DENSITY SKETCHES FOR SAMPLING AND ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

There has been an exponential increase in the data generated worldwide. Insights into this data led by machine learning (ML) have given rise to exciting applications such as recommendation engines, conversational agents, and so on. Often, data for these applications is generated at a rate faster than ML pipelines can consume it. In this paper, we propose Density Sketches(DS) - a cheap and practical approach to reducing data redundancy in a streaming fashion. DS creates a succinct online summary of data distribution. While DS does not store the samples from the stream, we can sample unseen data on the fly from DS to use for downstream learning tasks. In this sense, DS can replace actual data in many machine learning pipelines analogous to generative models. Importantly, unlike generative models, which do not have statistical guarantees, the sampling distribution of DS asymptotically converges to underlying unknown density distribution. On the top DS is a one-pass algorithm which can be computed on data streams in compute and memory constrained environments include edge devices

1 INTRODUCTION

With the advent of big data, the rate of data generation is exploding. For instance, Google has around 3.8 million search queries per minute, amounting to over 5 billion data points or terabytes of data generated daily. Any processing over this data, such as using the training of a recommendation model, would suffer from data explosion. By the time existing data is consumed, newer data is already available. In such cases, we need to discard a lot of data. One of the critical research directions is how to reduce data storage. In this paper, we present Density Sketches (DS): an efficient and online data structure for reducing redundancy in data.

Often data comes from an underlying unknown distribution, and one of the challenges in data reduction is maintaining this distribution. In DS, we approximately store the data distribution in the form of a sketch. Using this DS, we can perform point-wise density estimation queries. Additionally, we can sample synthetic data from this sketch to use in downstream machine learning tasks. In this paper, we show that data sampled from DS, asymptotically converges to the underlying unknown distribution. We can also view density sketches through the lens of coresets. Specifically, DS is a compressed version of grid coresets. Grid coresets are the oldest form of coresets, giving lower additive errors than modern coresets. However, grid coresets are generally prohibitive as they are exponential in dimension (d). DS enables us to approximate grid coresets with the dependence of memory usage depending on the actual variety in the data instead of being exponential in d. Also, DS provides a streaming construction for this coreset.

In this paper, we focus more on the density estimation and sampling aspects of DS. Sampling from a distribution described using data requires estimating the underlying distribution. Popular methods to infer the distribution and sample from it belong to the following three categories: 1. Parametric density estimation (Friedman et al., 2001) 2. Non-parametric estimation - Histograms and Kernel Density Estimators (KDE) (Scott, 2015) 3. Learning-based approaches such as Variational Auto Encoders (VAE), Generative Adversarial Networks (GANs), and related methods (Goodfellow et al., 2014; 2016). Generally, parametric estimation is not suitable to model most real data as it can lead to large, unavoidable bias from the choice of the model (Scott, 2015). Learning the distribution, e.g., via neural networks, is one solution to this problem. Although learning-based methods have recently found remarkable success, they do not have any theoretical guarantees for the distribution of generated samples. Histograms and KDEs, on the other hand, are theoretically well understood. These statistical estimators of density are known to uniformly converge to the underlying true distribution *almost surely*. In this paper, we focus on such estimators, which have theoretical guarantees.

Storage of histograms and sampling from them is expensive because of an exponential number of partitions (i.e. bins). Apart from this, histograms also suffer from bin-edge problem: a slight variation in data can lead to large differences in the estimation of densities. KDEs are used to solve bin-edge problem. KDEs give smoother estimates of densities. While sampling from a KDE is efficient, KDE is expensive to store. KDEs require us to store the entire data. 'Coresets for KDE' is a good solution to storage problems of KDE. However construction of coresets, generally is quite expensive. In this work, we propose *Density Sketches*(DS) - a compressed sketch of density constructed in an efficient streaming manner. DS does not store actual samples of the data. But we can still efficiently produce samples from a KDE for specific kernels which, in turn, approximates $f(x)$. Being a compressed sketch we can tune the accuracy-storage trade-off of DS and we analyse this trade-off in theorem 1

2 Problem Statement and Related Work

Problem Statement: Formally, we want to create a data structure that has the following properties : (1) It sketches density information. (2) The size of the sketch is much smaller than the size of the data and does not scale linearly with it. (3) The construction is streaming and efficient. (4) We do not store any actual samples in the data structure created (for privacy reasons). (5) We want the sampling distribution, say $\hat{f}_S(x)$, obtained by sampling from these data structures to approximate the underlying true distribution $f(x)$.

The problem we are aiming to solve can be thought of as a data reduction problem and has been widely pursued in literature. The set of existing approaches can be broadly classified into two sections. (1) **Sampling based / Coresets** : Approaches such as clustering/importance sampling (Charikar & Siminelakis, 2017; Cortes & Scott, 2016; Chen et al., 2012) and coresets for KDE (Phillips & Tai, 2020; 2018) fall under this category. These approaches aim to find a small set of possibly weighted samples for a specific objective function such that the result obtained by applying the function to this small set is within a small approximation error of the result obtained by applying the objective function on complete dataset. The issue with these approaches is that of efficiency. Most of these algorithms require complicated computation on the entire data. Some streaming algorithms were recently proposed for coresets for KDE (Karnin & Liberty, 2019). However, even these algorithms need to perform $\mathcal{O}(m)$ (compactor size) computationally expensive operations per sample for large chunks of size (m) making them unsuitable for our purposes. (2) **Dimensionality reduction:** These approaches aim to reduce the width of the data matrix. Approaches such as Principle component analysis (PCA) are computationally expensive and require iterative computation over entire dataset. Random projections are an efficient streaming algorithm for dimensionality reduction. However, this approach leads to compressed data that increases linearly with original data size. As we can see existing approaches fall short of the requirements we have set out in the section 2

3 Background

3.1 Histograms and Kernel Density Estimation

Histograms and KDE (Scott, 2015; Scott & Sain, 2004) are popular methods to estimate the density of a distribution given a finite i.i.d sample of size n of points in R^d drawn from the true density, say $f(x)$.

Histogram: Histogram divides the support ($S \subset R^d$) of the data into multiple partitions. It then uses the counts in every partition to predict the density, $\hat{f}_H(x)$, at a point x. Formally the density predicted at the point $x \in S$ is given by

$$\hat{f}_H(x) = \frac{\mathcal{C}(\mathcal{B}(x))}{n.\mathcal{V}(\mathcal{B}(x))}$$

where $\mathcal{B}(x)$ identifies the partition of x, $\mathcal{C}(b)$ counts the number of samples in partition b, and $\mathcal{V}(b)$ measures the volume of parition b. $\hat{f}_H(x)$ integrates to 1 and hence $\hat{f}_H(x)$ is also an estimate of the underlying density function f(x). Regular histograms use hyper-cube partitions of width B aligned with the data axes. As B increases, the bias of estimate increases and its variance decreases. Histograms suffer from bin-edge problems where a slight change in data across the bin's edge can change predictions significantly

Kernel Density Estimation(KDE): KDE provides a smoother estimate of $f(x)$ which resolves the bin-edge problem of histograms. For a positive semi-definite kernel function $k(x, y) : R^d \times R^d \rightarrow R$

and data, say D, the KDE at point x is defined as

$$\hat{f}_K(x) = KDE(x) = \frac{1}{n} \sum_{i \in [1,n], x_i \in D} k(x, x_i)$$

Kernel functions are positive, symmetric, and may be normalized to integrate to 1. Gaussian, Epanechnikov, Uniform, (Friedman et al., 2001) etc. are some of the most widely used kernels. A smoothing parameter B also parameterizes kernel function. It determines the standard deviation parameter for Gaussian kernel function. For uniform and Epanechnikov kernel functions, B is the width of window around x where kernel is non-zero. Again, as B increases, the bias of KDE increases, and its variance decreases. Histograms estimator and KDE, both uniformly converge to underlying true distribution asymptotically. However, both suffer from the curse of dimensionality. To get a decent estimate of density in high dimensions, the number of samples needed is exponential in dimensions. For the Density Estimation task, dimensions of 4-50 are considered large. (Wang & Scott, 2019)

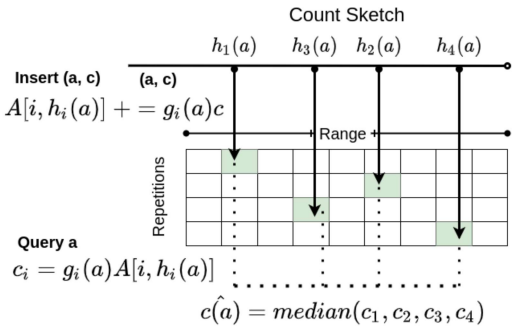


Figure 1: Countsketch, sketching and query

3.2 Count Sketches

The count sketch (CS) (Cormode & Muthukrishnan, 2009; Charikar et al., 2002), along with its variants, is one of the most popular probabilistic data structures used for the heavy hitter problem. Given a stream of (a_t, c_t) key-value pairs, $a_t \in \mathcal{U}$, CS stores the compressed total counts for each of the keys in a small $K \times R$ array of integers and can be queried to retrieve this total count, $c(a_t)$. CS offers a probabilistic solution in memory logarithmic in total number of unique keys. There is a standard memory accuracy trade-off for CS. Let m be the number of distinct keys and $\mathcal{C}$ be the vector of counts indexed by each key. For count median sketch (Charikar et al., 2002), the (ϵ, δ) guarantee $\mathbb{P}(|\hat{c(a)} - c(a)| > \epsilon ||\mathcal{C}||_2) \leq \delta$ is achieved using $\mathcal{O}(\frac{1}{\epsilon^2}\frac{1}{\delta}(\log m + \log |\mathcal{U}|))$ space. (Chakrabati, 2020). As can be seen from the above equation, the approximation accuracy for a particular key depends on how it compares to the $||\mathcal{C}||_2$. Specifically, CS can give very good approximation for keys with highest values in a setting where most of other keys have very low values. More discussion on CS can be found in Appendix F.

3.3 Locality Sensitive Hashing

Locality sensitive hashing(LSH)(Darrell et al., 2005) is a popular approach to solving approximate near neighbour problem. If a function h : $\mathcal{U} \rightarrow \{0, ...r-1\}$ for some r, is randomly drawn from the LSH family $\mathcal{L}$, the probability of collision of the hash values for two distinct elements u_1 and u_2 is

$$\mathbb{P}_{h \in \mathcal{L}}(h(u_1) == h(u_2)) \propto Sim(u_1, u_2)$$

Where Sim(.,.) is a some similarity metric corresponding to the LSH family. The probability collision is referred to as kernel of the LSH family, generally denoted by $\phi(.,.)$. Most kernels are positive, bounded, symmetric and reflective. We can use p independent LSH functions, $h_1, h_2, ...h_p$ to obtain a LSH function, $h^{(p)}(u) = (h_1(u), h_2(u), ..., h_p(u))$. The function $h^{(p)}$ has kernel $\psi(,.,) = \phi(.,.)^p$. We call p as the power of the LSH function. Popular LSH functions for $\mathcal{U} = R^d$ are L2-LSH, L1-LSH and SRP (signed random projection). More details on LSH functions can be found in (Darrell et al., 2005)

3.4 Uniform Sampling from Convex polytopes

Uniform Sampling from Convex spaces is a well-studied problem (Bélisle et al., 1993; Chen et al., 2017). For general convex polytopes, this is achieved by finding a point inside the polytope using convex feasibility algorithms and then running an MCMC walk inside the polytope to generate a point with uniform probability. In the case of regular convex polytopes like hypercubes and parallelopiped, uniform sampling is much simpler. Sampling a data point at random in a d-dimensional hypercube of width 1 is equivalent to sampling d real values uniformly in the interval $[0, 1]$. For sampling within a d-dimensional parallelopiped, we first locate $(d-1)$-dimensional hyperplane parallel to each face at a distance drawn uniformly from $[0, B]$ where B is the width of parallelopiped in that direction. The sampled point is then the intersection of these $(d-1)$ dimensional hyperplanes.

partitioning scheme	parameters	$\mathbf{bin(x) \in N^d; x \in R^d}$	Sample s from partition b
Regular Histogram	width: B	$bin(x)_i = \lfloor \frac{x_i}{B} \rfloor$	$r \in R^d, r_i \sim U[0,1])$ $s = B(bin_{id} + r)$
Aligned Histogram	widths: B= $(B_1, B_2, ...B_d)$	$bin(x)_i = \lfloor \frac{x_i}{B_i} \rfloor$	$r \in R^d, r_i \sim U[0,1])$ $s_i = B_i(b_i + r_i)$
Power d L1/L2-LSH	$W \in R^{d \times d}$ W=($w_1, w_2, ..., w_d$) $t : R^{d \times 1} t = (t_1, ..., t_d)$ width: B	$bin(x)_i = \lfloor \frac{\langle x_i, w_i \rangle + t_i}{B} \rfloor$	$r \in R^d, r_i \sim U[0,1])$ $y = B * (b + r)$ Solve $Ws = y - B$ to get s
Power k SRP	$W \in R^{d \times k}$ W=($w_1, w_2, ..., w_k$)	$bin(x)_i = sign\langle x_i, w_i \rangle$	MCMC algorithm constraints = $(sign(\langle x_i, w_i \rangle == b_i)$ + bounding box

Table 1: bin(x) for different partitioning schemes

4 Density Sketches

In DS, we aim to build a compressed non-parametric estimation object in an efficient streaming fashion. As KDEs give better approximation of underlying function $f(x)$ than histograms, we want to build DS as compressed KDE object. In order to achieve this, we use a nice connection between KDE and Histograms with LSH based partition function.

4.1 Histogram with LSH based Partition and Kernel Density Estimates

Any LSH function on R^d, will partition the space into different bins. Specifically, if power d L1/L2-LSH, these partitions will be polytopes in R^d. Similarly, power k SRP would give conical partitions with hyper-plane boundaries. We can employ histogram based estimation strategy on the top of these randomly drawn partitions. The density estimate using such a histogram would be

$$\hat{f}_H(x) \propto \frac{1}{n} \sum_{i \in [1,n], x_i \in D} \mathcal{I}(x_i \in bin(x))$$

where $\mathcal{I}$ is a indicator function. This estimate of the density has an expected value (over random partitions) equal to the KDE with the corresponding LSH kernel, $\phi(.,.)$

$$E_p(\hat{f}_H(x)) = \frac{1}{n} \sum_{i=1}^{n} P(x_i \in bin(x)) = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i, x) = \hat{f}_\phi(x)$$

The expectation is over random partitions. This connection between randomized histograms and KDE was first observed in (Coleman & Shrivastava, 2020). In order to better approximate KDE, we can combine results from multiple histograms with independent LSH functions. For example, if we use m independent histograms, say $H_1, H_2, ..., H_m$, then the density estimate can be written as

$$\hat{f}_H^{(m)}(x) = \frac{1}{m} \sum_{i=1}^{m} \hat{f}_{H_i}(x)$$

We can sample a data point from this set by first choosing a histogram randomly and then sample a point from that histogram. One can check that the sampling distribution, thus obtained, is $\hat{f}_H^{(m)}(x)$.

4.2 Constructing Density Sketches

Now that we have reduced the problem of KDE approximation to histograms, we will now show how to obtain a compressed representation of a histogram in a streaming fashion. We also show how to generate samples from this representation. First, let us establish some notation we will use

Notation: (1) Data D consists of n i.i.d samples of dimension d drawn from true distribution $f(x) : S \subset R^d \rightarrow R$. (2) $\mathcal{B}(x)$: ID of the partition in which point x falls. In case of p-power LSH functions, $\mathcal{B}(x) : R^d \rightarrow N^p$ and each bin can be identified with a unique tuple of p integers. In regular histogram, we have a tuple of d unique integers. For example, in regular histogram with width B, $\mathcal{B}(x)_i = \lfloor x_i / B \rfloor$. $\mathcal{B}()$ is generally parameterised with band-width parameter B which measures the size of the partition. Some partitioning schemes and sampling algorithms are mentioned in Table 1 (3) $\mathcal{M}$ CS with range R and repetitions K as described in section 3. (4) $\mathcal{H}$: Augmented min-heap of size H used with $\mathcal{M}$. Hence for a given partitioning scheme $(\mathcal{B}(), B)$, DS is parameterised by (K, R, H) and includes two data structures $\mathcal{M}(K, R)$, $\mathcal{H}(H)$.

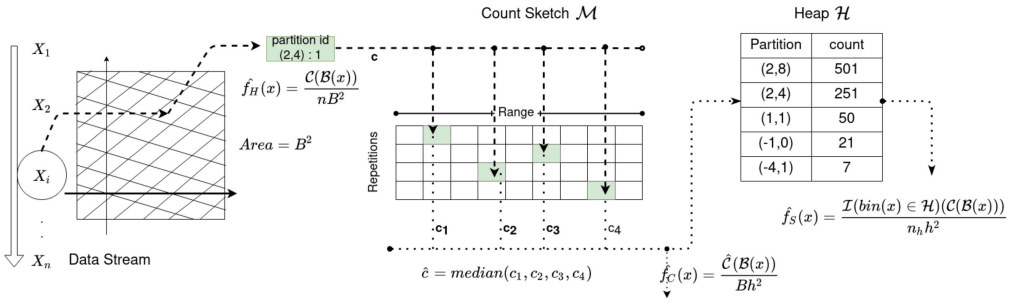


Figure 2: Overview of the sketching algorithm

Histogram has an exponential (in d) number of partitions. Hence, in high dimension, we cannot imagine to store histograms. However, most high dimensional real data is clustered and hence has highly sparse histograms. This does not help with histograms as post-pruning of histogram still requires building and enumerating it first. Nevertheless, the sparsity in histogram makes it a good candidate for heavy hitter problem. We use CS, $\mathcal{M}$, to store a compressed version of the histogram. Unfortunately, sampling with just $\mathcal{M}$ does not have a efficient solution. For the purpose of sampling we maintain a set of heavy partitions in the min-heap $\mathcal{H}$. We will discuss sampling in later subsections.

sketching $\mathcal{M}$: As shown in figure 4 and algorithm 1, we process the data in a streaming fashion. For each data point, say x, we first find the partition $b = \mathcal{B}(x)$. We increment the count of b by 1 by inserting $(b, 1)$ into $\mathcal{M}$. Along with each insertion, we also update $\mathcal{H}$. If the $\mathcal{H}$ is not at its capacity, we insert this b into the heap along with its updated count, $\mathcal{C}(b)$. In case the heap is at its capacity, we check b's updated count against the minimum of the $\mathcal{H}$. If b's count is found greater, we pop the minimum element from the heap and insert $(b, \mathcal{C}(b))$.

4.3 $\hat{f_C}(x)$: Estimate of density at a point

We can use $\mathcal{M}$ for querying the density estimate at a particular point. The algorithm for querying is presented in Algorithm 2 and is explained in the figure 2. Reusing notation from 3.1, the density predicted by the histogram can be written as $\hat{f}_H(x)$. When using the sketch, instead of using actual $\mathcal{C}(\mathcal{B}(x))$, we would use the its estimate from $\mathcal{M}$. Let this estimate be $\mathcal{C}(\hat{\mathcal{B}(x)})$. Then we can write the density predicted using countsketch as $\hat{f}_C(x)$

$$\hat{f}_H(x) = \frac{\mathcal{C}(\mathcal{B}(x))}{n.\mathcal{V}(\mathcal{B}(x))} \qquad \hat{f}_C(x) = \frac{\mathcal{C}(\hat{\mathcal{B}(x)})}{n.\mathcal{V}(\mathcal{B}(x))}$$

We know from CS literature, that $\mathcal{C}(\hat{\mathcal{B}(x)})$ is closely distributed around $\mathcal{C}(\mathcal{B}(x))$ and so we can expect $\hat{f}_C(x)$ to be close to $\hat{f}_H(x)$ and hence to $f(x)$. Note that though $\hat{f}_C(x)$ is a good estimate of density at a point x, the function $\hat{f}_C(.)$ is not a density function as it does not integrate to 1.

4.4 $\hat{f}^*_C(x)$: Estimate of density Function

In order to obtain a density function from the sketches, we have to normalize the function $\hat{f}_c(x)$ over the support. We can write $\hat{f}^*_C(x)$ as

$$\hat{f}^*_C(x) \propto \hat{\mathcal{C}(x)} \qquad \hat{f}^*_C(x) = \frac{\hat{\mathcal{C}(x)})}{\int \hat{\mathcal{C}(x)} dx} = \frac{\hat{\mathcal{C}(x)}}{\mathcal{V}(\mathcal{B}(x)) \sum_{b \in bins} \hat{\mathcal{C}(b)}} = \frac{\hat{\mathcal{C}(x)}}{\mathcal{V}(\mathcal{B}(x))\hat{n}}$$

It is easy to check the integral can be written as the sum over all the bins in the support. As is clear from the equations for $\hat{f}^*_C(x)$ and $\hat{f}_C(x)$, $n = \sum_{b \in bins} \mathcal{C}(b)$, is replaced by $\hat{n} = \sum_{b \in bins} \hat{\mathcal{C}(b)}$ to get a density function. We can check that $\hat{n}$ is an estimate of n using estimate of count for each bin from the DS.

4.5 $\hat{f_S(x)}$: Sampling from Density Sketches

$\mathcal{M}$ is a good enough representation for querying the density at a point. However, it is not the best data structure to efficiently generate samples. One naive way of sampling from these sketches is to

Algorithm 1: Constructing density sketch of $f(x)$

Result: Density Sketch (DS)
$f(x) : R^d \rightarrow R$: true distribution
$x_1, \ldots x_n \sim f(x)$: sample drawn from $f(x)$
$\mathcal{B}(()x) : S \subset R^d \rightarrow N^d$: partition function
$\mathcal{M}$: CS with range R, repetitions K
$\mathcal{H}(H)$: min-heap to store top H partitions

for $i \leftarrow 1$ **to** n **do**
  $b = \mathcal{B}(x_i)$
  $\mathcal{M}.insert(b, 1)$
  $c = \mathcal{M}.query(b)$
  $\mathcal{H}.update(b, c)$

Algorithm 2: query $\hat{f_C}(y)$, $y \in R^d$

Result: $\hat{f_C}(y)$
$y \in R^d$
$b = \mathcal{B}(y)$
$c = \mathcal{M}.query(b)$
return $(c/(n\mathcal{V}(b)))$

Algorithm 3: sample $y \in R^d$ such $y \sim \hat{f_S}(x)$

Result: y: sample from $f_S(y)$
P : multinomial distribution over bins s.t.
– $P(b) = (\mathcal{H}[b]/n_h)$ if $b \in \mathcal{H}$
– $P(b) = 0$ if $b \notin \mathcal{H}$
$b \sim P$
$y =$ UniformRandomPoint(b))
return y

randomly select a point in support of $f(x)$ and then do a rejection sampling using estimate $\hat{f_C}(x)$. However, given the enormous volume of support in high dimensions, this method is bound to be immensely inefficient. Another way is to choose a partition with probability proportional to the count of elements in that partition and then sample a random point from this chosen partition. It is easy to check that the probability of sampling a point x in this manner, precisely, is $\hat{f_H}(x)$ if we use exact counts and $\hat{f_C^*}(x)$ if we use approximate counts from CS. However, given that number of bins is exponential in dimension, sampling a bin proportional to its counts requires prohibitive memory and computation. This is essentially the reason why we needed a CS in the first place. Here, we further approximate the distribution by storing only top H partitions which contain most data points and discarding other partitions. As mentioned in 1, we can maintain top H partitions efficiently with an augmented heap $\mathcal{H}$. We then sample a partition present in this heap with probability proportional to its count and sample a random data point from this partition (Algorithm 3). The probability of sampling a data point whose bin is not present augmented heap is then zero. The distribution of this sampling algorithm is,

$$\hat{f_S}(x) = \mathcal{I}(\mathcal{B}(x) \in \mathcal{H}) \frac{\hat{\mathcal{C}(\mathcal{B}(x))}}{\hat{n_h}\mathcal{V}(\mathcal{B}(x))}$$

where $\hat{n_h} = \sum_{b \in \mathcal{H}} \hat{\mathcal{C}(b)}$, is the count-sketch estimate of total number of elements captured in all partitions present in heap. $\mathcal{I}(.)$ is the indicator function with values 0 or 1 evaluating the Boolean statement inside it. Let $ratio_h = \hat{n_h}/\hat{n}$ be the capture ratio of heap. It is easy to see that as capture ratio tends to 1, $\hat{f_S}(x)$ tends to $\hat{f_C^*}(x)$. Note that $\hat{f_S}(x)$ is indeed a density function.

5 ANALYSIS

Histogram and Kernel Density Estimators are well studied non-parametric estimators of density. Both of these estimators are shown to be capable of approximating a large class of functions (Scott, 2015). For example , with the condition of Lipschitz Continuity on $f(x)$, we can prove that point wise $MSE(\hat{f_H}(x)$ converges to 0 at a rate of $\mathcal{O}(n^{-2/3})$. Better results can be obtained for functions that have continuous derivatives. In our analysis we make all such assumptions on the lines of those made in (Scott, 2015); specifically, existence and boundedness of all function dependent terms that appear in the theorems below. We refer reader to (Scott, 2015) for in depth discussion on assumptions.

We restrict our analysis to convergence in probability for all the estimators discussed in this paper which is standard (Scott, 2015). In this section we consider the regular histogram partitioning scheme and show that our sampling distribution $\hat{f_S}(x)$ is an approximation of underlying distribution $f(x)$ and converge to it. However, similar analysis holds even for random partitioning schemes / KDE and is skipped here.

Mean integrated square error(MISE): MISE of an estimator of function is a widely used tool to analyse the performance of a density estimator.

$$MISE = E\int(\hat{f}(x) - f(x))^2 dx$$

A density estimator, with MISE asymptotically tending to 0, is a consistent estimator of true density and converges to it in probability. We would use this tool to make statements about convergence of our estimators. By Fubini's theorem, MISE is equal to IMSE (Integrated mean square error.

$$MISE(\hat{f})IMSE(\hat{f}) = \int E((\hat{f}(x) - f(x))^2)dx$$

We now present our main result of the paper,

Theorem 1 (**Main Theorem**$\hat{f}_S(x)$ to $f(x)$). *The probability density function of sampling, $\hat{f}_s(x)$, using a DS over regular histogram of width B, with parameters(K,R,H) created with n i.i.d samples from original density function $f(x)$, has an IMSE :*

$$IMSE(\hat{f}_S(x)) \leq 12(1 - ratio_h)^2 + 3(1 + 2\epsilon)(\frac{1}{nB^d} + \frac{\mathcal{G}(f)}{n} + o(\frac{1}{n}) + \frac{\#bins - 1}{KRnB^d})$$
$$+ 3(1 + 3\epsilon)\frac{B^2 d}{4}\mathcal{G}(\|\nabla f\|_2) + 3\epsilon(1 + 2\mathcal{G}(f) + B\sqrt{d}\int_{x\in S}(f(x)\|\nabla f\|_2)))$$

with probability $(1 - \delta)$, where $\delta = \frac{\#bins}{\epsilon^2 nR}$, $\#bins$ is the number of non-empty bins in histogram, $ratio_h$ is the estimated capture ratio as described in section 4.5 and $\mathcal{G}(g)$ is roughness defined as $\int g(x)^2 dx$

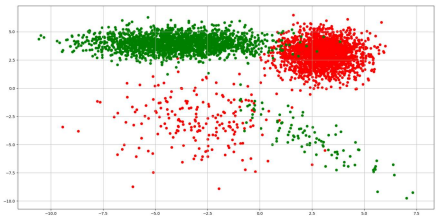
The dependence of IMSE on properties of f(x), such as roughness, is standard (Scott, 2015) and cannot be avoided.

Interpretation The estimator $f_S(x)$ of $f(x)$ is obtained by a series of approximations from $f(x) \rightarrow f_H(x) \rightarrow f_C(x) \rightarrow f^*_C(x) \rightarrow f_S(x)$. Hence in order to interpret this result, we break down the result above into multiple theorems enabling the reader to easily notice which step of approximations lead to what terms in the theorem above. We provide these details in Appendix A and B. We notice a few things from the theorem below

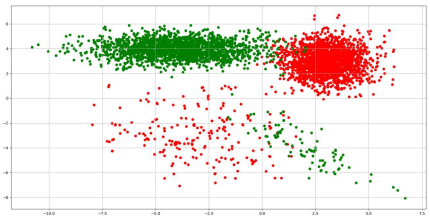
- In order for the IMSE to go to zero, both the terms $1/nB^d$ and $B^2 d$ should go to zero. That is to say that B has to decrease to zero and n has to increase exponentially faster than the rate at which B is decreasing. This is exactly the manifestation of curse of dimensionality.
- The term $\frac{\#bins-1}{KRnB^d}$ shows the effect of storing the counts in a compressed form. This term shows the effect of using larger smaller memory and as long as $(\#bins - 1)/KR$ is bounded, this term will go to zero as nB^d increases
- The term $12(1 - ratio_h)^2$ shows the effect of truncation that occurs due to using only heavy partitions. As can be seen , this term is data dependent and IMSE does not depend directly on H (number of partitions) but $ratio_h$. If we can capture the entire data in heap, then the term adds no penalty to IMSE. H, via $ratio_h$ controls the accuracy-memory trade-off of DS

6 Discussion

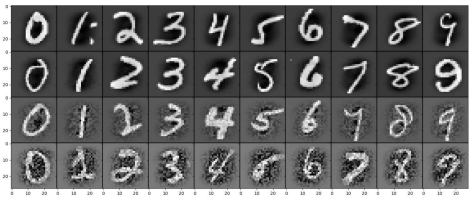
Curse of dimensionality: As DS are built over Histograms, they inherit the curse from Histograms : i.e. number of samples needed increase exponentially with dimension. With increased data collection, the issue of the unavailability of large amounts of data is fast vanishing. We want to emphasize that DS' advantages are best seen when data is humongous. DS can absorb tons of data and give better density estimates and samples without increasing their memory usage. Also, most real data in high dimensions is clustered or stays on a low-dimensional manifold. DS, throw away empty bins, and only store the histogram's populated bins. DS can deal with the curse of dimensionality better than Histograms. **DS on original data space:** Some data types like images do not reside in a space where the usual distances or cosine similarities imply conceptual similarity. On these data types, DS will not perform well. One way is to learn a transformation and create sketches on the transformed data. While this will give better performance in practice, we might loose theoretical guarantees for certain transformations.



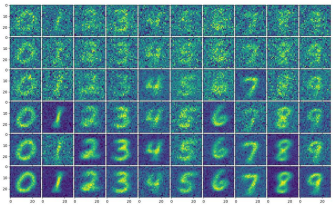
(a) sample drawn from true distribution



(b) sample drawn from Density Sketch



(c) MNIST Sample from Density Sketch using L2-LSH partitioning width: 0.01,0.1,1,10 from top to bottom



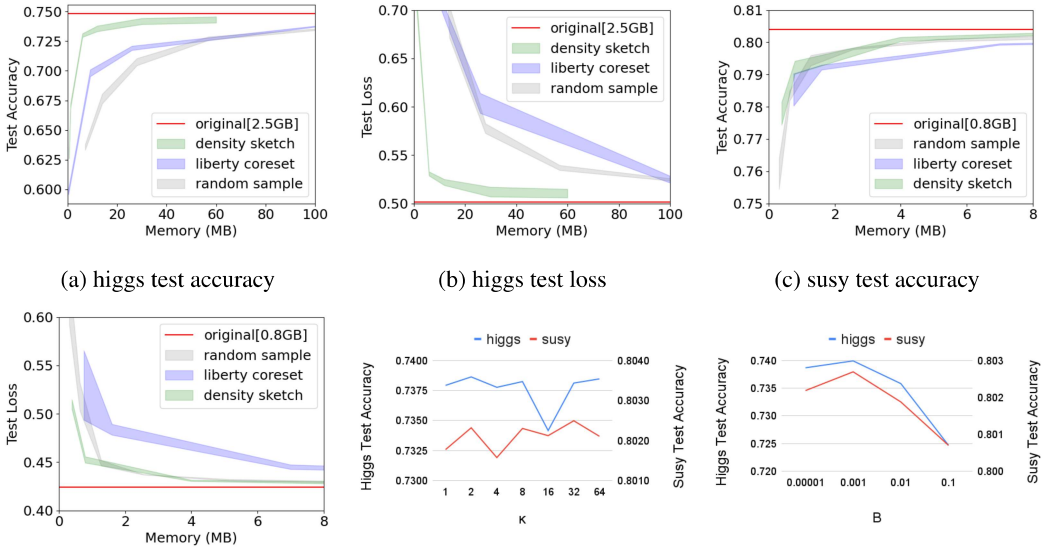
(d) MNIST Sample from Density Sketch using conical partitions (multiple signed random projections (SRP)). The rows have varying number of SRPs used - increasing from top to bottom

Figure 4: Visualization of samples drawn from Density Sketch. (a-b) DS captures density information. (c) Higher power LSH functions lead to fine DS behaving like random sample (d) Coarser partitions lead to samples that resemble "avg" of samples in data

7 EXPERIMENTS

Visualization of samples from density sketches: In the first set of experiments we provide a sanity check for DS in the form of visualization of data generated from DS. (1) In the first experiment, the Figure 4 (a) shows the samples drawn from the actual multi Gaussian distribution, whereas figure 4 (b) shows the samples drawn from the DS built on the samples from the true distribution. In this experiment, we use L2-LSH partitions with a B=0.25, K=3, R=50000 and H=3000. As can be seen, the two samples are indistinguishable. So DS do capture the density information. (2) Figure 4 (c) shows some samples drawn from DS built over the "MNIST" dataset (Chang & Lin, 2011) with varying bin-width sizes. (again R=50000, K=3 and H=5000) . We should notice that MNIST with 784 dimensions and 60K samples is not a ideal dataset for DS. In fact,with L2-LSH partitions the data would be so much scattered that every sample is contained in its own bin. If we make the bin-width finer then we should sample data points that are very close to random sample from original data. So in the worst conditions, DS converges to a random sample which we know is a good representation of data. 4 (d) shows results again with MNIST. However, here we have created conical partitions (created using multiple signed random projections). While L2-LSH partitions use power 784 L2-LSH functions, in this experiment we use smaller number of SRP functions(10-25, increases as we go to lower rows in image) thus promoting clustering. As is expected this coarse partitioning does show significant clustering and hence the images drawn from the partitions look like average of multiple samples from original data. The results support that DS can create samples which resemble the original data.

Evaluation of Samples on Classification Tasks: For most datasets, it is not possible to inspect samples visually. Hence we evaluate the quality of samples from DS by using them to train classification models. In these experiments, the data loader of training algorithm is replaced with sampler from DS. This sampler returns a training batch when requested by the algorithm. All the experiments are performed on Tesla P-100 GPUs with 16GB memory.
Datasets: We choose big datasets from the liblinear website (Chang & Lin, 2011), which satisfy the constraints of 1) data dimension less than 100 and 2) the number of samples per class greater than 1,000,000. Large Datasets is the main application domain for DS. Thus, we have datasets of Higgs (10M samples, 28 dimension) and Susy (5M samples, 18 dimension) for our experiments.
Baselines: For baselines, we consider random sample on same size and Liberty coresets proposed



(a) higgs test accuracy (b) higgs test loss (c) susy test accuracy

(d) susy test loss (e) varying K,R, with KR=1000000 (f) varying B

Figure 5: (a-d) Performance of density sketches, Liberty coresets and random sample on downstream classification task. The bar widths refer to $2 \times std$. DS performs consistently better at all memory sizes (e) Performance of DS is stable with K,R configs for decent budget KR=1000000, (f) Optimal B exists for DS performance. larger and smaller values of B lead to performance degradation

by (Karnin & Liberty, 2019) to compare DS performance. For Liberty coresets, we use m=100 as for larger m the process is very slow. Dimensionality reduction via random projections is another streaming algorithm but in these datasets we cannot get significant compression using dimensionality reduction (it can be $d\times$ at best where d is the dimension). So we cannot compare against this method. For more detailed discussion on baselines refer to appendix D.

Results: The DS have parameters: partition function(bandwidth B). We use L2-LSH partitions in all our experiments, sketch parameters K,R and heap parameter H. The memory of the DS used for sampling is affected by only the heap parameter (see appendix F for details on memory computation). In figure 5(f), we use the config (K=4,R=250000,H=100000) and vary B. It is clear from the figure that B=0.001 works best for these datasets. Lower and higher values of B affect the performance adversely. Larger B implies that we will capture more space than needed in single partition and smaller B implies that we will capture lesser data in the heap. So it is expected that a sweet spot for B exists. In figure 5(e), we fix (B=0.01,H=100000, KR=1000000) and vary K from 1 to 64. We can see that for a reasonable memory budget the results are stable with varying K. For the experiments in figure 5(a-d) we fix (B=0.01,K=5,R=250000) and vary H. This gives us DS of different sizes. We plot test accuracy and losses for DS, random samples and Liberty coresets for different sizes of memory used. The width of the band signifies the $2 \times$ std-dev of performance on three independent runs. As can be seen, for "Higgs" dataset , the accuracy of the model achieved on original data of size 2.5GB, can be closely reached by using a DS of size 50MB. So we get around **50x compression** ! We see similar results for datasets of "susy" (**100x compression**, 0.8GB) as well. The results show that DS is much more informative than Random Sample and Liberty Coresets. For more details on running the experiments (data processing, memory measurements etc) refer to appendix F.

Estimation of statistical properties of dataset: We also perform the experiments on covariance estimation task. The observations are similar to classification experiment. DS perform better than corresponding random sample. The results are presented in Appendix F for shortage of space.

8 Conclusion

In this paper, we talk about Density Sketches which is a streaming algorithm to construct a summary of density distribution from data. We show that new samples generated from this sketch asymptotically converges to the underlying distribution. Thus, DS comes with theoretical guarantees. Additionally, the cheap nature of online updates in Density Sketches, makes it an attractive alternative to constructing coresets for the data.In terms of coresets, DS can be viewed as a compressed form of randomized grid-coresets - one of the oldest form of coresets.

REFERENCES

Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

Claude JP Bélisle, H Edwin Romeijn, and Robert L Smith. Hit-and-run algorithms for generating multivariate distributions. *Mathematics of Operations Research*, 18(2):255–266, 1993.

Amit Chakrabati. *https://www.cs.dartmouth.edu/∼ac/Teach/data-streams-lecnotes.pdf*, volume 1. 2020.

Chih-Chung Chang and Chih-Jen Lin. Libsvm : a library for support vector machines. In *Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.* ACM Transactions on Intelligent Systems and Technology, 2011.

Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 1032–1043. IEEE, 2017.

Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pp. 693–703. Springer, 2002.

Yuansi Chen, Raaz Dwivedi, Martin J Wainwright, and Bin Yu. Vaidya walk: A sampling algorithm based on the volumetric barrier. In *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1220–1227. IEEE, 2017.

Yutian Chen, Max Welling, and Alex Smola. Super-samples from kernel herding. *arXiv preprint arXiv:1203.3472*, 2012.

Benjamin Coleman and Anshumali Shrivastava. Sub-linear race sketches for approximate kernel density estimation on streaming data. In *Proceedings of The Web Conference 2020*, pp. 1739–1749, 2020.

Graham Cormode and M Muthukrishnan. Count-min sketch., 2009.

Efren Cruz Cortes and Clayton Scott. Sparse approximation of a kernel mean. *IEEE Transactions on Signal Processing*, 65(5):1310–1323, 2016.

Trevor Darrell, Piotr Indyk, and Gregory Shakhnarovich. *Nearest-neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005.

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint arXiv:1406.2661*, 2014.

Zohar Karnin and Edo Liberty. Discrepancy, coresets, and sketches in machine learning. In *Conference on Learning Theory*, pp. 1975–1993. PMLR, 2019.

Jeff M Phillips and Wai Ming Tai. Improved coresets for kernel density estimates. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 2718–2727. SIAM, 2018.

Jeff M Phillips and Wai Ming Tai. Near-optimal coresets of kernel density estimates. *Discrete & Computational Geometry*, 63(4):867–887, 2020.

David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

DW Scott and SR Sain. Multi-dimensional density estimation handbook of statistics vol 23 data mining and computational statistics ed cr rao and ej wegman, 2004.

Zhipeng Wang and David W Scott. Nonparametric density estimation for high-dimensional data—algorithms and applications. *Wiley Interdisciplinary Reviews: Computational Statistics*, 11(4):e1461, 2019.