Space and Time Efficient Kernel Density Estimation in High Dimensions

Arturs Backurs[*]
TTIC
backurs@ttic.edu

Piotr Indyk
MIT
indyk@mit.edu

Tal Wagner
MIT
talw@mit.edu

Abstract

Recently, Charikar and Siminelakis (2017) presented a framework for kernel density estimation in provably sublinear query time, for kernels that possess a certain hashing-based property. However, their data structure requires a significantly increased super-linear storage space, as well as super-linear preprocessing time. These limitations inhibit the practical applicability of their approach on large datasets.

In this work, we present an improvement to their framework that retains the same query time, while requiring only linear space and linear preprocessing time. We instantiate our framework with the Laplacian and Exponential kernels, two popular kernels which possess the aforementioned property. Our experiments on various datasets verify that our approach attains accuracy and query time similar to Charikar and Siminelakis (2017), with significantly improved space and preprocessing time.

1 Introduction

Kernel density estimation is a fundamental problem with many applications in statistics, machine learning and scientific computing. For a *kernel function* $k : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$, and a set of points $X \subset \mathbb{R}^d$, the kernel density function of X at a point $y \in \mathbb{R}^d$ is defined as:[2]

$$\mathrm{KDE}_X(y) = \frac{1}{|X|} \sum_{x \in X} k(x, y).$$

Typically the density function is evaluated on a multiple queries y from an input set Y. Unfortunately, a naïve exact algorithm for this problem runs in a rectangular $O(|X||Y|)$ time, which makes it inefficient for large datasets X and Y. Because of this, most of the practical algorithms for this problem report approximate answers. Tree-based techniques [GS91, GM01, GB17] lead to highly efficient approximate algorithms in low-dimensional spaces, but their running times are exponential in d. In high-dimensional spaces, until recently, the best approximation/runtime tradeoff was provided by simple uniform random sampling. Specifically, for parameters $\tau, \epsilon \in (0, 1)$, it can be seen that if X' is a random sample of $O\left(\frac{1}{\tau}\frac{1}{\epsilon^2}\right)$ points from X, then $\mathrm{KDE}_{X'}(y) = (1 \pm \epsilon)\,\mathrm{KDE}_X(y)$ with constant probability[3] as long as $\mathrm{KDE}_X(y) \geq \tau$.

This approximation/runtime tradeoff was recently improved in [CS17], who proposed a framework based on *Hashing-Based Estimators (HBE)*. The framework utilizes locality-sensitive hash (LSH)

[*]Authors ordered alphabetically.

[2]We note that all algorithms discussed in this paper easily extend to the case where each term $k(x, y)$ is multiplied by a positive weight $w_x \geq 0$, see e.g., [CS17].

[3]The probability of correct estimation can be reduced to $1 - \delta$ for any $\delta > 0$ at the cost of increasing the sample size by a factor of $\log(1/\delta)$. Since the same observation applies to all algorithms considered in this paper, we will ignore the dependence on δ from now on.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Table 1: Comparison of runtime and space bounds. Notation: $\tau \in (0,1)$ denotes a lower bound for KDE values; d denotes the dimension; $\epsilon \in (0,1)$ denotes the approximation error.[4]

Algorithm	Query Time	# Stored hashes
Random Sampling	$O(d/\tau \cdot 1/\epsilon^2)$	$O(1/\tau \cdot 1/\epsilon^2)$
HBE	$O(d/\sqrt{\tau} \cdot 1/\epsilon^2)$	$O(1/\tau^{3/2} \cdot 1/\epsilon^4)$
This paper	$O(d/\sqrt{\tau} \cdot 1/\epsilon^2)$	$O(1/\tau \cdot 1/\epsilon^2)$

functions [IM98], i.e., randomly selected functions $h : \mathbb{R}^d \to U$ with the property that for any $x, y \in \mathbb{R}^d$, the collision probability $\Pr_h[h(x) = h(y)]$ is "roughly" related to the kernel value $k(x,y)$. HBE reduces the evaluation time to (about) $O\left(\frac{1}{\sqrt{\tau}}\frac{1}{\epsilon^2}\right)$. A recent empirical evaluation of this algorithm [SRB+19] showed that it is competitive with other state of the art methods, while providing significant (up to one order of magnitude) runtime reduction in many scenarios.

One drawback of HBE approach, however, is its space usage, which is super-linear in the dataset size. Specifically, the algorithm constructs $O\left(\frac{1}{\sqrt{\tau}}\frac{1}{\epsilon^2}\right)$ hash tables, and stores the hash of each data point in each table. Consequently, the additional storage required for the hashes is proportional to the number of tables times the number of data points. As mentioned above, we can uniformly subsample the dataset down to $O\left(\frac{1}{\tau}\frac{1}{\epsilon^2}\right)$ points, leading to an overall space usage of $O\left(\frac{1}{\tau^{3/2}}\frac{1}{\epsilon^4}\right)$, which is $O\left(\frac{1}{\sqrt{\tau}}\frac{1}{\epsilon^2}\right)$ times that of the simple random sampling approach. The increase in storage also effects the preprocessing time of the HBE data structure, which requires $O\left(\frac{1}{\tau^{3/2}}\frac{1}{\epsilon^4}\right)$ hashes computations due to having to store every point in every table. As τ and ϵ can be very close to zero in practice, these drawbacks may pose a substantial bottleneck in dealing with large datasets.

Our results. In this paper we show that the super-linear amount of storage is in fact *not needed* to achieve the runtime bound guaranteed by the HBE algorithm. Specifically, we modify the HBE algorithm in a subtle but crucial way, and show that this modification reduces the storage to (roughly) $O\left(\frac{1}{\tau}\frac{1}{\epsilon^2}\right)$, i.e., the same as simple random sampling. Table 1 summarizes the performance of the respective algorithms. Our main result is the following theorem.

Theorem 1. *Let $k(x,y)$ be a kernel function, for which there exists a distribution H of hash functions and $M \geq 1$ such that for every $x, y \in \mathbb{R}^d$,*

$$M^{-1} \cdot k(x,y)^{1/2} \leq \Pr_{h \sim H}[h(x) = h(y)] \leq M \cdot k(x,y)^{1/2}. \tag{1}$$

There exists a data structure for Kernel Density Estimation with the following properties:

- *Given a dataset $X \subset \mathbb{R}^d$ and parameters $\tau, \epsilon \in (0,1)$, we preprocess it in $O\left(\frac{1}{\tau} \cdot \frac{T_H M^3}{\epsilon^2}\right)$ time, where T_H is the time to compute a hash value $h(x)$.*
- *The space usage of the data structure is $O\left(\frac{1}{\tau} \cdot \frac{(S_X + S_H)M^3}{\epsilon^2}\right)$, where S_X is the space needed to store a point $x \in X$, and S_H is the space needed to store a hash value $h(x)$.*
- *Given a query point y such that $KDE_X(y) \geq \tau$, we can return with constant probability a $(1 \pm \epsilon)$-approximation of $KDE_X(y)$ in $O\left(\frac{1}{\sqrt{\tau}} \cdot \frac{(T_k + T_H)M^3}{\epsilon^2}\right)$ time, where T_k is the time to compute a kernel value $k(x,y)$.*

We empirically evaluate our approach on the Laplacian kernel $k(x,y) = e^{-\|x-y\|_1/\sigma}$ and the exponential kernel $k(x,y) = e^{-\|x-y\|_2/\sigma}$. Both are commonly used kernels, and fit into the framework as

[4]For simplicity, the bounds in the table assume that the kernel takes $O(d)$ time to compute, and that a hash value takes $O(d)$ time to compute. The kernels we consider have these properties (for bandwidth $\sigma = \Omega(1)$). See Theorem 1 for the full parameter dependence.

they satisfy the requirements of Theorem 1 with $M = O(1)$, $T_k = O(d)$, $T_H = O(\min\{d, d/\sigma\})$ and $S_H = O(\min\{d\log(1/\sigma), d/\sigma\})$, with high probability (over $h \sim H$). Our experiments confirm the analytic bounds and show that our approach attains a similar query time to approximation tradeoff as HBE, while using significantly less space and preprocessing time.

Our techniques. Our algorithm builds on the HBE approach of [CS17]. Recall that the algorithm selects $L = \Theta\left(\sqrt{1/\tau} \cdot 1/\epsilon^2\right)$ LSH functions $h_1 \dots h_L$, and creates L hash tables, such that for each $i = 1 \dots L$, each point $x \in X$ is placed in the jth table in bin $h_j(x)$. To estimate $\mathrm{KDE}_X(y)$, the algorithm selects one point from each bin $h_1(y) \dots h_L(y)$, and uses those points for estimation. To achieve the performance as in Table 1, the algorithm is applied to a random sample of size $s = O(1/\tau \cdot 1/\epsilon^2)$. The total space is therefore bounded by $O(sL) = O(1/\tau^{3/2} \cdot 1/\epsilon^4)$.

A natural approach to improving the space bound would be to run HBE on a smaller sample. Unfortunately, it is easy to observe that any algorithm must use at least $\Omega(1/\tau \cdot 1/\epsilon^2)$ samples to guarantee $(1 \pm \epsilon)$-approximation. Therefore, instead of sub-sampling the whole input to the HBE algorithm, we sub-sample the content of each hash table *independently* for each hash function $h_j, i = 1 \dots L$. Specifically, for each hash function h_j, we include a point $x \in X$ in the jth hash table with probability $1/(s\sqrt{\tau})$. This reduces the expected number of stored hashes to $O(\sqrt{\tau}L)$. If we start from a sample of size $s = \Theta(1/\tau \cdot 1/\epsilon^2)$, then $\sqrt{\tau}L = O(s)$, yielding the desired space bound; at the same time, each point is included in at least one hash table with constant probability, which means that at least $\Omega(1/\tau \cdot 1/\epsilon^2)$ points will be included in the union of the hash tables with high probability. Perhaps surprisingly, we show that this increases the variance of the overall estimator by only a constant factor.

For an intuition of why subsampling by a factor $\sqrt{\tau}$ does not distort the kernel values by much, consider a simple setting where ϵ is a constant, $n = 1/\tau$, and there is only one data point x that is very close to the query y (contributing ≈ 1) while all other points are far from y (contributing ≈ 0). In this case, the original HBE algorithm would collect the point x from every bin $h_1(y) \dots h_L(y)$, where $L = \sqrt{1/\tau}$. In contrast, if we subsample by a factor $\sqrt{\tau}$, then x is expected to survive in one table, and thus our algorithm is still likely to identify one such bin in expectation. Conditioned on this event, the estimate of the algorithm is approximately correct. See more details in Section 3.

1.1 Related work

There is a vast amount of work on fast kernel density estimation in low dimensions, including the seminal Fast Gauss Transform [GS91] and other tree-based methods [GM01, GB17]. However, as mentioned above, they entail an exponential dependence on the input dimension. The tree-based ASKIT algorithm [MXB15] avoids this dependence and is suitable for the high-dimensional regime. However, it lacks rigorous guarantees on the approximation quality. The empirical evaluation in [SRB+19] showed that HBE is consistently competitive with ASKIT, and in some settings outperforms it by an order of magnitude.

Another important line of research has focused on *sparsifying* (reducing the size) of the input pointset while preserving kernel density function values. This can be accomplished by constructing *core-sets* [Phi13, ZJPL13, PT18] or related approaches [CWS12, SRB+19]. Although effective in low dimensions, in high dimensions such approaches require $\Omega(1/\epsilon^2)$ samples (for an additive error of $\epsilon > 0$ [PT18]), which is the same as the simple random sampling approach.[5] We note that the sparsification approach can be combined with our improvement, as we can run our algorithm on a core-set instead of the original data set, and retain the core-set size while speeding up the query time.

In addition to the aforementioned works of [CS17, SRB+19], LSH-based estimators have been applied in [CXS18, LS18b, WCN18, LS18a] to a variety of machine learning tasks.

2 Preliminaries

Kernel Density Estimation. Consider a kernel map $k : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$. The kernel density estimation problem can be formally stated as follows.

[5] However, core-sets preserve all KDE values with high probability, while simple random sampling only preserves the KDE of any individual query with high probability.

Definition 2. *Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ be an input dataset, and $\epsilon, \tau \in (0,1)$ input parameters. Our goal is to construct a data structure such that for every query point $y \in \mathbb{R}^d$ that satisfies $KDE_X(y) \geq \tau$, we can return an estimate $\widetilde{KDE}(y)$ such that with constant probability,*

$$(1-\epsilon)KDE_X(y) \leq \widetilde{KDE}(y) \leq (1+\epsilon)KDE_X(y).$$

An exact computation of $KDE_X(y)$ performs n kernel evaluations. By standard concentration inequalities, the above approximation can be achieved by evaluating the kernel y with only $O(\frac{1}{\tau}\frac{1}{\epsilon^2})$ points chosen uniformly at random from X, and returning the average. As a result, we can assume without loss of generality (and up to scaling ϵ by a constant) that $n = O(\frac{1}{\tau}\frac{1}{\epsilon^2})$.

LSHable kernels. Locality-Sensitive Hashing (LSH) is a widely used framework for hashing metric datasets in a way that relates the collision probability of each pair of points to their geometric similarity. Kernel maps for which such hashing families exist are called "LSHable" [CK15]. The precise variant we will need is defined as follows.

Definition 3. *The kernel k is called $(\frac{1}{2}, M)$-*LSHable *if there exists a family H of hash functions $h : \mathbb{R}^d \to \{0,1\}^*$, such that for every $x, y \in \mathbb{R}^d$, Equation (1) holds.*[6]

Laplacian and Exponential kernels. The Laplacian kernel is $k(x,y) = e^{-\|x-y\|_1/\sigma}$, where $\sigma > 0$ is the bandwidth parameter. The exponential kernel is defined similarly as $k(x,y) = e^{-\|x-y\|_2/\sigma}$ (the difference is in use of the ℓ_2-norm instead of the ℓ_1-norm). For our purposes the two are essentially equivalent, as they give the same analytic and empirical results. We will mostly focus on the Laplacian kernel, since as we will see, it is $(\frac{1}{2}, 1)$-LSHable. As a corollary, a random rotation of the dataset [DIIM04, CS17] can be used to show that the Exponential kernel is $(\frac{1}{2}, O(1))$-LSHable.

3 The Data Structure

We begin by recalling the HBE-based KDE data structure of [CS17]. For simplicity consider the case $M = 1$. During preprocessing, they sample $L = O(\frac{1}{\sqrt{\tau}\epsilon^2})$ hash functions $h_1, \ldots, h_L$ from the LSH family H, and store $h_j(x_i)$ for every $i = 1, \ldots, n$ and $j = 1, \ldots, L$. The preprocessing time is $O(T_H \cdot \frac{n}{\sqrt{\tau}\epsilon^2})$, and the space usage (in addition to the dataset) is $O(S_H \cdot \frac{n}{\sqrt{\tau}\epsilon^2})$, where T_H is the time needed to evaluate the hash value of a point, and S_H is the space needed to store it. Recalling we have assumed that $n = O(\frac{1}{\tau}\frac{1}{\epsilon^2})$, these become $O(T_H \cdot \frac{1}{\tau^{1.5}\epsilon^4})$ and $O(S_H \cdot \frac{1}{\tau^{1.5}\epsilon^4})$ respectively.

Given a query point y, let $b_j(y) := \{x_i : h_j(x_i) = h_j(y)\}$ be the set ("bin") of data points whose h_j-hash is the same as that of y. The estimator picks a uniformly random data point x from $b_j(y)$ and computes $Z_j = \frac{1}{n}|b_j(y)| \cdot \sqrt{k(x,y)}$. If $b_j(y)$ is empty, then $Z_j = 0$. The final KDE estimate is $\widetilde{KDE}(y) = \frac{1}{L}\sum_{j=1}^{L} Z_j$. The query time is $O((T_H + T_k)/(\sqrt{\tau}\epsilon^2))$, where T_k is the time it takes to evaluate k on a single pair.

Our data structure is similar, except that for every h_j, we store the hash of every data point only with probability $\delta = 1/(n\sqrt{\tau})$. Therefore, on average we only compute and store a constant number of hashes of each data point, yielding expected preprocessing time of $O(T_H/(\tau \cdot \epsilon^2))$ and space usage of $O(S_H/(\tau \cdot \epsilon^2))$. The exact algorithm is given in Algorithm 1. Theorem 1, whose proof appears in the appendix, shows this still returns a sufficiently good estimate of $KDE_X(y)$.

Example. Let us give an illustration of the different approaches on the setting mentioned in the introduction. Suppose $\epsilon = \Theta(1)$ and $n \approx 1/\tau$. Consider a setting in which the query point is very close to a unique data point and very far from the rest of the data points. Concretely, $k(x_1, y) \approx 1$, while $k(x_i, y) \approx 0$ for every $i > 1$. The KDE value is $KDE_X(y) \approx \tau$. Naïve random sampling would have to sample $\Omega(1/\tau)$ points in order to pick up x_1 and return a correct estimate.

[6] The HBE framework of [CS17] accommodates (β, M)-LSHable kernels, that satisfy $M^{-1} \cdot k(x,y)^\beta \leq \Pr_{h\sim H}[h(x) = h(y)] \leq M \cdot k(x,y)^\beta$, where β can take any value in $[\frac{1}{2}, 1)$, and lower β is better. Since the kernels we consider attain the optimal setting $\beta = \frac{1}{2}$, we fix this value throughout.

[7] This can be implemented in expected time $O(L)$ by sampling $\tilde{L} \sim \text{Binomial}(n, \frac{L}{n})$, and then sampling a uniformly random subset of size $\tilde{L}$.

Algorithm 1 : Space-Efficient HBE

Proprocessing:

Input: Dataset $X \subset \mathbb{R}^d$ of n points; kernel $k(\cdot,\cdot)$; LSH family H; integer $1 \le L \le n$.
For $j = 1, \ldots, L$:
  Sample a random hash function h_j from H.
  Let $X_j \subset X$ be a random subset that includes each point with independent probability $\frac{L}{n}$.[7]
  For every $x \in X_j$, evaluate and store $h_j(x)$.

Query:

Input: Query point $y \in \mathbb{R}^d$.
For $j = 1, \ldots, L$:
  Sample a uniformly random point $x^{(j)}$ from $b_j(y) = \{x \in X'_j : h_j(x) = h_j(y)\}$.
  Let $Z_j \leftarrow \frac{k(x^{(j)},y)\cdot|b_j(y)|}{L\cdot \Pr_{h\sim H}[h(x^{(j)})=h(y)]}$.
Return $\frac{1}{L}\sum_{j=1}^{L} Z_j$.

In the HBE algorithm of [CS17], essentially in all hash tables x_1 would be the unique data point in the same bin as y, leading to a correct estimate $\frac{1}{L}\sum_{j=1}^{L}\frac{1}{n}|b_j(y)|\sqrt{k(x_1,y)} \approx \tau$. However, note that all terms in the sum are equal (to τ), which seems to be somewhat wasteful. Indeed, it would suffice to pick up x_1 in just one hash table instead of all of them.

In our method, x_1 would be stored in $\delta L \approx 1$ hash tables in expectation, say only in h_1, and in that table it would be the unique data point in $b_1(y)$. In the other tables ($j > 1$) $b_j(y)$ would be empty, which means the estimator evaluates to zero. The resulting KDE estimate is $\frac{1}{L}\left(\frac{1}{n\delta}|b_j(y)|\sqrt{k(x_1,y)} + \sum_{j=2}^{L} 0\right) \approx \tau$, which is still correct, while we have stored a hash of x_1 just once instead of L times.

3.1 LSH for the Laplacian Kernel

The Laplacian kernel $k(x,y) = e^{-\|x-y\|_1/\sigma}$ is a popular kernel, which fits naturally into the above framework since it is $(\frac{1}{2}, 1)$-LSHable. For simplicity, let us assume w.l.o.g. that in the dataset we need to hash, all point coordinates are in $[0,1]$. This does not limit the generality since the Laplacian kernel is shift-invariant, and the coordinates can be scaled by inversely scaling σ.

The LSHablility of the Laplacian kernel follows from the Random Binning Features construction of Rahimi and Recht [RR07] (see details in the appendix). The expected hash size is $O(d\log(1/\sigma))$, and the hash evaluation time is $O(d)$. We also give a variant (described below) with better hash size and evaluation time for $\sigma \ge 1$. Together, the following lemma holds.

Lemma 4. *There is an LSH family H_σ such that for every $x, y \in \mathbb{R}^d$, $\Pr_{h\sim H_\sigma}[h(x) = h(y)] = e^{-\|x-y\|_1/(2\sigma)}$. The expected hash size is $S_{H_\sigma} = O(\min\{d\log(1/\sigma), d/\sigma\})$ bits. The expected hashing time is $T_{H_\sigma} = O(\min\{d, d/\sigma\})$.*

The hashing family for the case $\sigma \ge 1$ is given as follows. Sample $\rho \sim \text{Poisson}(d/(2\sigma))$. Then sample $\xi_1, \ldots, \xi_\rho \in \{1, \ldots, d\}$ independently and uniformly at random, and $\zeta_1, \ldots, \zeta_\rho \in [0,1]$ independently and uniformly at random. These random choices determine the hash function h. Next we describe h. Given a point x to hash, for every $i = 1, \ldots, \rho$ set $b_i = 1$ if $x_{\xi_i} > \zeta_{\xi_i}$ and $b_i = 0$ otherwise. The hash $h(x)$ is the concatenation of $b_1, \ldots, b_\rho$. It is not hard to verify (see appendix) that $\Pr_h[h(x) = h(y)] = e^{-\|x-y\|_1/(2\sigma)}$.

Using the LSH family from Lemma 4 in Theorem 1 yields the following concrete data structure.

Corollary 5 (Data structure for Laplacian KDE)**.** *For the Laplacian kernel, there is a data structure for the KDE problem with expected space overhead $O(\min\{d\log(1/\sigma), d/\sigma\}/(\tau\epsilon^2))$, expected preprocessing time $O(\min\{d, d/\sigma\}/(\tau\epsilon^2))$, and query time $O(d/(\sqrt{\tau}\epsilon^2))$.*

Table 2: Properties of the datasets used in our experiments.

Name	Description	Number of points	Dimension
Covertype	forest cover type	$581,012$	55
Census	U.S. census	$2,458,285$	68
GloVe	word embeddings	$1,183,514$	100
MNIST	hand-written digits	$60,000$	784

4 Empirical Evaluation

We empirically evaluate our data structure for the Laplacian kernel.[8] For brevity, we will refer to the random sampling method as RS. The experimental results presented in this section are for the the Laplacian kernel $k(x, y) = e^{-\|x-y\|_1/\sigma}$. The results for the Exponential kernel are qualitatively similar are included in the appendix.

Choice of datasets. While the worst-case analysis shows that the HBE approach has asymptotically better query time than RS, it is important to note that RS can still attain superior performance in some practical settings. Indeed, the recent paper [SRB$^+$19] found this to be the case on various standard benchmark datasets, such as GloVe word embeddings [PSM14]. To reflect this in our experiments, we choose two datasets on which [SRB$^+$19] found HBE to be superior to RS as well as to state-of-the-art methods, and two datasets on which RS was found to be superior. The former two are Covertype [BD99] and Census[9], and the latter two are GloVe [PSM14] and MNIST [LC98]. Their properties are summarized in Table 2.

Experimental setting. We implement and evaluate Algorithm 1. Note that it is parameterized by the number of hash tables L, while its analysis in Theorem 1 is parameterized in terms of τ, ϵ, where we recall that $L = \Theta(1/(\sqrt{\tau}\epsilon^2))$. For practical implementation, parameterizing by L is more natural since it acts as a smooth handle on the resources to accuracy tradeoff – larger L yields better KDE estimates at the expense of using more time and space. τ, ϵ need not be specified explicitly; instead, for any τ, ϵ that satisfy $L = \Omega(1/(\sqrt{\tau}\epsilon^2))$, the guarantee of Theorem 1 holds (namely, for every query whose true KDE is at least τ, the KDE estimate has up to ϵ relative error with high probability).

We compare our method to the HBE method of [CS17], as well as to RS as a baseline. The plots for HBE and our method are generated by varying the number of hash functions L. The plots for RS are generated by varying the sample size. Note that neither method has any additional parameters to set. For each method and each parameter setting, we report the median result of 3 trials. For each dataset we choose two bandwidth settings, one which yields median KDE values of order 10^{-2}, and the other of order 10^{-3}.[10] The bandwidth values and their precise method of choice are specified in the appendix. The appendix also includes accuracy results for varying bandwidth values (Fig. 9).

Evaluation metrics. We evaluate the query time, space usage and preprocessing time. In all of the plots, the y-axis measures the average relative error (which directly corresponds to ϵ) of the KDE estimate, over 100 query points randomly chosen from the dataset. In the query time plots, the x-axis counts the number of kernel evaluations per query, which dominates and serves as a proxy for the running time. In the space usage plots, the x-axis counts the number of stored hashes. We use this measure for the space usage rather than actual size in bits, since there are various efficient ways to store each hash, and they apply equally to all algorithms. We also note that the plots do not account for the space needed to store the sampled dataset itself, which is the same for all methods. RS is not displayed on these plots since it has no additional space usage. In all three methods the preprocessing time is proportional to the additional space usage, and is qualitatively captured by the same plots.

Results. The query time plots consistently show that the query time to approximation quality tradeoff of our method is essentially the same as [CS17], on all datasets. At the same time, the space usage plots show that we have achieve a significantly smaller space overhead, with the gap from [CS17] substantially increasing as the target relative error becomes smaller. These findings affirm the direct advantage of our method as specified in Table 1.

[8] Our code is available at `https://github.com/talwagner/efficient_kde`.

[9] Available at `https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)`.

[10] In all the considered settings, the average KDE value is within a factor of at most 2 from the median KDE.

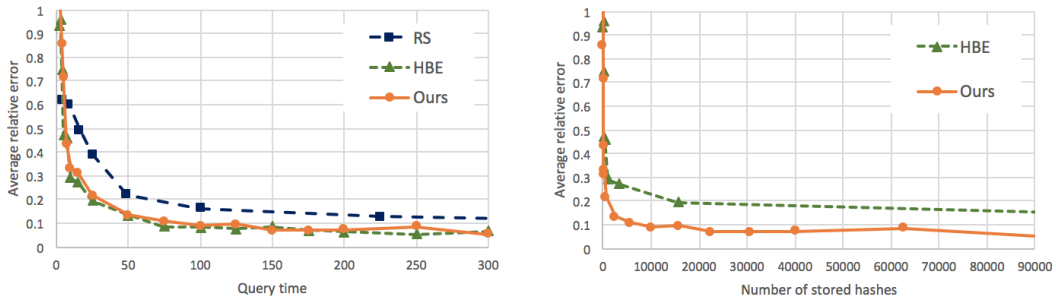

Figure 1: Covertype dataset, typical KDE values of order 10^{-2}.

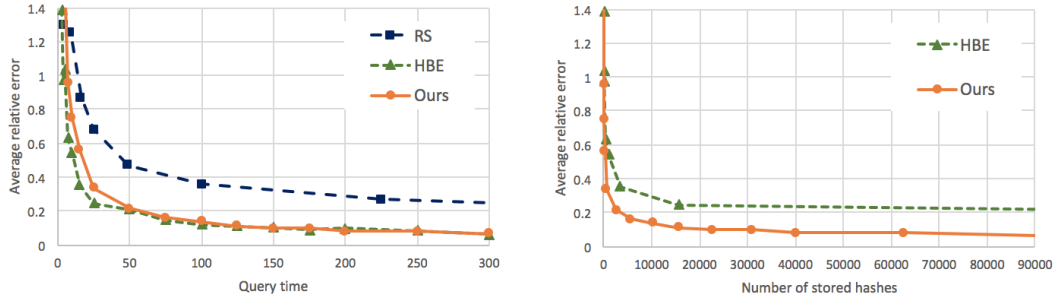

Figure 2: Covertype dataset, typical KDE values of order 10^{-3}.

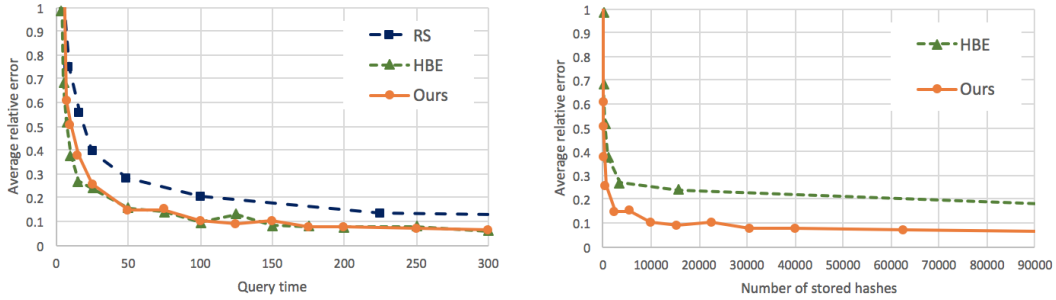

Figure 3: Census dataset, typical KDE values of order 10^{-2}.

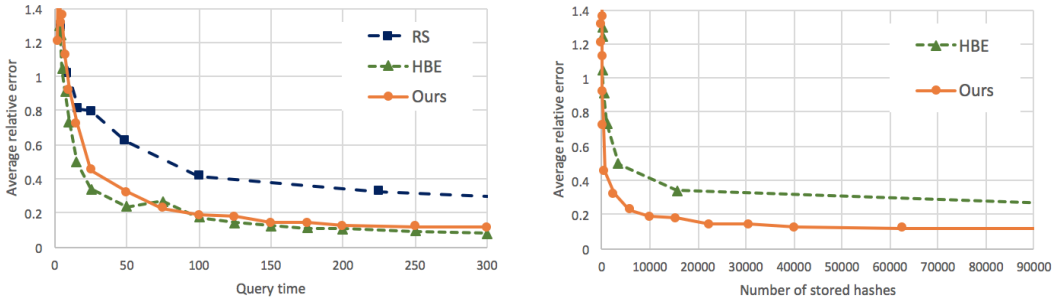

Figure 4: Census dataset, typical KDE values of order 10^{-3}.

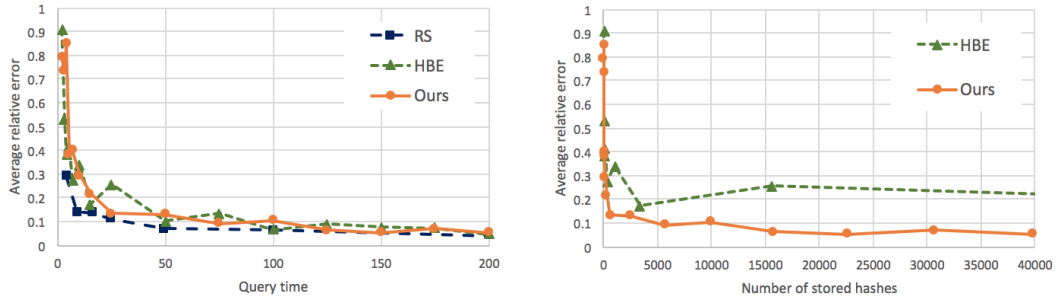

Figure 5: MNIST dataset, typical KDE values of order 10^{-2}.

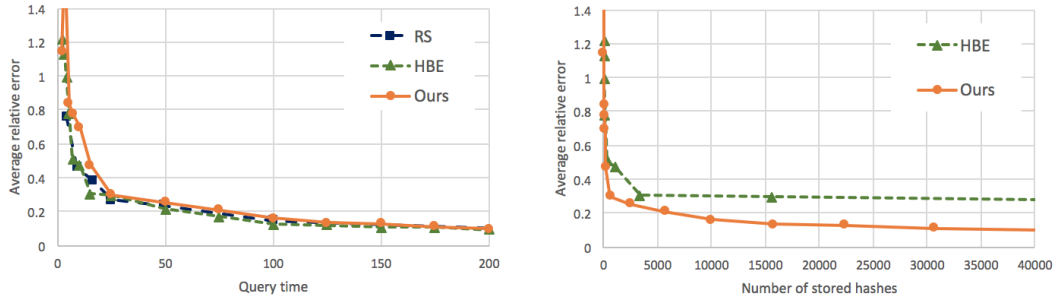

Figure 6: MNIST dataset, typical KDE values of order 10^{-3}.

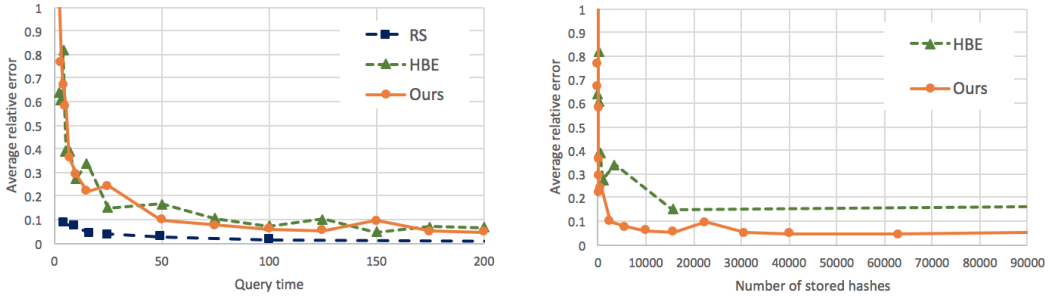

Figure 7: GloVe dataset, typical KDE values of order 10^{-2}.

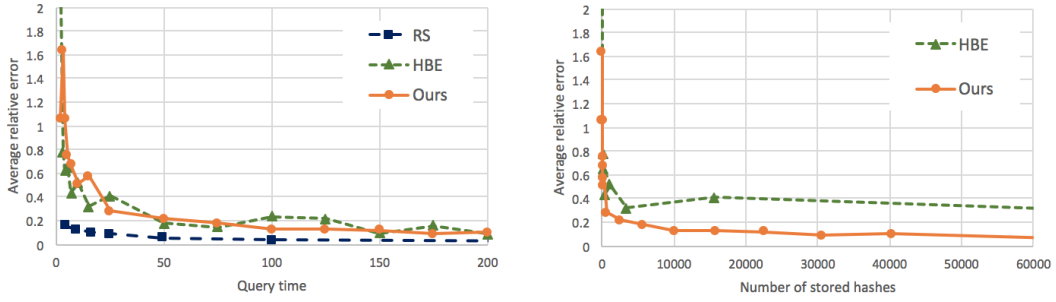

Figure 8: GloVe dataset, typical KDE values of order 10^{-3}.

Acknowledgments

We thank the anonymous reviewers for useful suggestions. Piotr Indyk was supported by NSF TRIPODS award #1740751 and Simons Investigator Award.

References

[AI06] Alexandr Andoni and Piotr Indyk, *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions*, Foundations of Computer Science (FOCS), IEEE, 2006, pp. 459–468.

[BD99] Jock A Blackard and Denis J Dean, *Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables*, Computers and electronics in agriculture **24** (1999), no. 3, 131–151.

[CK15] Flavio Chierichetti and Ravi Kumar, *Lsh-preserving functions and their applications*, Journal of the ACM (JACM) **62** (2015), no. 5, 33.

[CS17] Moses Charikar and Paris Siminelakis, *Hashing-based-estimators for kernel density in high dimensions*, Foundations of Computer Science (FOCS), 2017.

[CWS12] Yutian Chen, Max Welling, and Alex Smola, *Super-samples from kernel herding*, 2012.

[CXS18] Beidi Chen, Yingchen Xu, and Anshumali Shrivastava, *Lsh-sampling breaks the computational chicken-and-egg loop in adaptive stochastic gradient estimation*.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni, *Locality-sensitive hashing scheme based on p-stable distributions*, Proceedings of the twentieth annual symposium on Computational geometry, ACM, 2004, pp. 253–262.

[GB17] Edward Gan and Peter Bailis, *Scalable kernel density classification via threshold-based pruning*, Proceedings of the 2017 ACM International Conference on Management of Data, ACM, 2017, pp. 945–959.

[GM01] Alexander G Gray and Andrew W Moore, *N-body'problems in statistical learning*, Advances in neural information processing systems, 2001, pp. 521–527.

[GS91] Leslie Greengard and John Strain, *The fast gauss transform*, SIAM Journal on Scientific and Statistical Computing **12** (1991), no. 1, 79–94.

[IM98] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM, 1998, pp. 604–613.

[JDH99] Tommi S Jaakkola, Mark Diekhans, and David Haussler, *Using the fisher kernel method to detect remote protein homologies.*, ISMB, vol. 99, 1999, pp. 149–158.

[LC98] Yann LeCun and Corinna Cortes, *The mnist database of handwritten digits*, 1998.

[LS18a] Chen Luo and Anshumali Shrivastava, *Arrays of (locality-sensitive) count estimators (ace): Anomaly detection on the edge*, Proceedings of the 2018 World Wide Web Conference on World Wide Web, 2018, pp. 1439–1448.

[LS18b] ———, *Scaling-up split-merge mcmc with locality sensitive sampling (lss)*, Preprint. Available at (2018).

[MXB15] William B March, Bo Xiao, and George Biros, *Askit: Approximate skeletonization kernel-independent treecode in high dimensions*, SIAM Journal on Scientific Computing **37** (2015), no. 2, A1089–A1110.

[Phi13] Jeff M Phillips, *ε-samples for kernels*, Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms, SIAM, 2013, pp. 1622–1632.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning, *Glove: Global vectors for word representation*, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.

[PT18] Jeff M Phillips and Wai Ming Tai, *Near-optimal coresets of kernel density estimates*, arXiv preprint arXiv:1802.01751 (2018).

[RR07] Ali Rahimi and Benjamin Recht, *Random features for large-scale kernel machines*, Advances in neural information processing systems, 2007, pp. 1177–1184.

[SRB+19] Paris Siminelakis, Kexin Rong, Peter Bailis, Moses Charikar, and Philip Levis, *Rehashing kernel evaluation in high dimensions*, International Conference on Machine Learning, 2019.

[WCN18] Xian Wu, Moses Charikar, and Vishnu Natchu, *Local density estimation in high dimensions*, arXiv preprint arXiv:1809.07471 (2018).

[ZJPL13] Yan Zheng, Jeffrey Jestes, Jeff M Phillips, and Feifei Li, *Quality and efficiency for kernel density estimates in large data*, Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, 2013, pp. 433–444.

A Proof of Theorem 1

Theorem 1 is a strengthening of the main results of [CS17]. Let us first describe their analysis.

Fix a dataset $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$, a query point $y \in \mathbb{R}^d$, and a kernel map $k : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$. For every $i = 1, \dots, n$, let $w_i = k(x_i, y)$.

Let H be a family of hash functions from $\mathbb{R}^d$ to an arbitrary range U. For every x_i, denote its collision probability with y by $p_i = \Pr_{h \sim H}[h(x_i) = h(y)]$. Let $b_h(y) = \{i : h(x_i) = h(y)\}$ be the set of points with the same hash as y. Suppose we have $M \geq 1$ and $\beta \in [\frac{1}{2}, 1)$ such that for every i, $M^{-1} \cdot w_i^\beta \leq p_i \leq M \cdot w_i^\beta$. (It is instructive to think of the case $M = 1$ and $\beta = \frac{1}{2}$.)

The KDE estimator of [CS17] is $Z = \frac{w_i \cdot |b_h(y)|}{n \cdot p_i}$, where i is chosen uniformly at random from $b_h(y)$. If $b_h(y)$ is empty, then $Z = 0$.

Theorem 6 ([CS17])**.** $\mathbb{E}[Z] = \mathrm{KDE}_X(y)$*, and* $\mathrm{Var}[Z] \leq M^3 \cdot \mathrm{KDE}_X(y)^{2-\beta}$.

Since the dependence of $\mathrm{Var}[Z]$ on $\mathbb{E}[Z]$ is strictly better than quadratic, one can use this estimator to estimate $\mathrm{KDE}_X(y)$ with a smaller number of samples than naïve random sampling. In particular, if $\tau > 0$ is a lower bound $\mathrm{KDE}_X(y)$, then in order to get a multiplicative $(1 \pm \epsilon)$-approximation, the sufficient number of samples is $\mathrm{Var}[Z]/(\epsilon\,\mathbb{E}[Z])^2 = O(M^3/(\tau^\beta \epsilon^2))$. If the kernel admits an LSH family with good parameters β and M, then this is better than random sampling, which would require $O(1/(\tau\epsilon^2)$ samples. This is the driving force behind the HBE method of [CS17].

To obtain Theorem 1, we hash each point only with probability $\delta = 1/(n\tau^{1-\beta})$, where $\tau \leq \mathrm{KDE}_X(y)$. Formally, let $r_1, \dots, r_n$ be i.i.d. Bernoulli random variables with $\Pr[r_i] = \delta$. Let

$$b'_h(y) = \{i : h(x_i) = h(y) \text{ and } r_i = 1\}$$

be sparsified counterpart of $b_h(y)$. Our modified KDE estimator is $Z' = \frac{w_i \cdot |b'_h(y)|}{n \cdot \delta \cdot p_i}$, where i is chosen uniformly at random from $b'_h(y)$. If $b'_h(y)$ is empty, then $Z' = 0$. We prove the following.

Theorem 7. $\mathbb{E}[Z'] = \mathrm{KDE}_X(y)$*, and* $\mathrm{Var}[Z'] \leq (M^3 + M) \cdot \mathrm{KDE}_X(y)^{2-\beta}$.

Proof. Our proof closely follows that of Theorem 6. Starting with the expectation,

$$\begin{aligned}
\mathbb{E}[Z'] &= \frac{1}{n\delta}\,\mathbb{E}\,\frac{w_i}{p_i/|b'_h(y)|} \\
&= \frac{1}{n\delta} \mathop{\mathbb{E}}_{\substack{r_1,\dots,r_n \\ h \sim H}} \mathop{\mathbb{E}}_{i \in b'_h(y)} \left[|b'_h(y)| \frac{w_i}{p_i} \right] \\
&= \frac{1}{n\delta} \mathop{\mathbb{E}}_{\substack{r_1,\dots,r_n \\ h \sim H}} \sum_{i \in b'_h(y)} \frac{w_i}{p_i} \\
&= \frac{1}{n\delta} \sum_{i=1}^{n} \frac{w_i}{p_i} \Pr_{\substack{r_1,\dots,r_n \\ h \sim H}} [i \in b'_h(y)] \\
&= \frac{1}{n\delta} \sum_{i=1}^{n} \frac{w_i}{p_i} \Pr_{\substack{r_1,\dots,r_n \\ h \sim H}} [i \in b_h(y)\ \&\ r_i = 1] \\
&= \frac{1}{n} \sum_{i=1} w_i \\
&= \mathrm{KDE}_X(y).
\end{aligned}$$

Next we bound the variance:

$$\begin{aligned}
\mathrm{Var}[Z'] &\le \mathbb{E}[(Z')^2] \\
&= \frac{1}{n^2\delta^2}\,\mathbb{E}\left[\frac{w_i^2}{p_i^2/|b'_h(y)|^2}\right] \\
&= \frac{1}{n^2\delta^2}\,\mathop{\mathbb{E}}_{\substack{r_1,\ldots,r_n\\ h\sim H}}\,\mathop{\mathbb{E}}_{i\in b'_h(y)}\left[|b'_h(y)|^2\frac{w_i^2}{p_i^2}\right] \\
&= \frac{1}{n^2\delta^2}\,\mathop{\mathbb{E}}_{\substack{r_1,\ldots,r_n\\ h\sim H}}\left[|b'_h(y)|\sum_{i\in b'_h(y)}\frac{w_i^2}{p_i^2}\right] \\
&= \frac{1}{n^2\delta^2}\,\mathop{\mathbb{E}}_{\substack{r_1,\ldots,r_n\\ h\sim H}}\left[\sum_j [j\in b'_h(y)]\sum_i [i\in b'_h(y)]\frac{w_i^2}{p_i^2}\right] \\
&= \frac{1}{n^2\delta^2}\sum_i \frac{w_i^2}{p_i^2}\sum_j \mathop{\mathbb{E}}_{\substack{r_1,\ldots,r_n\\ h\sim H}}[[j\in b'_h(y)][i\in b'_h(y)]] \\
&= \frac{1}{n^2\delta^2}\sum_i \frac{w_i^2}{p_i^2}\sum_j \mathop{\Pr}_{\substack{r_1,\ldots,r_n\\ h\sim H}}[j\in b'_h(y)\ \&\ i\in b'_h(y)].
\end{aligned}$$

We split the last term into two expressions:

$$\frac{1}{n^2\delta^2}\sum_i \frac{w_i^2}{p_i^2}\sum_{j:j\neq i}\mathop{\Pr}_{\substack{r_1,\ldots,r_n\\ h\sim H}}[j\in b'_h(y)\ \&\ i\in b'_h(y)], \tag{2}$$

and

$$\frac{1}{n^2\delta^2}\sum_i \frac{w_i^2}{p_i^2}\mathop{\mathbb{E}}_{\substack{r_1,\ldots,r_n\\ h\sim H}}[i\in b'_h(y)]. \tag{3}$$

To upper bound Eq. (2), we observe that, since $j\neq i$,

$$\mathop{\Pr}_{\substack{r_1,\ldots,r_n\\ h\sim H}}[j\in b'_h(y)\ \&\ i\in b'_h(y)] = \delta^2\mathop{\Pr}_{h\sim H}[j\in b_h(y)\ \&\ i\in b_h(y)] \le \delta^2 p_j.$$

Therefore, Eq. (2) is upper bounded by $\frac{1}{n^2}\sum_i \frac{w_i^2}{p_i^2}\sum_j p_j$. This expression is bounded in the proof of Theorem 6 in [CS17], and we now reproduce the argument for completeness. We observe that $\Pr[j\in b_h(y)\ \&\ i\in b_h(y)] \le \Pr[j\in b_h(y)] = p_j$ and, using the bounds $\frac{w_i^\beta}{M}\le p_i \le M w_i^\beta$, conclude that

$$\frac{1}{n^2}\sum_i \frac{w_i^2}{p_i^2}\sum_j p_j \le \frac{M^3}{n^2}\sum_i w_i^{2-2\beta}\sum_j w_j^\beta.$$

To prove $\mathrm{Var}[Z]\le M^3\,\mathrm{KDE}_X(y)^{2-\beta}$, it is sufficient to show

$$\frac{1}{n^2}\sum_i w_i^{2-2\beta}\sum_j w_j^\beta \le \left(\frac{1}{n}\sum_i w_i\right)^{2-\beta}.$$

This follows from the inequalities $\frac{1}{n}\sum_i w_i^{2-2\beta}\le\left(\frac{1}{n}\sum_i w_i\right)^{2-2\beta}$ and $\frac{1}{n}\sum_j w_j^\beta\le\left(\frac{1}{n}\sum_i w_i\right)^\beta$. The first inequality holds for any β that satisfies $0\le 2-2\beta\le 1$ and the second inequality holds for any $0\le\beta\le 1$. That is, both inequalities hold if $\frac{1}{2}\le\beta\le 1$.

To upper bound Eq. (3) we observe that $\mathbb{E}_{\substack{r_1,\ldots,r_n\\ h\sim H}}[i\in b'_h(y)] = p_i\delta$ and therefore Eq. (3) is upper bounded by $\frac{1}{n^2\delta}\sum_i\frac{w_i^2}{p_i}\le\frac{M}{n^2\delta}\sum_i w_i^{2-\beta}$. Since $\sum_i w_i^{2-\beta}\le\sum_i w_i = n\cdot\mathrm{KDE}_X(y)$ and $\delta = 1/(n\tau^{1-\beta})\ge 1/(n\cdot\mathrm{KDE}_X(y)^{1-\beta})$, this is upper bounded by $M\cdot\mathrm{KDE}_X(y)^{2-\beta}$, as needed. □

To derive Theorem 1, set $\beta = 1/2$. By the above theorem, the estimator Z' is unbiased and satisfies $\mathrm{Var}[Z'] = 2M^3/\tau^{1.5}$. Therefore, in order to obtain a $(1\pm\epsilon)$-approximation for $\mathrm{KDE}_X(y)$, it is sufficient to return the average over $L = O(M^3/(\sqrt{\tau}\epsilon^2))$ independent samples of the estimator.

Preprocessing time: To be able to draw samples from the estimator, we need to hash a subset of the n pointset $x_1, \ldots, x_n$. The expected size of the subset is $\delta n = 1/\sqrt{\tau}$. The time needed to hash a single point is T_H. We need to repeat this L times (once for each sample of the estimator we would draw int he query phase). The total preprocessing time complexity becomes

$$1/\sqrt{\tau} \cdot T_H \cdot L = O\left(\frac{1}{\tau} \cdot \frac{T_H M^3}{\varepsilon^2}\right).$$

Space usage: In order to draw a single sample from the estimator, we store the hash of each point x_i for which $r_i = 1$. We also need to fully store x_i itself, since if we draw it from $b'_h(y)$ during the query phase, we would need to evaluate $k(x_i, y)$ in order to compute Z'. The expected numbers of these points is $\delta n = 1/\sqrt{\tau}$, so for a single sample we store in expectation $(S_X + S_H)/\sqrt{\tau}$ bits, where S_X is the storage size of a data point, and S_H is the storage size of a hash value. Repeating this L times, the total storage size is

$$\frac{S_X + S_H}{\sqrt{\tau}} \cdot L = O\left(\frac{1}{\tau} \cdot \frac{(S_X + S_H)}{\varepsilon^2}\right) \text{ bits.}$$

Query time: To draw a sample from the estimator, we need to hash the query point y. This takes time T_H. Furthermore, given the hash value, we need to sample a random element from the corresponding bucket $b'_h(y)$ and evaluate the random variable Z'. This takes time T_k. Thus, we spend $T_H + T_k$ time to draw a single sample from the estimator. Since we do that L times, the total query time is

$$L \cdot (T_H + T_k) = O\left(\frac{1}{\sqrt{\tau}} \cdot \frac{(T_H + T_k)M^3}{\varepsilon^2}\right)$$

as promised.

B KDE Data Structure for the Gaussian Kernel

For the Gaussian kernel $k(x, y) = e^{-\|x-y\|^2/\sigma^2}$, the best LSHability result we are aware of is based on the ball-carving LSH of [AI06].

Theorem 8 ([AI06]; see also Theorem 11 in [CS17])**.** *For any $R > 0$ there exists a distribution H of hash functions such that for any $x, y \in \mathbb{R}^d$ with $\|x - y\|_2 \leq R$ the following bounds hold.*

$$e^{-\|x-y\|_2^2} \cdot e^{-O(R^{4/3}\log R)} \leq \Pr_{h\sim H}[h(x) = h(y)] \leq e^{-\|x-y\|_2^2} \cdot e^{O(R^{4/3}\log R)}.$$

The time complexity of computing a hash value $h(x)$ is $d \cdot e^{O(R^{4/3}\log R)}$. Finally, the probability $Pr_{h\sim H}[h(x) = h(y)]$ is non-increasing in the distance $\|x - y\|_2$.

It can be used to give the following time and space efficient data structure for Gaussian KDE.

Theorem 9. *Given n points $y_1, \ldots, y_n \in \mathbb{R}^d$ and parameters $1 \geq \tau \geq \frac{1}{n^2}$ and $1 \geq \varepsilon \geq \frac{1}{n^2}$, we can build a data structure in space $\frac{1}{\tau} \cdot \frac{n^{o(1)}}{\varepsilon^4}$ that efficiently answers $\mathrm{KDE}(x)$ queries for the Gaussian kernel $k(x, y) = e^{-\|x-y\|_2^2}$. In particular, given a query point $x \in \mathbb{R}^d$ with $\mathrm{KDE}(x) \geq \tau$, we can approximate $\mathrm{KDE}(x)$ within the multiplicative factor of $1 + \varepsilon$ in time $O(d) \cdot \frac{\log^3 n}{\varepsilon^2} + \frac{1}{\sqrt{\tau}} \cdot \frac{n^{o(1)}}{\varepsilon^4}$.*

Proof. The proof proceeds in two steps. First apply Theorem 1 to the above LSHability result. We use Theorem 8 and set $R = (\log n)^{2/3}$. We get that the hashing scheme H satisfies

$$e^{-\|x-y\|_2^2} \cdot n^{-o(1)} \leq \Pr_{h\sim H}[h(x) = h(y)] \leq e^{-\|x-y\|_2^2} \cdot n^{o(1)}$$

for all $x, y \in \mathbb{R}^d$ with $\|x - y\|_2 \leq R = (\log n)^{2/3}$. Furhermore, the hashing can be performed in $d \cdot n^{o(1)}$ time.

We can get rid of the assumption that $\|x - y_i\|_2 \leq R$ for all $i = 1, \ldots, n$ as follows. Observe that, if $\|x - y_i\|_2 > R$, then $\Pr_{h\sim H}[h(x) = h(y_i)] \leq e^{-R^2} n^{o(1)} \leq e^{-(\log n)^{4/3}} n^{o(1)} \leq n^{-\omega(1)}$. We used the fact that the probability $\Pr_{h\sim H}[h(x) = h(y_i)]$ is non-increasing in the distance $\|x - y_i\|_2$. This

implies that with probability $1 - n^{\omega(1)}$ we have that all i with $h(x) = h(y_i)$ satisfy $\|x - y_i\|_2 \leq R$. Since all our samples y_i satisfy $h(x) = h(y_i)$, we lose at most a neglibile factor in the probability of success.

We can get a KDE algorithm by setting $M = n^{o(1)}$, $T_H = dn^{o(1)}$, $S_X = S_H = O(d \log n)$ and $T_k = O(d)$.

In the second step, we improve the dependence on d by dimension reduction. In particular, we reduce the space by projecting the points from the d dimensional space to $\frac{O(\log^3 n)}{\varepsilon^2}$ dimensional space. The extra term $O(d) \cdot \frac{\log^3 n}{\varepsilon^2}$ in the time complexity comes from the time needed to perform the projection.

We randomly project the points y_i and the point x to $O(\log n)/\delta^2$ dimensional space for $\delta = \frac{\varepsilon}{2\ln(1/(\varepsilon\tau))}$. This preserved all distances $\|x - y_i\|$ within the multiplicative factor of $1 \pm \delta$. After the projection, the contribution from a point y_i to the KDE value becomes $\exp(-(1 \pm \delta)\|x - y_i\|_2^2)$. Consider the case $\|x - y_i\|_2^2 \geq 2\ln(1/(\varepsilon\tau))$. The contribution of such a point y_i after the projection is $\leq \exp(-\|x - y_i\|_2^2/2) \leq \varepsilon\tau$. Thus, the average contribution from such point to the KDE value after the projection is $\varepsilon\tau$, which can be subsumed by the $1+\varepsilon$ multiplicative approximation. Consider the case $\|x - y_i\|_2^2 < 2\ln(1/(\varepsilon\tau))$. In this case we observe that after the projection the contribution $\exp(-(1\pm\delta)\|x-y_i\|_2^2) = \exp(-\|q-p_i\|_2^2)\exp(\pm\delta\|x-y_i\|_2^2)$ differs from the original contribution by a multiplicative factor of at most $\exp(\delta\|x-y_i\|_2^2) \leq 1+O(\varepsilon)$ since $\delta\|x-y_i\|_2^2 \leq \varepsilon \leq 1$. Therefore, in this case too we introduce a multiplicative error of at most $1 + \varepsilon$.

This allows us to reduce the dimensionality of the pointset from d to

$$\frac{O(\log n)}{\delta^2} = \frac{O(\log n)\log(1/(\varepsilon\tau))^2}{\varepsilon^2} \leq \frac{O(\log^3 n)}{\varepsilon^2}$$

for the purpose of estimating $\mathrm{KDE}_X(y)$. □

C Laplacian Kernel LSH

In this section we fully describe the LSHability of the Laplacian kernel, as per Lemma 4. Recall that we assume w.l.o.g. that all point coordinates are in $[0, 1]$. For the sake of clarity, we will describe LSH families H such that $\Pr_{h\sim H}[h(x) = h(y)] = e^{-\|x-y\|_1/\sigma}$. The $(\frac{1}{2}, 1)$-LSHable property then follows simply by doubling the bandwidth σ.

For $\sigma < 1$ we use the Random Binning Features of Rahimi and Recht [RR07], which we now recall. Start with the one-dimensional case $d = 1$. Sample c from the Gamma distribution with shape 2 and scale σ. The probability density function of this distribution is $p(x) = \sigma^{-2} \cdot x \cdot e^{-x/\sigma}$. Then, impose on the real line a one-dimensional uniform grid of side length c, shifted by a uniformly random $s \sim [0, c)$. The random choices of c and s determine the hash function h. Given a point $x \in [0, 1]$, h maps it to the grid cell containing it.

One can verify that $\Pr_h[h(x) = h(y)] = e^{-|x-y|/\sigma}$ for every $x, y \in [0, 1]$ [RR07], and that the time to evaluate $h(x)$ is $O(1)$. Furthermore, the number of grid cells intersecting the interval $[0, 1]$ is $\Theta(1/c)$. Since $1/c$ has an inverse-Gamma distribution, its expected value is $1/\sigma$, hence there are $\Theta(1/\sigma)$ grid cells in expectation, and thus the expected space to store a hash value is $\log(1/\sigma)+O(1)$ bits. Finally, for an arbitrary dimension d, we simply perform the above for each dimension independently, and concatenate the resulting hashes. We then have $\Pr_h[h(x) = h(y)] = e^{-\|x-y\|_1/\sigma}$ with hash evaluation time $O(d)$ and expected hash size $O(d \log(1/\sigma))$.

For $\sigma \geq 1$, we use the LSH family described in Section 3.1. Start with the one-dimensional case $d = 1$. For a uniformly random $\zeta \in [0, 1]$, let $b(x) = 1$ if $x > \zeta$ and $b(x) = 0$ otherwise, and similarly $b(y) = 1$ if $y > \zeta$ and $b(y) = 0$ otherwise. Then we have $\Pr[b(x) = b(y)] = 1 - |x - y|$. In the arbitrary dimensional case $x, y \in [0, 1]^d$, applying this to a uniformly random dimension $\xi \in \{1, \ldots, d\}$ yields $\Pr[b(x) = b(y)] = \frac{1}{d}\sum_{\xi=1}^{d}(1 - |x_\xi - y_\xi|) = 1 - \frac{1}{d}\|x - y\|_1$. If we repeat this ρ independent times, where ρ is a fixed non-negative integer, and let $h(x)$ be the concatenation of the $b(x)$'s of the ρ repetitions (and similarly define $b(y)$), then $\Pr[h(x) = h(y)] = (1 - \frac{1}{d}\|x - y\|_1)^\rho$. Finally, choosing $\rho \sim \mathrm{Poisson}(d/\sigma)$ yields

$$\Pr[h(x) = h(y)] = \sum_{\rho=0}^{\infty} \frac{e^{-d/\sigma} \cdot (d/\sigma)^\rho}{\rho!} \cdot (1 - \tfrac{1}{d}\|x - y\|_1)^\rho = e^{-\|x-y\|_1/\sigma}.$$

Table 3: Bandwidth settings used in our experiments.

Dataset	Estimate of median NN distance (ϕ)	Bandwidth for KDE values $\sim 10^{-2}$	Bandwidth for KDE values $\sim 10^{-3}$
Covertype	0.005	$20 \cdot \phi$	$10 \cdot \phi$
Census	0.01	$5 \cdot \phi$	$3 \cdot \phi$
GloVe	4.48	$0.5 \cdot \phi$	$0.25 \cdot \phi$
MNIST	38.1	$1 \cdot \phi$	$0.5 \cdot \phi$

D Additional Exprimental Details

Bandwidth selection. The rule of thumb suggested in [JDH99] for bandwidth selection is to take the median distance of a query point to its nearest neighbor in the dataset. We estimate this parameter for each dataset and denote it by ϕ. Since its effect of the KDE values is inconsistent between the various datasets, we scale it by a constant so as to make the typical KDE values be within a certain order of magnitude. Specifically, we experiment with two orders of magnitude, 10^{-2} and 10^{-3}. (Note that larger typical values are essentially trivial to estimate by standard concentration inequalities, while for smaller values an approximation is largely uninformative). The specific numbers used are listed in Table 3. Note that the listed values of ϕ are estimated after shifting and scaling each dataset such that all point coordinates in are in $[0, 1]$.

Accuracy with varying bandwidth. Figure 9 displays the accuracy on the Covertype and Census datasets for varying bandwidth values. The results are with $L = 250$ (i.e., each KDE value is estimated using 250 kernel evaluations). It shows that the accuracy of our method is similar to HBE, and significantly better then RS (whose accuracy improves and converges to the hashing-based methods as the the bandwidth grows and the KDE values become bounded away from 0). At the same time, the space usage of our method is smaller than HBE by a factor of $L = 250$.

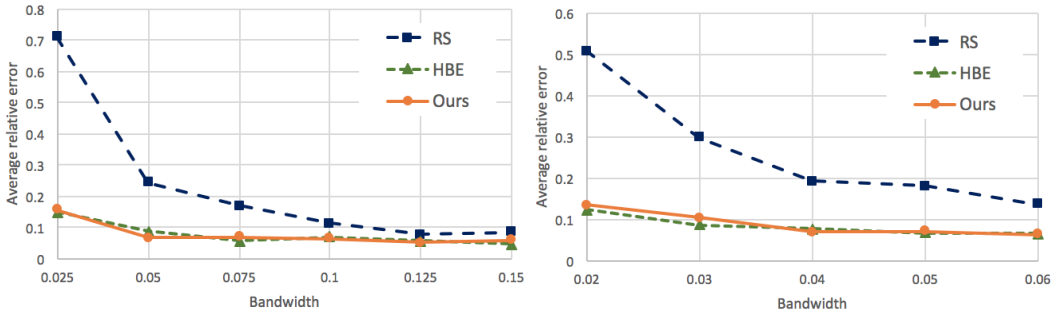

Figure 9: Accuracy with $L = 250$ and varying bandwidth, for the Covertype (left) and Census (right) datasets. The space usage (not displayed in the plot) of HBE is larger by a factor of 250 than ours.

Exponential kernel. Section 4 presented empirical results for the Laplacian kernel, and mentioned that similar results are achieved for the Exponential kernel. Some of these results are depicted in the figures below. The results for both hashing-based methods (HBE and ours) are obtained by a random rotation of the dataset[11] followed by the algorithm presented in the main text. Ground-truth KDE and RS are computed directly on the original ℓ_2-distances.

[11] It is known that the ℓ_2-distances after a random rotation are approximately equal, with high probability, to the ℓ_1-distances before the projection.

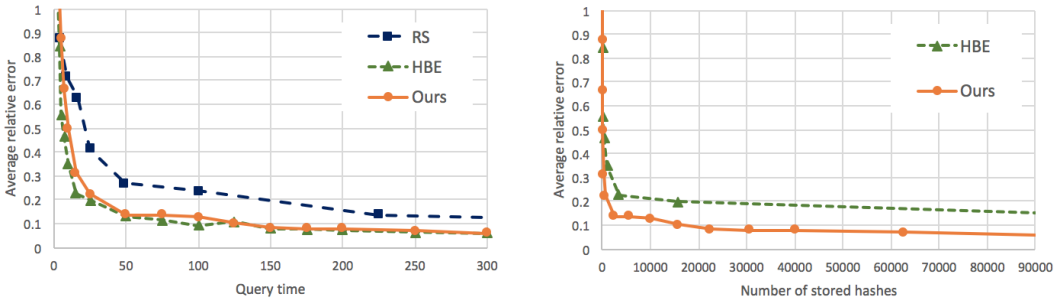

Figure 10: Covertype dataset, Exponential kernel, typical KDE values of order 10^{-2}.

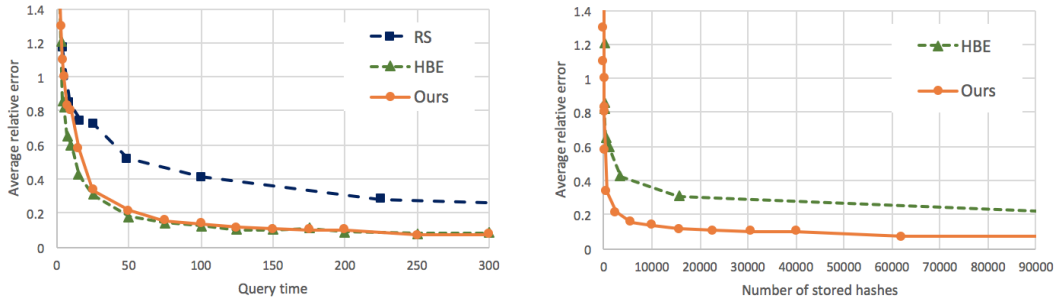

Figure 11: Covertype dataset, Exponential kernel, typical KDE values of order 10^{-3}.

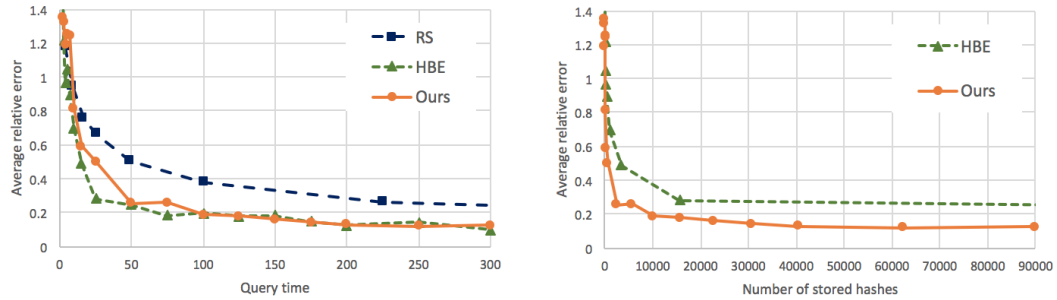

Figure 12: Census dataset, Exponential kernel, typical KDE values of order 10^{-3}.

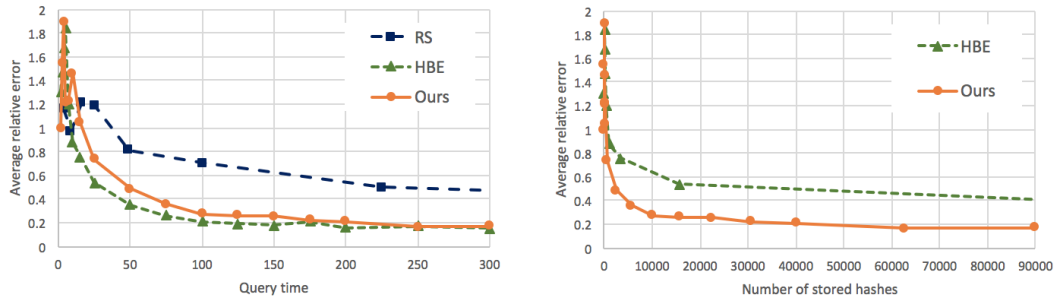

Figure 13: Census dataset, Exponential kernel, typical KDE values of order 10^{-4}.