c^6 n \log n)$ time and it needs to be carried out only *once* for our setup. In fact, we only need to construct the tree up to a fixed degree of accuracy $\bar{\jmath}$ in case of moderate number of clusters. Once $\mathbb{T}_D$ has been constructed, we obtain and record prototypes $\bar{x}$ for each data point x through nearest neighbor queries on $\mathbb{T}_D$, incurring a one-time cost of $O(c^6 n \log n)$. 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 $\bar{x}$, for example, as described in Sec. 4.1. From the nesting property of cover trees, we have $\|\phi(x) - \phi(\bar{x})\| \leq 2^\jmath$ for all points.

Cluster tree ($\mathbb{T}_C$): Cover tree generated with cluster parameters $\tilde{\theta}_z$. For simplicity we assume here that the expansion rates of clusters and data are identical. Hence, construction of $\mathbb{T}_C$ costs at most $O(c^6 m \log m)$ time.

4.1 Canopy I: Moderate number of clusters

We introduce a variant of 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. Note that this algorithm will be used as a subroutine to perform inference whenever the number of clusters is considerably large (Sec. 4.2). In a nutshell, our approach for moderate number of clusters works as follows:

1. Construct $\mathbb{T}_D$ and pick a level $\bar{\jmath} \in \mathbb{Z}$ with accuracy $2^{\bar{\jmath}}$ such that the average number of elements per node in $S_{\bar{\jmath}}$ is $O(m)$.
2. For each of the prototypes $\bar{x}$, which are members of $S_{\bar{\jmath}}$, compute $p(z|\bar{x})$ using the alias method (see Appendix Sec. A.1) to draw from m components θ_z. By construction, this cost amortizes $O(1)$ per observation, *i.e.*, a total cost of $O(n \log n)$.
3. For each observation x with prototype $\bar{x}$, perform rejection sampling or Metropolis-Hastings sampling using the draws from $p(z|\bar{x}) =: q(z)$ as proposal to obtain samples from $p(z|x)$.

We can control the number of samples wasted by rejection sampling or convergence rate of the Metropolis-Hastings sampler with the choice of accuracy level $\jmath$. Note that for both kind of sampling,

the key quantity quantity controlling rejections or convergence is [32]:

$$\pi = \min_z \frac{p(z|\bar{x})}{p(z|x)} \geq e^{-\|\phi(x)-\phi(\bar{x})\|\|\theta_z\|} \geq e^{-2^{\bar{\jmath}+1}L}$$

for $\|\theta_z\| \leq L$. This follows from the Cauchy Schwartz inequality and property of cover tree that all children of $\bar{x}$ are no more than $2^{\bar{\jmath}+1}$ apart from each other. Larger the π is faster we converge or lesser is the number of the samples wasted [32]. Thus with choice of how much to descend the cover tree, we can trade-off between approximation and speed.

4.2 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, a larger m reduces our affordability in terms of granularity in x, for similar sampling complexity. The problem arises because we are trying to distinguish clusters at a level of resolution that is too fine. 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. 4.2.1) (b) Sample for a single observation x from a subset of clusters arranged in $\mathbb{T}_C$ (Sec. 4.2.2), when appropriate conditions are met in (a). We begin with the following 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 $\alpha(i, z) = 0$. 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 $\beta(\theta_z)$ as $\mathbb{T}_C$ has exactly m nodes. Notice that aggregated value $\beta(\theta_z)$ 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. Here $\hat{\imath}$ and $\hat{\jmath}$ are the accuracy levels of $\mathbb{T}_D$ and $\mathbb{T}_C$ respectively. This partition can be done by search over the levels.

4.2.1 Descending $\mathbb{T}_D$ and $\mathbb{T}_C$

Given the two cover trees $\mathbb{T}_D$ and $\mathbb{T}_C$ with accuracy levels $\hat{\imath}$ and $\hat{\jmath}$, we now iterate over the generated hierarchy, as shown in Fig. 3. We recursively descend simultaneously in both trees until the number of observations for a given cluster is too small. In that case, we simply default to sampling algorithm described in Sec. 4.2.2 for each observation in a given cluster.

The reason why it works is 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

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

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. Specifically, for each x we sample from $\mathbb{T}_D$ 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. 4.2.2 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.

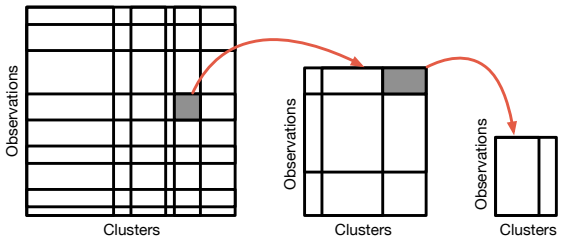


Figure 3: Hierarchical partitioning over both data observations and clusters. The accuracy at a particular level is indicated by rows for observations and columns for clusters. Once we sample clusters at a coarser level, we descend the hierarchy and sample at a finer level, until we have few number of points per cluster. We then default to Sec. 4.2.1 for rejection sampler.

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 move probabilities are always at least as high as the bounds derived in the previous section 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.

4.2.2 Sampling for a single observation x

Let x be the single observation for which we want to sample z from possibly a subset of clusters 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 use MH sampling or alternatively use a rejection sampler. We describe the latter approach in detail below, with its theoretical analysis being deferred to Appendix Sec. B. 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 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 $\mathbb{T}_C$, as follows:

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

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

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

For our experiments, we observe that the following simplification for 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 cover tree $\mathbb{T}_C$. For the other clusters, we assign the uniform probability equivalent to least out of the clusters whose posterior is computed exactly.

5 Experiments

5.1 Design

We now present some empirical studies for our fast sampling techniques. In order to illustrate the effectiveness of the proposed method in terms of both space and time, we evaluate on Gaussian

Mixture model (GMM), a widely used probabilistic model for clustering. However, the proposed method can be applied effortlessly to any latent variable models like Topic Modeling through Gaussian Latent Dirichlet allocation (Gaussian LDA) [2] or K-means. We picked GMMs due to its wide-spread application in various fields spanning computer vision, natural language processing, neurobiology, etc. For each of the setups, we compare two approaches (`Canopy I`, Section 4.1 and `Canopy II`, Section 4.2) with both the traditional Expectation Maximization (`EM`) [33] and Stochastic EM through ESCA (ESCA) [15] using execution time and likelihood on a held out TEST set. To elaborate, for all the four models (Canopy I, Canopy II, EM, ESCA), parameters are learnt using TRAIN set and Likelihood of the TEST set is used as evaluation.

5.2 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 small cluster of 16 Amazon EC2 c4.8xlarge nodes connected through 10Gb/s Ethernet. Each node has 36 virtual threads per node and 60GB of memory. For running the deep networks, we utilize a mix of nVidia 970 and 980 GPUs.

5.3 Data

Synthetic data: In order to demonstrate the correctness of our method and effects of varying number of points or number of cluster, we generate many synthetic datasets. Here, the data points are assumed to be generated from m Gaussian probability distributions parameterized by (μ_i^*, Σ_i^*) for $i = 1, 2, \cdots, m$, with mixing 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 being 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 $i = 1, 2, \cdots, m$, (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.

Real world data: In order to demonstrate the effectiveness of our method on real world data, we collected two datasets of very different nature. The first dataset is satellite-view images of earth obtained from Google Maps API [2]. We downloaded satellite images from Google Earth for metropolitan areas of Pittsburgh, San Francisco, New York City, Detroit, Berlin, Miami, and Austin. The satellite images are obtained as 'tiles' at various zoom levels. We collected the tiles at "level19", which represent patches about 58 meters across. Altogether more than geographical region spanning more than 12700km^2 is fully searchable.

We also download the crowdsourced labeled maps from the OpenStreetMap project [3], which has generously categorized a few parts of the world with its Nominatim taxonomy [4] using 466 value of the Nominatim categories (such as "airport", "marsh", "gas station", "prison", "monument", "church", etc.). The dataset consists of approximately 1,000 satellite images per category. This small labeled data can help guide the larger unsupervised tasks.

[2] https://developers.google.com/maps/documentation/javascript/maptypes
[3] https://www.openstreetmap.org/
[4] http://wiki.openstreetmap.org/wiki/Nominatim/Special_Phrases/EN

The second dataset is a large collection of images representing wide concepts in real word. To begin with we use Wordnet [5] which has an ordered sets of cognitive synonyms (synsets), each expressing a distinct concept. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet, majority of them are nouns (80,000+). For each synset, we take top three words and for each of these three words we search the large image hosting website Flickr API[6]. The API returns unique IDs of the result images with rank for each keyword. Union of these results are taken and the top 5000 are downloaded in JPEG format along with metadata (e.g. tags, comments). This way, we ended up with 100+ million images with weak labels occupying 40TB. (Note: We have only 100+ million images and not 400 million because for some synsets less than 5000 images are returned by Flickr).

The images collected this way form a non-trivial dataset. Some images maybe repeated as well under different synsets. The search results are also far from being perfect. For example, searching for alpha, yields many images of cats. Also due to polysemous nature of many English words, images returned by Flickr search would not be grouped nicely according to sense of the synset. For example searching "apache", results in some helicopters, some native tribes of America and some images related to Apache software foundation.

Finally for both datasets, images are reshaped such that the shorter dimension is 256 and the center patch of size 224 by 224 is taken to feed into the ResNet model.

5.4 Results

Nearest Neighbor Search: Quantitative study of nearest neighbor search using cover trees have been extensively studied in [7, 28]. This method was used in terrapattern, which is a prototype for helping people quickly scan extremely large geographical areas for specific visual features. We are particularly keen to help people identify, characterize, and track indicators which have not been detected or measured previously, and which have sociological, humanitarian, scientific, or cultural significance. An example finding all school bus depots in Pittsburgh, PA is shown in Figure 4. It is available for realtime use at http://www.terrapattern.com.

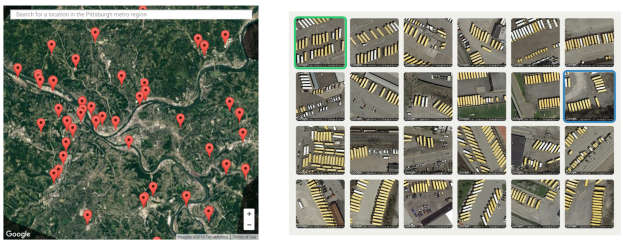
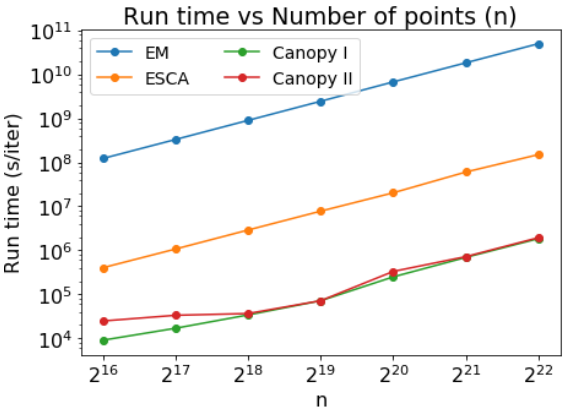


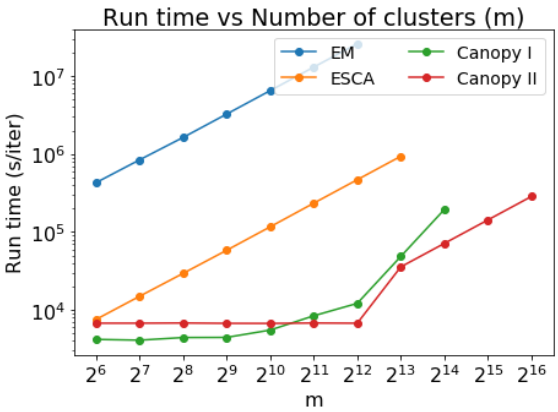
Figure 4: Terrapattern is an open-source tool for discovering patterns in unlabeled satellite imagery. A prototype for exploring the unmapped, and the unmappable. Here it identified some of Pittsburgh's finest school bus depots.

For feature extraction, we trained a 34-layer DCNN using hundreds of thousands of satellite images labeled in OpenStreetMap, teaching the neural network to predict the category of a place from a satellite photo. Our resulting model, which took 5 days to compute on an nVidia 980 GPU, has a top-5 error rate of 25.4% over 466 classes. In the appendix, we share some discoveries of our own, made with the Terrapattern system. It is important to point out that we did not trained on any of the categories shown, but Terrapattern was able to recognize them because of their common visual features. Thus showcasing the generalization power of the representation learnt by the ResNet.

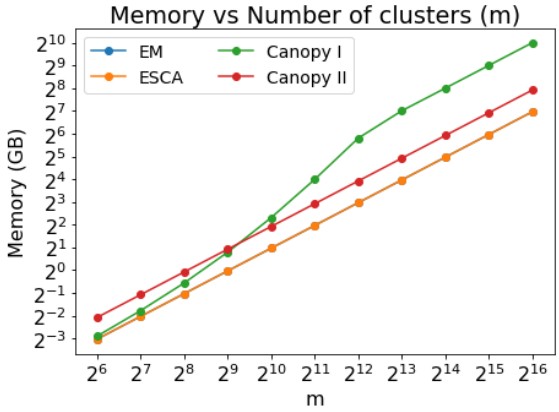
[5] https://wordnet.princeton.edu/
[6] https://www.flickr.com/services/api/flickr.photos.search.html



(a) Increasing number of points n with fixed $(m, d) = (2^{10}, 2^{10})$



(b) Increasing number of clusters m with fixed $(n, d) = (2^{10}m, 2^{10})$



(c) Increasing number of clusters m with fixed $(n, d) = (2^{10}m, 2^{10})$

Method	time (s/iter)	Log-likelihood
EM	5209.1	-14409.74
SEM	935.4	-14409.74
Canopy I	48.5	-14409.77
Canopy II	35.7	-14409.74
Ground truth	n/a	-14295.76

(d) Explicit comparison of different methods for the setting: $(n, m, d) = (32\text{mil}, 4096, 1024)$

Figure 5: Showing scalability of per-iteration runtime of different algorithms with increasing dataset size. From Fig. 5a and Fig. 5b 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 (data structure) for speed as seen from Fig. 5c. For instance, with $(n, m, d) = (32\text{mil}, 4096, 1024)$, we see that there is a speed-up of *150x* for a mere 2x memory overhead.

Gaussian Mixture Models - Synthetic Data: We begin with inference on Gaussian Mixture Models. We run all algorithms for a fixed number of iterations and vary n, m, d individually to further investigate the respective dependence on performance of our approach as shown in Figure. 5, 5d. It is in line with our claim that proposed method reduced the per iteration complexity from $O(nm)$ of EM/ESCA to $\tilde{O}(n + m)$. Table 5d gives the comparison of running times for the setting: $n = 32$ million, $m = 4096$, and $d = 1024$. 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, regarding the case reported in Table 5d for 2x increase in memory we get a speed up of 150x.

Figure 6: Illustration of concepts captured by clustering images in the feature space extracted by ResNet [13, 31]. Figure shows four closest images of three randomly selected clusters (one in each row) possibly denoting the semantic concepts of 'crowd', 'ocean rock scenery' and 'tricycle'. Few of the concepts are discovered by clustering as Resnet received supervision only for 1000 categories (does not include 'crowd').

Image Clustering: We extracted the image features with ResNet of 200 layers trained on ImageNet 1000 classes data set[7]. It took 5 days with 20 nVidia 970 GPUs to extract these feature for all the images. We then use Canopy II to cluster these images with $m = 64,000$, taking a total time of around 27 hours. The other methods, like EM or ESCA is too slow to complete within reasonable time-frame and thus we cannot compare against them quantitatively.

For a qualitative assessment of our clustering procedure, we randomly picked four clusters and showed four images closest to the means in Figure. 6 (each cluster in a row). We would like to highlight two important observations: (a) Even though the underlying deep architecture used to extract visual features, ResNet, is trained on 1,000 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) These images are associated with a variety of key words which do not semantically collate to the concepts in the image. For example, images in the 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 preprocessing 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 obtained from querying corresponding labels as keywords.

Furthermore, the clusters close by are semantically related. For example, we found the clusters of drummers and guitarists close to each other.

[7] https://github.com/facebook/fb.resnet.torch

6 Analysis and Discussion

Initialization: Recall that speed and quality of MCMC methods or variational algorithms depends 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++ [34], K-MC2 [35]. 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, it has no extra cost.) Our 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 entirity 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.

Reduction in Memory Bandwidth for Free: As a side-effect of the optimizations, we are able to reduce memory consumption and bandwidth to a great extent. ESCA has a minimal memory footprint because it stores only the data and the minimal sufficient statistics (by the very definition of minimal sufficient statistics, the footprint cannot be further reduced). Thus, ESCA substantially reduces memory costs, enabling larger datasets to fit in memory, and significantly reducing communication costs in distributed setups. As compared to variational methods, which require K memory accesses (one for each cluster) per data point, we only have a single access (for the sampled luster) per data point. Such reduced pressure on the memory bandwidth can improve performance significantly for highly parallel' applications. Moreover, Cover tree partitions the space so that we have to consider only a subset of the parameters thereby not requiring full memory access and saving bandwidth.

Load balance: We used Ganglia [36] to monitor the entire clusters performance. Ganglia, as a cluster-wide monitoring tool, can provide insight into overall cluster utilization and resource bottlenecks. In the appendix we have a figure (Figure 8) that shows the clusters snapshot during an iteration. In our algorithm, the dataset is partitioned evenly and work load is evenly distributed. We can see the memory optimization clearly worked and we are not bandwidth-bound, even when dealing with such dense vectors. A clustering implementation on spark [21] [8] also provides the resource utilisation through ganglia, we are clearly utilizing the resources much more efficiently.

7 Limitations

First of all our method requires some amount of labeled data, albeit small for representation learning. Next, we hope the learnt representations poses the desired property [11] of distinguishability of parameters θ_z. 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 summarily short of testing them all by brute force.

While the proposed method has tremendous potential for speeding up inference of a variety of LVM, in some cases, its applicability is not clear, e.g. Ising model. Fortunately, in most cases we

[8] http://users.eecs.northwestern.edu/~cji970/pub/cjinBigDataService2015.pdf, figure 6(a) on page 7

are interested in a model over a dataset in which the data is i.i.d. That is, we can fix our example as follows. Rather than applying canopy on a single Ising model at the granularity of pixels (over a single torus or grid), we instead attempt to speed up the Ising model at the granularity of the data (over multiple tori, one for each image).

8 Future

In the future we plan to integrate our method with the following methods:

Coresets: Another line of work to speed up mixture models and clustering involves finding a weighted subset of the data, called coreset. Models trained on the coreset are provably competitive with models trained on the original data set. Such approach will reduce the number of samples n, but then perform traditional inference on the coreset. Thus our approach can be combined and used after finding the coreset.

Inner product acceleration: Using the Locality Sensitive Importance Sampling [30], we can accelerate the inner product computation. Since the inner product is evaluated m times each iteration, it becomes the bottleneck for large m. A solution to overcome this problem was proposed in [30] by using binary hashing. This provides a good approximation and therefore a proposal distribution that can be used in a Metropolis-Hastings scheme without an excessive rejection rate.

9 Conclusion

A new scalable search and clustering method to explore massive datasets has been demonstrated on real-world datasets. In particular, we implemented a fast nearest neighbor search and we present a novel and efficient sampler for mixture models over exponential families using cover trees. We show that the use of cover trees over both data and clusters combined with Alias sampling can significantly improve sampling time in this class of models with no effect on the quality of the final clustering. Furthermore, speed and scalability is guaranteed by a runtime that is logarithmic in the number of data points & clusters and polynomial in the expansion rate of the underlying parameter space per sample. Thereby, becoming a prototype for exploring the *unheard*, the *unmapped*, and the *unmappable*.

10 Acknowledgements

We thank Satwik Kottur, Aman Tiwari, and Zichao Yang, fellow students at CMU for assistance with coding and web-deployment. Also we would like to thank Prof Barnabas Poczos; CMU, Alexander J Smola; CMU, Prof Golan Levin; CMU, and Amr Ahmed; Google for guidance, feedback and comments that greatly improved the work and the manuscript.

This research was partially supported by Amazon Inc in providing the computational resources and a Media Innovation Prototype Fund grant from the John S. and James L. Knight Foundation.

References

[1] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in neural information processing systems*, 2012, pp. 2222–2230.

[2] R. Das, M. Zaheer, and C. Dyer, "Gaussian lda for topic models with word embeddings," in *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics*, 2015.

[3] H. Robbins and S. Monro, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *CACM*, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[5] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine Learning*, vol. 37, no. 2, pp. 183–233, 1999.

[6] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice*. Chapman & Hall, 1995.

[7] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *International Conference on Machine Learning*, 2006.

[8] T. Liu, C. Rosenberg, and H. A. Rowley, "Clustering billions of images with large scale nearest neighbor search," in *Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on*. IEEE, 2007, pp. 28–28.

[9] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 1998, pp. 604–613.

[10] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[11] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[12] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., 2013.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014. [Online]. Available: http://arxiv.org/abs/1409.4842

[15] M. Zaheer, M. Wick, J.-B. Tristan, A. Smola, and G. L. Steele Jr, "Exponential stochastic cellular automata for massively parallel inference," in *AISTATS: 19th Intl. Conf. Artificial Intelligence and Statistics*, 2016.

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

[17] A. G. Gray and A. W. Moore, "N-body'problems in statistical learning," in *NIPS*, vol. 4. Citeseer, 2000, pp. 521–527.

[18] M. Lucic, O. Bachem, and A. Krause, "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*, 2016, pp. 1–9.

[19] D. Feldman, M. Schmidt, and C. Sohler, "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*. Society for Industrial and Applied Mathematics, 2013, pp. 1434–1453.

[20] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets." *HotCloud*, vol. 10, pp. 10–10, 2010.

[21] C. Jin, R. Liu, Z. Chen, W. Hendrix, A. Agrawal, and A. Choudhary, "A scalable hierarchical clustering algorithm using spark," in *Big Data Computing Service and Applications (BigDataService), 2015 IEEE First International Conference on*. IEEE, 2015, pp. 418–426.

[22] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause, "Approximate k-means++ in sublinear time," in *Conference on Artificial Intelligence (AAAI)*, 2016.

[23] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. Cambridge, MA: MIT Press, 2002.

[24] R. Neal, "Markov chain sampling methods for dirichlet process mixture models," University of Toronto, Tech. Rep. 9815, 1998.

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

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

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

[28] M. Izbicki and C. R. Shelton, "Faster cover trees," in *Proceedings of the Thirty-Second International Conference on Machine Learning*, 2015.

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

[30] A. Ahmed, S. Ravi, S. Narayanamurthy, and A. Smola, "Fastex: Hash clustering with exponential families," in *Neural Information Processing Systems 25*, 2012, pp. 2807–2815.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[32] K. L. Mengersen and R. L. Tweedie, "Rates of convergence of the hastings and metropolis algorithms," *The Annals of Statistics*, vol. 24, no. 1, pp. 101–121, 1996.

[33] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society B*, vol. 39, no. 1, pp. 1–22, 1977.

[34] D. Arthur and S. Vassilvitskii, "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*, 2007, pp. 1027–1035. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283494

[35] O. Bachem, M. Lucic, S. H. Hassani, and A. Krause, "Approximate k-means++ in sublinear time," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, 2016, pp. 1459–1467. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12147

[36] M. Massie, B. Li, B. Nicholes, V. Vuksan, R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal *et al.*, *Monitoring with ganglia.* " O'Reilly Media, Inc.", 2012.

[37] A. J. Walker, "An efficient method for generating discrete random variables with general distributions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 253–256, 1977.

A Background

A.1 Alias Sampler

A key component is the alias sampler of [37]. 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.

A.2 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

$$e^{-\epsilon} p(x|\theta_z) \leq q_z \leq e^{\epsilon} p(x|\theta_z) \tag{11}$$

then it follows that a sampler drawing z from

$$z \sim \frac{q_z p(z)}{\sum_{z'} q_{z'} p(z')} \tag{12}$$

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:

$$r(z) = \frac{q_z p(z)}{\gamma} e^{-\epsilon} q_z^{-1} p(x|\theta_z) + \sum_{z'} (1 - e^{-\epsilon} q_{z'}^{-1} p(x|\theta_{z'})) \frac{q_{z'} p(z')}{\gamma} r(z) \tag{13}$$

$$= \frac{e^{-\epsilon}}{\gamma} p(z) p(x|\theta_z) + \frac{r(z)}{\gamma} \sum_{z'} q_{z'} p(z') - r(z) \frac{e^{-\epsilon}}{\gamma} \sum_{z'} p(x|\theta_{z'}) p(z') \tag{14}$$

$$= \frac{e^{-\epsilon}}{\gamma} p(z) p(x|\theta_z) + r(z) - r(z) \frac{e^{-\epsilon}}{\gamma} \gamma_T \tag{15}$$

$$r(z) = \frac{p(z) p(x|\theta_z)}{\gamma_T} \tag{16}$$

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

A.3 ResNet

Convolutional Neural Networks (CNNs) have achieved extraordinary performance in the visual domain, sometimes even surpassing human-level performance. Examples include image classification (He et al. (2015)), face recognition (Taigman et al. (2014)), handwritten digit recognition, and recognizing traffic signs (Ciresan et al. (2012)). Lately, CNNs have been applied to speech recognition using spectrogram features (Hannun et al. (2014)) and achieve state-of-the-art speech recognition performance.

B 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_D 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 will need to traverse the $\mathbb{T}_C$ cover tree 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 $c^2 \log n$ 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

$$e^x - 1 \leq x e^a \text{ for } x \in [0, a] \text{ and } \sum_{l=1}^{\infty} 2^{-l} = 1. \tag{17}$$

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 (17) 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 2 *The cover tree sampler incurs worst-case computational complexity per sample of*

$$O\left(|S_{\hat{\imath}}| + c^6 \log n + c^6 \log m + c^4 e^{2^{\hat{\imath}} \left\| \tilde{\phi}(x) \right\|} \right) \tag{18}$$

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).) It is only the size that matters. The aforementioned constants are all natural quantities inherent to the problems we analyze. 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.

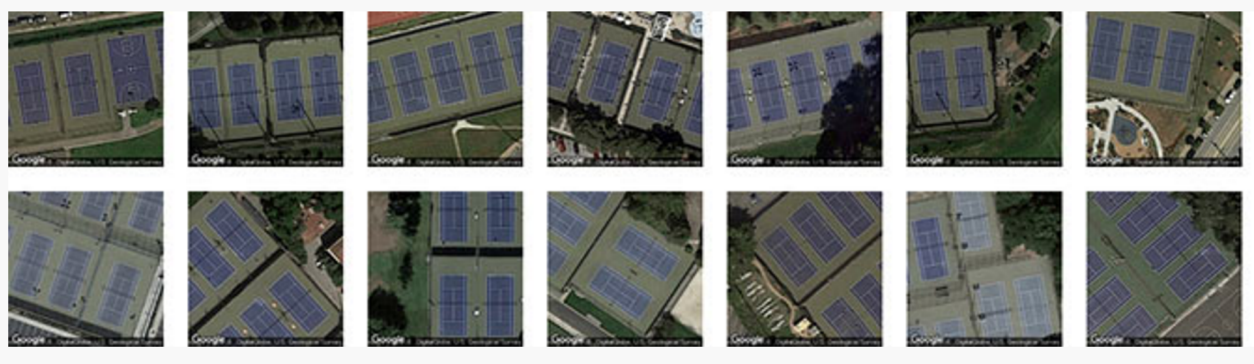
C More Results

Terrapattern is ideal for discovering, locating and labeling typologies that aren't customarily indicated on maps. These might include ephemeral or temporally-contingent features (such as vehicles or construction sites), or the sorts of banal infrastructure (like fracking wells or smokestacks) that only appear on specialist blueprints, if they appear at all. In this section, we share some discoveries of our own, made with the Terrapattern system. It is important to point out that the Terrapattern system was not trained on any of the categories shown below, but instead recognizes them because of their common visual features.

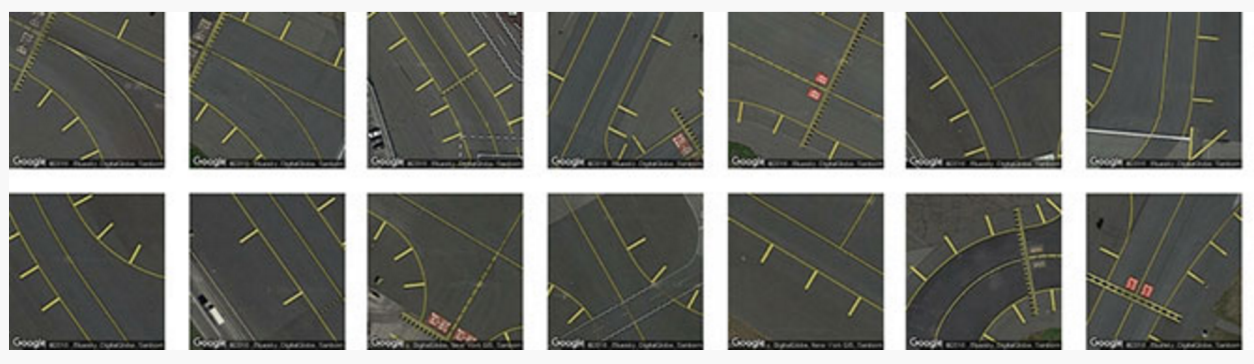
Here are some more interesting example searches that you can try in real-time at `http://www.terrapattern.com`:

- transformer station click here.
- cracked tarmac click here.
- solar panels click here.
- cars click here.
- baseball diamonds click here.
- USAF bombers click here.
- bridges click here.

(a) Golf course sand traps identified by our system in the Pittsburgh metro region click here.



(b) Purple tennis courts in the Bay Area, click here.



(c) Attractive runway markings from various New York ariports click here.

Figure 7: Some examples of interesting patterns found by our method.

D Resource utilization

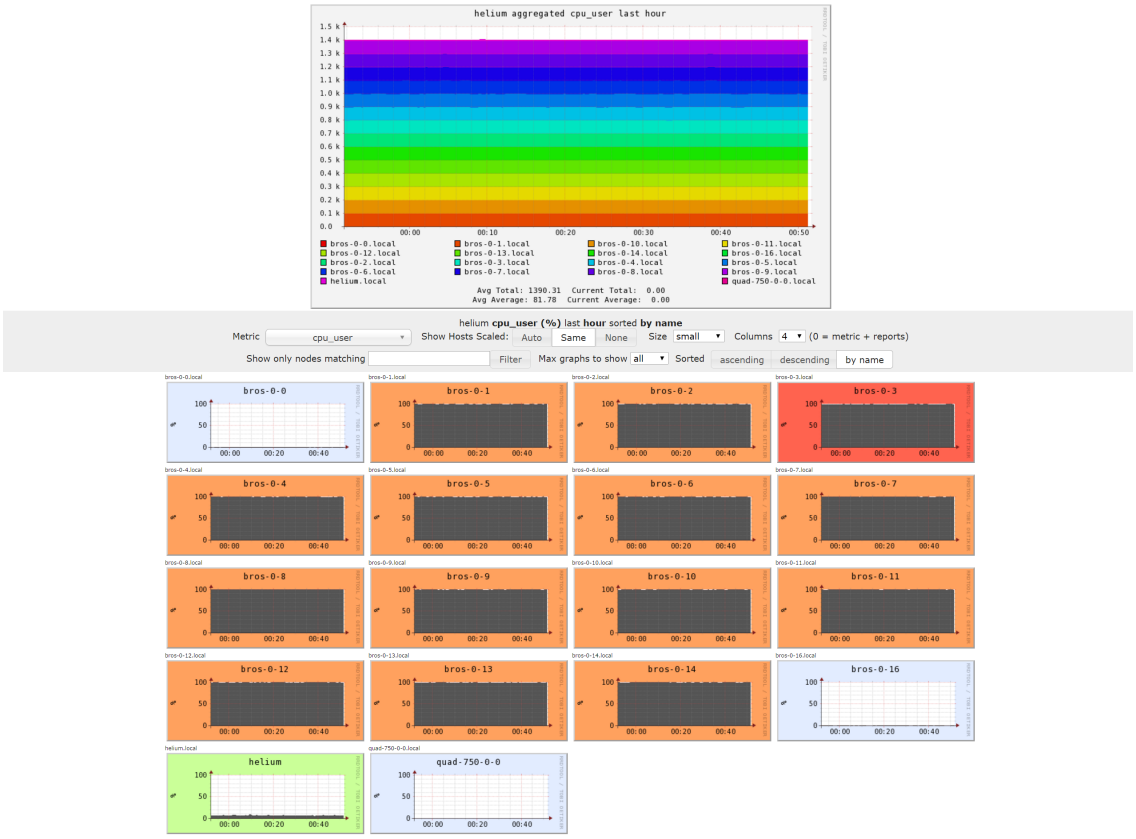


Figure 8: Snapshot of resource utilization at during an iteration of clustering. This shows we are not stalling or are bandwidth bound, *c.f.* [21, figure 6(a) on page 7] . (Note that machines named "bros-0-0", "bros-0-16" and "quad-750-0-0" are not part of the experiments.)