Kernel-Based Cardinality Estimation on Metric Data

Michael Mattig Thomas Fober Christian Beilschmidt Bernhard Seeger
Department of Mathematics and Computer Science
University of Marburg, Germany
{mattig, thomas, beilschmidt, seeger}@mathematik.uni-marburg.de

ABSTRACT

The efficient management of metric data is extremely important in many challenging applications as they occur e.g. in the life sciences. Here, data typically cannot be represented in a vector space. Instead, a distance function only allows comparing individual elements with each other to support distance queries. As high-dimensional data suffers strongly from the curse of dimensionality, distance-based techniques also allow for better handling of such data. This has already led to the development of a plethora of metric indexing and processing techniques. So far, the important problem of cardinality estimation on metric data has not been addressed in the literature. Standard vector-based techniques like histograms require an expensive and error-prone embedding. Thus, random sampling seems to be the best choice for selectivity estimation so far, but errors are very high for moderately small queries. In this paper, we present a native cardinality estimation technique for distance queries on metric data based on kernel-density estimation. The basic idea is to apply kernels to the one-dimensional distance function among metric objects and to use novel global and local bandwidth optimization methods. Our results on real-world data sets show the clear advantage of our method in comparison to its competitors.

1 INTRODUCTION

Statistics about the distribution of data in a database are used for two very important aspects of data management: query optimization and data exploration. In query optimization, they allow estimating the costs of operations, choosing appropriate algorithms, and computing the order of joins. For very large databases, where computations take a very long time, small in-memory statistics can deliver approximate answers. Those are often sufficient to determine whether it is worth further investigating the data in a particular direction.

While one- and multidimensional vector data is very common in traditional applications, there are many domains for which data is in a metric space only. This means data is not describable by a d-dimensional vector, instead there exists only a metric measuring distances between pairs of objects. Examples include the life sciences, where e.g. proteins are usually described by their geometrical structure or at least a sequence of amino acids. Multimedia data comes in different datatypes such as JPEG or MPEG which are also not appropriate for a relational representation.

In such domains there is a severe lack of native statistical support. Thus, a standard approach is to transform metric data into a multidimensional vector space and to apply one of the standard estimation techniques [18]. There are two serious opposing effects. First, a metric embedding causes in general a considerable information loss. In order to alleviate this, the number of dimensions needs to be sufficiently high. Second, the well-known curse of dimensionality is already noticeable for a moderate number of dimensions. Thus, statistics provide only accurate results for low-dimensional vector spaces [1].

In this paper, we present the first native method for cardinality estimation of distance queries in metric spaces. The basic idea is to consider the distances of objects in a metric space and to use kernel techniques to estimate the underlying distance distribution. By tuning the bandwidth of the kernels and the kernel function, we obtain a robust estimator for the cardinality of distance queries in metric spaces. Moreover, our approach is also beneficial for high-dimensional vector spaces by treating them as metric spaces, thus considering only the distance among objects, to overcome the shortcomings of standard vectorial statistics.

The main contributions of this paper are:

- We show the deficiencies of traditional cardinality estimation techniques on metric data sets.
- We present the first effective and efficient method for cardinality estimation in metric space.
- Extensive experiments on real-world data show the validity of our approach.

The rest of the paper is structured as follows. Section 2 describes several applications for cardinality estimation in metric spaces, formally defines the problem, and emphasizes the differences of vector and metric data. Section 3 presents related work in the areas of cardinality estimation in general, techniques for embedding metric data into vector space and kernel-based techniques for cardinality estimation. Section 4 presents our distance-based kernel estimator approach in metric space. Section 5 describes our methods for global and local bandwidth optimization. Section 6 presents our experimental findings. Finally, Section 7 concludes the paper.

2 PRELIMINARIES

We first give a formal description of the problem of cardinality estimation on metric data. Then, we discuss several applications that greatly benefit from a suitable solution to this problem. Finally, we discuss the fundamental differences between vector and metric data that lead to the ineffectiveness of established methods.

2.1 Problem Specification

Let X be a set of N objects $\{x_1, \ldots, x_N\} \subseteq \mathcal{X}$. These objects are all of a certain type, in particular a type which can differ from $\mathbb{R}^n$. Moreover a distance function $dist_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is given which fulfills the three properties of a metric, namely

(a) identity of indiscernibles: $dist_{\mathcal{X}}(x, y) = 0 \Leftrightarrow x = y$,
(b) symmetry: $dist_{\mathcal{X}}(x, y) = dist_{\mathcal{X}}(y, x)$ and
(c) triangle inequality: $dist_{\mathcal{X}}(x, z) \leq dist_{\mathcal{X}}(x, y) + dist_{\mathcal{X}}(y, z)$,

with $x, y, z \in \mathcal{X}$. We will refer to the combination of X and $dist_{\mathcal{X}}$ as metric data. In mathematics the pair $(\mathcal{X}, dist_{\mathcal{X}})$ is called metric space.

Cardinality estimation for metric data can be formalized as follows: Given a distance query $Q = (x_Q, r_Q)$, with object $x_Q \in \mathcal{X}$

and distance $r_Q \in \mathbb{R}_+$, *efficiently* approximate the cardinality of the set $\{x \in X \mid dist_{\mathcal{X}}(x_Q, x) \leq r_Q\}$. We will refer to the distance r_Q as query radius. The true cardinality is denoted by $c(Q)$ and the estimated cardinality by $\tilde{c}(Q)$. The goal is to minimize the error of the estimation, but also the construction costs and the size, as well as the query time of the estimator. Note that the actual cardinality can, of course, be calculated by computing the distance to every other item in the data set. This requires a linear number of distance calculations. Each of them can be very costly as, e.g., in video dissimilarity. Hence, we want to minimize the computational costs induced by an estimator.

2.2 Applications

Cardinality estimation in metric spaces has many applications. Among others we want to mention the domain of Machine Learning and Data Mining, where algorithms are usually based on a distance measure. The most prominent example is the so-called k-nearest neighbor (kNN) classifier which can be used to classify any kind of data, if it is endowed with a distance measure.

The basic idea is to retrieve the k objects from a database which have the smallest distances to a certain query object. Assuming that the objects within the database carry a certain class label (e.g. customers of an insurance with label *churn* or *no churn*), the query is classified with the majority label from the set of the k nearest neighbors. kNN classifiers are known to be inefficient since for each run of such an algorithm a complete scan of the data set is required. To accelerate this algorithm typically metric index structures [28] are employed that allow the efficient retrieval of elements within a given distance of a query object. However, it is hard to specify the radius for the corresponding queries since the kNN classifier requires rather the set of k nearest neighbors than a certain set of neighbors exhibiting a certain maximal distance to the query object. For calculating the minimum radius, leading to a retrieval of k results, cardinality estimation can be used. For example, [14] make use of cardinality estimation to assign objects to different Locality Sensitive Hashing tables of different radii. This improves kNN queries on data sets where the distances of the k nearest neighbours of items vary greatly over the data set. A single LSH table could not sufficiently answer such queries.

Another example is the estimation of densities, e.g. for Bayes Classifiers, where the density around an element x is proportional to the amount of elements in a database that are in a small vicinity to x. This vicinity is typically specified by a certain small distance. Obviously, a reliable cardinality estimation approach would increase the efficiency of such classifiers enormously.

If a query is expressed as a conjunction of multiple proximity predicates, each of them with a different distance function, cardinality estimation is useful for computing an efficient order of their computation. In an optimal execution plan queries should be applied in an order that leads to quickly decreasing result sets. Cardinality estimation can be used to answer exactly this question and to find an order in which the different distance measures are to be applied. Applications in which such scenarios occur are, e.g., pharmaceutical chemistry, where different distance measures covering certain requirements are applied onto protein and/or ligand databases to get the final result in form of a very small set of therapeutically effective drugs. In general, this is a metric scenario, since proteins cannot be described on the structural level by vectors without a considerable loss of information.

2.3 Vector Data vs. Metric Data

In order to emphasize the fundamental differences of metric data to vector data that lead to the in-applicability of established methods, we now briefly review important properties of a vector space. We limit our discussion to vector spaces over the real numbers. Here, a d-dimensional vector space consists of elements $\boldsymbol{x} = (x_1, ..., x_d)$ with a real value $x_i \in \mathbb{R}$ called *coordinate* for each dimension. Individual elements can be added to each other and multiplied with scalar values $v \in \mathbb{R}$. This, e.g., allows to compute the mean of multiple elements which is not possible in a metric space. Thus one of the most basic data summarization operators is not available in a metric space. Furthermore, the coordinates of the vector allow determining the location of an element with respect to other elements. Such a *direction* cannot be determined in a metric space.

A set of vectors can be ordered globally by component-wise sorting or by a space-filling curve [27] that better preserves the proximity of subsequent elements. In contrast, elements in a metric space can only be ordered based on the distance to a single reference object. Furthermore, it is straight-forward to divide a vector space into a finite number of distinct subsets by incrementally subdividing the space along the dimensions. In a metric space such subsets have to be defined using a center object and a radius. In general, such partitions will overlap if the complete data space should be covered.

A vector space has a *measure* that allows calculating the volume of subspaces and their intersections. In particular, this is the foundation for the definition of a density and a distribution of a data set. The notions of volume, density and distribution are not available in a metric space.

Finally, in a vector space, the costs of distance calculations between elements is linear in the number of dimension if an L_p norm (typically $p = 2$ for Euclidean distance) is used. In a metric space a distance function can be arbitrarily complex, such as e.g. the edit distance between two strings which has a quadratic runtime. Furthermore, we can calculate a bounding box of a vector data set in linear time by finding the minimum and maximum value for each dimension. In contrast, finding the maximum distance between elements in a metric space requires a quadratic number of distance computations. In summary, metric data lacks most of the tools available in traditional scenarios for cardinality estimation. This makes most established methods infeasible as we discuss in the Section 3.

However, as discussed previously, metric data appears in many different applications naturally. Furthermore, it supports distance queries, which are also highly relevant for vector data [4]. As our experiments will show later, using distance-based techniques helps lowering the impact of the curse of dimensionality.

3 RELATED WORK

The most basic idea for estimating the size of a query result is to perform the query on a sample of the data and scale up the resulting cardinality by the sample's fraction of the total data size. Using Reservoir Sampling [32], a random data selection can be computed in linear time. We can apply this method also on metric data. However, small sample sizes result in underestimates often equal to zero because metric spaces are sparse.

Histograms are the most popular technique for cardinality estimation in database systems [18]. They divide a domain into multiple buckets and store the number of contained elements. When estimating the cardinality within a given query range, they

approximate the actual cardinality usually by assuming a uniform distribution within the buckets. Computing optimal histograms that minimize the error induced by this assumption is NP-hard [25]. The most prominent example of an efficient heuristic is MinSkew [2]. It recursively subdivides the space by splitting the most skewed bucket until the desired number of buckets is reached. Other techniques like rkHist [11] and R-V histogram [1] start from the leaves of a spatial index structure and merge them together for limiting the amount of buckets.

The introduced histograms are, however, not applicable for metric data. There is no straight-forward criterion for subdividing a metric space into a finite amount of disjoint buckets. The missing notion of uniform distribution within a bucket and the unavailability of a volume measure make the incorporation of such buckets into a cardinality estimate impossible. It is possible to transform metric data into vector data in order to build a spatial histogram, though. We can then extract the cardinality estimate for a distance query by calculating the intersection of the query (in form of a hyper-sphere) with the histogram buckets. However, such a transformation into a vector space is costly and introduces an error in form of distance distortions.

Compression techniques like wavelets and cosine transformations are also suitable for cardinality estimation [24]. Both techniques are applicable to multi-dimensional vector data and are shown to provide accurate results. They approximate the actual data distribution by means of a basis function and several coefficients, thus drastically reducing the amount of data. The cardinality estimate is computed as a cumulative joint distribution of the individual dimensions of the data set. However, in a metric space we are not able to use these techniques as the data has no such dimensions and there is no notion of a distribution.

Another method for approximate query processing is Local Sensitive Hashing (LSH). LSH performs very well on data from a high-dimensional vector space. It is for example used for approximate similarity search [15] and thus related to distance queries in metric spaces. There has been work in cardinality estimation of similarity joins using LSH [21]. Also, multiple LSH indexes with different radii can be used for cardinality estimation by counting collisions of hash buckets [14]. However, LSH requires a similarity-preserving hash function which does not universally exist for metric data.

A more recent approach uses Machine Learning [4] for cardinality estimation. It is, to the best of our knowledge, the only method supporting distance queries. The query-driven approach learns to differentiate several prototype queries and predicts the cardinality of unseen queries by assignment to a prototype and subsequent interpolation using regression. The optimization of the query prototypes is performed via gradient descent where the prototype query is moved across the data space. This manipulation of a query object is not possible in a metric space. Thus, like the other approaches, this approach is infeasible for metric data, unless it is mapped into a vector space first.

A distance preserving mapping of data from a metric space to a vector space is called *embedding*. The goal is to find for each $x_i \in X$ an embedding $y_i \in \mathbb{R}^d$, such that the induced *stress* [19] on the distances is minimized. This stress measure incorporates the deviations of the resulting distances among objects with respect to the original distances.

There are different approaches available to embed metric data into a vector space [3]. One prominent example is Multidimensional scaling (MDS) [20]. It tries to preserve the pairwise distances in vector space by using such a stress function [19] and minimizing it subsequently. This minimization can be performed by eigendecomposition or gradient descent. However, both methods are expensive to compute, and thus, not suitable for very large data sets. Landmark MDS [9] was introduced as an alternative to MDS for big data scenarios. It uses samples of the data called landmarks and applies MDS on them. The remaining points are then embedded based on the distances to the l landmark elements.

Kernel estimators [26] are a competitor of histograms which exhibit a fast convergence for 1-dimensional data [7] and have been generalized to multi-dimensional data [16]. Note, that both approaches do not support distance queries on metric data. Here, samples distribute their *weight* using a kernel function K, e.g. Epanechnikov [12] or Gaussian. This weight corresponds to the probability of data points existing in the vicinity of the sample. One approximates the underlying probability density function $\hat{f}$ of a data set at the evaluation point x by using a set of samples S and summing up over all samples: $\hat{f}(x) = \frac{1}{|S| \cdot h} \sum_{s \in S} K(\frac{x-s}{h}) = \frac{1}{|S|} \sum_{s \in S} K_h(x - s)$. Here, h is the smoothing-factor called *bandwidth*. The cardinality estimate results from integrating the kernel density function within a given rectangle query and scaling the result up. In a d-dimensional vector space typically product kernels are used where the density function is integrated for each dimension separately. This is only feasible for rectangular queries and not for distance queries. Hence, the application of existing kernel-density estimators for distance queries on metric data embedded into a vector space is not straight-forward. Approximating the distance query as a hyper-sphere introduces an error that is also influenced by the curse of dimensionality. Our approach makes use of kernels, but we avoid the curse of dimensionality by using the one-dimensional distance function.

The choice of the actual kernel function is considered to be of low impact according to the literature [8]. Nevertheless, we consider different kernel functions in the experiment section of this paper. However, the selection of the kernel bandwidth h has a much more crucial impact on the resulting estimator quality.

There are two general approaches for the bandwidth selection: global and locally adaptive methods [31]. Using a global (fixed) bandwidth means that all samples and evaluation points use the same bandwidth. One method of obtaining this bandwidth is by minimizing the mean integrated squared error (MISE) [30]. In contrast to traditional applications, the underlying distribution that shall be fitted by the kernel estimator is known in cardinality estimation. It is given by the data itself. This enables other optimization techniques than those used in the statistics literature. Recent work [17] used a gradient descent based approach to find the optimal bandwidth for a given set of training queries. They fit a global bandwidth for each dimension of the vector space. However, a global bandwidth is usually not optimal, as the resulting estimator oversmoothes the distribution in dense regions and undersmoothes in sparse regions of the data set. While the authors of [17] were able to exploit the different distributions in the individual dimensions, we found the error of a global bandwidth for different query sizes in our metric scenario to be significantly high. Furthermore, a gradient descent based approach to bandwidth estimation turned out to get stuck in local optima of poor quality in our experiments. We thus also investigate locally adaptive kernel estimators that vary the bandwidth either based on

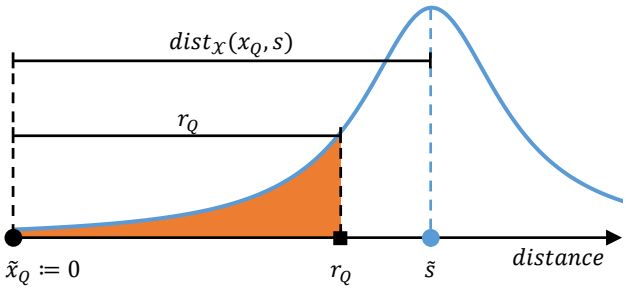

Figure 1: The incorporation of a kernel-sample s into the cardinality estimation for a query $Q = (x_Q, r_Q)$. The omitted y-axis corresponds to the probability density.

Algorithm 1: Generic Kernel Estimation Algorithm

Input : Kernel function $K_h : \mathbb{R} \to \mathbb{R}_+$, centered at 0
Optimized bandwidths $\mathcal{B} : X \times \mathcal{X} \times \mathbb{R}_+ \to \mathbb{R}_+$
Samples $S \subset X \subseteq \mathcal{X}$
Total data set size $|X|$
Distance function $dist_{\mathcal{X}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$
Query $Q = (x_Q, r_Q)$ with object and radius
Output : Estimated cardinality $\tilde{c}(Q)$

1 $total \leftarrow 0.0$;
2 **foreach** $s \in S$ **do**
3 $\quad h \leftarrow \mathcal{B}(s, x_Q, r_Q)$;
4 $\quad \tilde{s} \leftarrow dist_{\mathcal{X}}(x_Q, s)$;
5 $\quad contribution \leftarrow \int_0^{r_Q} K_h(x - \tilde{s})\,\mathrm{d}x$;
6 $\quad total \leftarrow total + contribution$;
7 **end**
8 $probability \leftarrow total/|S|$;
9 **return** $\lfloor probability \cdot |X| \rfloor$;

the sample point or the evaluation point. The latter is also called *balloon estimator* [30].

Other work in kernel-based techniques for cardinality estimation in vector spaces focuses also on improving the efficiency of the estimation process. One approach is reducing the number of samples to a so-called coreset [33] that maximizes both quality and efficiency of the estimator. In the scope of this paper we do not yet consider such improvements but focus on demonstrating the general applicability of kernel estimators to this new scenario of metric data.

4 DISTANCE-BASED KERNEL ESTIMATORS

Kernel estimators allow us to overcome a fundamental problem of using a sample directly for estimating the cardinality of a query result. Namely that the information is concentrated at a sample point. In contrast to a histogram we also get a continuous distribution. In a metric space it is, however, not straight-forward how we can apply a kernel function on a sample point, as there are no dimensions in which they could gradually distribute the mass of a sample. The central idea of our proposed technique is therefore to apply the kernel function on the distance to a sample point in order to incorporate the probability of elements in the vicinity fractionally.

In the following we show how to incorporate a sample point into the cardinality estimate. Here, the query $Q = (x_Q, r_Q)$ with object x_Q and radius r_Q is located at distance $\tilde{s} \coloneqq dist_{\mathcal{X}}(x_Q, s)$

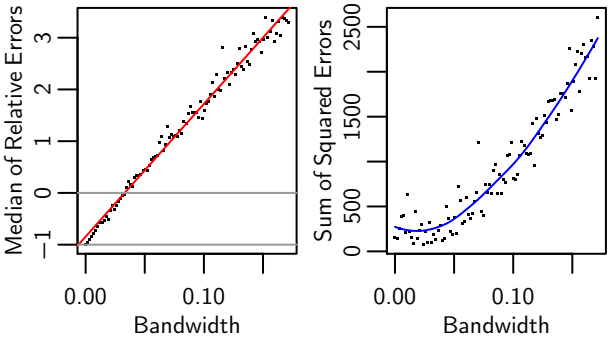

Figure 2: Influence of the bandwidth on the estimation error on the Moby data set (cf. Section 6) for a fixed query size. The left-hand side shows the median of the relative errors (Equation (2)). The right-hand side shows the sum of squared errors (measure M_{LS}).

from the sample point s. As depicted in Figure 1, we introduce an axis expressing the distance to x_Q. For that we map x_Q to $\tilde{x}_Q \coloneqq 0$, the origin of the axis. The sample point s is then mapped onto $\tilde{s}$. The kernel function K_h is then centered at point $\tilde{s}$ by subtracting $\tilde{s}$ from its argument. We take the area under the curve of the kernel function between $\tilde{x}_Q$ and r_Q as the *contribution* of this sample to the cardinality estimate.

Algorithm 1 shows the full estimation process. For each sample point we calculate the contribution and compute the sum. For this we first compute the optimized bandwidth by calling the function $\mathcal{B}$ for the given sample point and query with object and radius. In case of a global bandwidth, this function ignores the parameters and always returns the same bandwidth. In case of a locally adaptive approach, it either uses the sample or evaluation point (query) to obtain a specific bandwidth. We detail algorithms for computing the bandwidth in the next section. Given the optimized bandwidth, the distance between sample and query object, and the radius, we calculate the contribution of the sample to the running *total*. After all samples are processed, the probability is then the *total* divided by the number of samples, see line 8. Finally, we scale the resulting probability up by the total data set size and return this value as the cardinality estimate.

The general workflow of our technique consists of (1) collecting a set S of samples, (2) determining the optimal bandwidths $\mathbb{B}$ and (3) applying Algorithm 1 to estimate the cardinality of new queries. In the following we present the process of optimizing the bandwidths.

5 BANDWIDTH OPTIMIZATION

It is well-known [31] that the bandwidth of a kernel function has a crucial impact on the resulting cardinality estimate. A too small bandwidth leads to undersmoothing, a too large bandwidth to oversmoothing. The two edge cases are an infinitely small bandwidth that converges to sampling and an infinitely large bandwidth that converges to a uniform distribution. We thus take particular care of finding an optimal value. We distinguish between a global bandwidth for all samples and queries, and locally adaptive methods where the bandwidth is individually fitted to accommodate for sparser and denser regions of the data space.

5.1 Global

The computation of the optimal global bandwidth for a kernel function and a given data set is an optimization problem. We first formalize this problem and then present our optimization strategy.

5.1.1 Optimization Problem. We want to find a bandwidth h that minimizes the error of estimates for future queries on the given data set. As we do not know the future queries, we extract a set of training queries $\mathcal{Q}$ from the data set and minimize the error for these queries. Afterwards, we validate the performance against an independent set of test queries that we extracted from the data set beforehand. We formally define the optimization problem for a fixed kernel function as

$$\arg\min_h Error_X(h, \mathcal{Q}), \tag{1}$$

where h is the bandwidth, X is the data set and $Error_X$ a function that computes the error of the queries $\mathcal{Q}$ on X for the given bandwidth h.

We define an appropriate error measure for Equation (1) in two steps. First, we define an auxiliary function

$$error_X(h, Q) := \frac{\tilde{c}_h(Q) - c(Q)}{c(Q)}, \tag{2}$$

where $\tilde{c}_h(Q)$ is the estimated cardinality using bandwidth h and $c(Q)$ the actual cardinality of query Q on data set X. This measure differs slightly from the common relative error metric, as we do not take the absolute value in the numerator. This allows us to assess over- and underestimates separately. It returns values in the interval $[-1, \infty]$. Two values are of particular interest: -1 indicates that the estimator returns simply a result of zero even though there are results contained in the query. On the other hand, an error of zero indicates a perfect result: the estimated cardinality is equal to the true number of elements the query returns. There is no upper bound for our measure. However, one should notice, that a value of 1 means already an overestimation by a factor of 2.

To compute the error of a set of queries $\mathcal{Q}$ we combine the errors $error_X(Q)$ of the individual queries $Q \in \mathcal{Q}$ using a measure $M : \mathbb{R}^{|\mathcal{Q}|} \to \mathbb{R}_+$. M computes for a set of errors E a single value that is then subject to minimization. Two examples for M are the deviation of the median error from zero, and the sum of squared errors (LS for least squares):

$$M_{median}(E) := \mid median(E) \mid$$
$$M_{LS}(E) := \sum_{e \in E} e^2 .$$

For $M \in \{M_{median}, M_{LS}\}$, the final optimization problem is defined as

$$\arg\min_h Error_X(h, \mathcal{Q}) = \arg\min_h M(\{error_X(h, Q) \mid Q \in \mathcal{Q}\}) \tag{3}$$

5.1.2 Optimization Strategy. The minimization of the error function (3) requires an efficient and robust optimization method. Figure 2 shows the relationship between bandwidth and error for an example data set. On the left-hand side of the plot we observe that starting from an infinitely small bandwidth results first underestimate the true cardinality. A higher bandwidth reduces the error to a certain degree. At some point the bandwidth oversmoothes the distribution, leading to very high overestimations. The right-hand side shows the mean squared errors. While the general trend of the error function is clearly visible, we can also see that the results are noisy. This poses a difficult to find global optimum as the multitude of local optima has to be overcome. A method that has shown to be very effective in practice are Evolution Strategies.

An Evolution Strategy (ES) is a global numeric optimization approach inspired by the Darwinian theory of natural selection. We implemented the approach of Beyer and Schwefel [6]. Here, μ parents produce another set of λ offspring. From the thus obtained set of $\mu + \lambda$ individuals the best μ individuals are selected for the next generation based on a fitness function. An offspring is produced using a recombination of p parents followed by mutation. The evolutionary process is repeated until either a fitness threshold, a number of stall iterations without improvement, or a total number of iterations is reached.

In our case each individual represents a bandwidth h. For calculating the fitness, we use the error function from Equation (3) which we want to minimize. We tried different parametrizations and found the following to provide a good compromise between runtime and estimation errors: $\mu = 100, \lambda = 300, p = 2$, recombination via mean and mutation via a Gaussian distribution. We stop the optimization process after either 25 stall iterations or a given total amount of 1000 iterations is reached.

As the fitness function has no side-effects and only depends on the parameter h we can easily parallelize the evolutionary process by computing the score for different individuals on different threads. Because the evaluation of the fitness function is the most expensive part of the computation we can effectively scale up the throughput linearly in the amount of CPU cores.

We also exploit the fact that only a limited number of distinct distance computations are required. In the fitness evaluation for a given bandwidth h we need to compute a new estimate for each training query based on h. However, the distances of the query objects to the samples that give the interval for the integral in line 5 of Algorithm 1 remain constant. In our implementation we thus precompute these distances and store them in a matrix $\mathcal{D}$. In the iterations of the optimization process we then avoid redundant recomputations by performing simple look-up operations. The matrix $\mathcal{D}$ is of size $|S| \cdot |O|$, where S is the set of samples and $O := \{x_Q \mid (x_Q, r_Q) \in \mathcal{Q}\}$ is the set of distinct training query objects. $\mathcal{D}$ is discarded after the optimization process is finished. It thus only influences the construction-time space complexity of our algorithm. Reasonable numbers are 10^2 distinct query objects and 10^4 samples as used in our experiments. This means $\mathcal{D}$ typically requires only a few megabytes of memory.

5.2 Locally Adaptive

A fixed global bandwidth is usually insufficient for approximating the cardinality of small and large queries in dense as well as sparse regions. One possibility is to extend the bandwidth optimization process from the previous subsection to individual bandwidths per sample point. For the following reasons we refrain from this direction and rather focus on the balloon estimator [30].

In the Evolution Strategy we can extend the configuration to incorporate individual bandwidths. However, this leads to a very high-dimensional optimization problem as we potentially need to optimize hundreds or thousands of bandwidths simultaneously. Such an optimization may take very long to converge or get stuck in a local optimum of overall poor quality.

Another approach is to use coordinate search, where we optimize only one bandwidth at a time, keeping the others fixed. Here, we would initialize the N individual bandwidths h_i e.g.

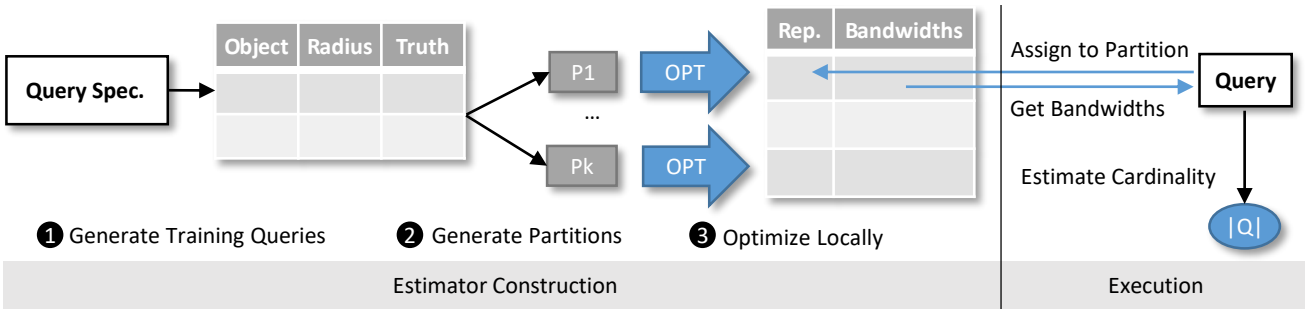

Figure 3: A general overview of the locally adaptive estimator construction and execution.

randomly and then optimize them one at a time in the order $i = 1, \ldots, N, N-1, \ldots 1$. We repeat this until no significant improvement is made in one full sweep. This approach, however, is also prohibitively expensive to compute.

We thus limit our investigation to the balloon estimator defined by the k^{th}-nearest neighbor estimator [23]. It chooses the bandwidth on basis of the evaluation point (query) and the surrounding sample points. In contrast to the traditional balloon estimator, we again use a set of training queries in the optimization process. Given a query $Q = (x_Q, r_Q)$ and the samples S we choose the bandwidth h as follows:

$$h = \mathcal{B}(\_, x_Q, r_Q) \propto distance(x_Q, sample_k(x_Q)) \,, \tag{4}$$

where $sample_k(x_Q) \in S$ is the k^{th}-nearest neighbor of the query object x_Q in S, and $\mathcal{B}$ the function returning the optimized bandwidth as introduced in Algorithm 1. We thus ignore the first parameter of $\mathcal{B}$ and apply the same bandwidth to all samples in the final cardinality estimation of the query Q. Obviously, the distance to the k^{th}-nearest neighbor is smaller in denser regions than it is in sparser regions. This leads to the desired adaptive bandwidth.

Using the distance to the k^{th}-nearest neighbor directly, however, turns out to be insufficient. First of all, we need a linear scaling factor to transform the distance to a bandwidth. Furthermore, we incorporate the query radius into the bandwidth optimization as well. As our experiments will show, the optimal bandwidth varies for different query sizes (selectivity). However, we only know the radius at query time. The result size is precisely the value we want to estimate.

We can, however, distinguish differently sized queries by the relationship between the query's radius and the distance to the k^{th}-nearest neighbor sample. Here, the latter is an indicator of the density around the query object. We denote this relationship as

$$\rho(Q) = \frac{r_Q}{dist_X(x_Q, sample_k(Q))} \,. \tag{5}$$

It is an indicator for the cardinality of the query. If we fix the query radius, $\rho(Q)$ gets larger the smaller the distance to the k^{th}-nearest neighbor gets. This matches our intuition as this indicates the data space becoming more dense. On the other hand, if we fix the density in form of the denominator, a larger radius corresponds to a larger ρ value, indicating a larger cardinality.

For our extended balloon estimator called *BalloonEstimator*$^+$ (B^+ for short) we thus optimize the local bandwidth for a given query Q depending on $\rho(Q)$. Here, we again make use of the methods presented in the previous subsection. The optimization process consists of the following steps which are also presented in Figure 3:

1. Generate a set $\mathcal{Q}$ of training queries based on a query specification (e.g. m query objects with a set of target selectivities). This involves calculating for a given query object x_Q a radius r_Q and the true cardinality that fulfill the query specification.
2. Divide $\mathcal{Q}$ into k disjoint partitions $P_i \subseteq \mathcal{Q}$ such that the intra-partition variance $Var(\{\rho(Q) \mid Q \in P_i\})$ is minimized.
3. For each partition P_i compute the optimal bandwidth h_{P_i} that minimizes $Error_X(h_{P_i}, P_i)$.

At query time we assign the query to a partition and use the optimized bandwidth.

In order to obtain the k partitions required in Step 2 we use *hierarchical* clustering with complete linkage based on ρ. We extract the k partitions from the dendrogram. For each partition we then compute the locally optimal bandwidth by means of an ES. Each partition is then identified by the mean ρ-value over all queries in the partition: $\overline{\rho}(P_i) = \frac{1}{|P_i|} \sum_{Q \in P_i} \rho(Q)$. Given a new query Q we first assign it to the nearest partition P^* by

$$P^* = \arg\min_{P_i} |\overline{\rho}(P_i) - \rho(Q)| \,.$$

We then use h_{P^*} as the bandwidth for the kernel functions when estimating the cardinality of query Q.

The depicted construction process of the B^+ estimator requires several distance computations and incurs some overhead in storage. For calculating the k-nearest neighbor when determining the ρ-value of the training queries we use the same matrix $\mathcal{D}$ as for the actual bandwidth optimization. Storing the B^+ estimator requires storing for each cluster the mean ρ-value and the optimized bandwidth. This slightly reduces the amount of samples that can be stored with a given amount of space.

When estimating the cardinality for a given query Q, we induce some overhead with respect to a kernel estimator with a global bandwidth. Namely, we need to compute $\rho(Q)$ which includes finding the k^{th}-nearest neighbor sample. However, we calculate the distances of all samples to the query object anyway when calculating the estimation after the bandwidth has been determined. Thus, the additional overhead of the B^+ estimator only consists of finding the k-smallest element in the list of distances. We can efficiently determine this element while computing the distances using a bounded max-heap. Here, we always store the k smallest elements and larger elements are discarded. At the end the k-smallest distance is at the top of the heap. We then incorporate this distance in the calculation of $\rho(Q)$ and retrieve the bandwidth of the nearest partition. The remaining computations are identical to a kernel estimator with a global bandwidth.

6 EXPERIMENTAL EVALUATION

This section presents the results of our experimental study. First, we present the experiment setting, data sets and investigated methods. Then, we present the results of the experiments.

6.1 Setting

We implemented the kernel estimators and baseline algorithms in Java using embedding techniques and histograms from the XXL library [10]. All experiments were run on an Intel Core i7-4771 CPU and 16GB of RAM. We consider four real-world data sets which, together with our generated queries, are on our website[1].

Moby Word List (Moby) contains word lists in different languages[2]. We used the German word list consisting of about 160 000 words and the Levenshtein distance [22] as a metric.

Protein Binding Sites (PBS) are taken from CavBase [29], a database of protein binding sites (PBS) from experimentally determined protein structures. In CavBase, currently 248 686 PBS are stored. Each PBS is described by a set of points in the 3-dimensional Euclidean space that model the shape of the protein. In addition to the coordinate, each point also carries a label specifying the physico-chemical property of that point. To compare pairs of PBS we use the measure presented in [13] which fulfills all metric properties.

Rea16 is a 16-dimensional vector-data set representing Fourier coefficients from CAD data. It contains 1.3 million points and is often used for benchmarking high-dimensional queries [5]. We use the Euclidean distance measure as metric.

Wikipedia (Wiki) is a data set consisting of 4.5 million Wikipedia articles represented by the article name and a short abstract. We used the data provided by the DBpedia project[3]. We removed articles that are only a disambiguation or a list of links to other articles. For easier syntactic comparisons we also removed all non-ASCII characters. The distance measure for this data set is based on the Jaccard coefficient of character shingles[4] of length $k = 3$. The distance between two articles is computed by $dist(a, b) = 1 - \frac{|s_k(a) \cap s_k(b)|}{|s_k(a) \cup s_k(b)|}$, where a and b are articles and s is a function that computes the set of k-shingles of a short abstract.

We generate our test queries uniformly at random from the data set at hand. For this, we select 100 items as query objects and remove them from the data set. In particular we remove these items from the data set before optimizing the kernel bandwidths on a different set of training queries. We then choose the minimum radius around these items such that at least 0.01%, 0.1% and 1% of the data set are contained in the query. This radius varies depending on the density at the particular query object. The training queries are generated in the same fashion as the test queries. Note that it is not always possible to create a query from a data object that contains exactly the desired amount of data. Especially for discrete distance measures, many objects share the same distance to a reference object. Also note that by generating queries this way, they follow the distribution of the data. This means that it is more likely for a query to be placed in a more dense region of the data space. However, this is no undesired effect, since this approach intuitively matches real-world query patterns [17]. In contrast to vector spaces it is not possible to uniformly sample queries from a domain for metric data (e.g. of newspaper articles). We are thus forced to limit our investigation to queries drawn uniformly from the data items.

In our experiments we compare the estimated cardinality for a given query with the actual cardinality – the closer both values, the better the estimator. We use our error measure from Equation (2) again. Recall that we use this slight variation of the common relative error to capture under- and overestimations individually.

For measuring the build time of the individual estimators we use wall-clock time. Note that we parallelized the Evolution Strategy and used all 8 threads of the machine. For our kernel-based approaches we measure only the time for creating the estimator and not for generating the training queries as, in practice, they can e.g. be obtained from historic queries or the current workload.

In addition to our kernel estimators we consider spatial histograms as a baseline method for cardinality estimation. For metric data, the spatial histograms require an embedding into a vector space. We use the following popular measure to assess the quality of the resulting embeddings:

$$STRESS_1 = \left(\frac{\sum_{i<j} (dist_{\mathcal{X}}(x_i, x_j) - dist_E(y_i, y_j))^2}{\sum_{i<j} dist_{\mathcal{X}}(x_i, x_j)^2} \right)^{\frac{1}{2}} , \quad (6)$$

where $dist_E$ is the Euclidean distance. It measures on a relative scale how well distances are preserved by the specific embedding. According to the literature [19] a value greater than 0.2 is already considered as a low-quality embedding.

We evaluated the performance of different kernels with optimized bandwidths. Even though the concrete choice of kernel function is considered to be insignificant in the literature [8], we found differences among the optimization results. We consider Cauchy, Epanechnikov, Exponential and Gauss kernels.

We now describe the estimators we used in our experiments. All of them are given 0.1% of the total data set size in memory to allow a fair comparison across the data sets of varying sizes. This size presents a reasonable value in big data scenarios and returned representative results in our experiments.

Sampling uses reservoir sampling to efficiently build a uniform random sample S of the data in one pass. The query is executed on the sample and the resulting cardinality $c_S(Q)$ is scaled up to return the estimated cardinality $\tilde{c}(Q) = c_S(Q) \cdot |X|/|S|$.

MS*n*d and R-V*n*d use a MinSkew or R-V histogram, respectively. The parameter n indicates that the data was first embedded in an n-dimensional vector space using Landmark MDS. A distance query Q is then also embedded and becomes an n-dimensional query sphere $\widehat{Q}$. For estimating the cardinality we calculate the volume $V_I(\widehat{Q}, B)$ of the intersection between $\widehat{Q}$ and histogram bucket B and sum up the fractional bucket counts:

$$\tilde{c}(Q) = \sum_B V_I(\widehat{Q}, B) \cdot count(B) .$$

For $n > 2$ we use a Monte Carlo simulation for V_I, as it is more efficient than an analytical solution.

Global *size* uses a kernel estimator with the Cauchy kernel function and a global bandwidth optimized by means of an Evolution Strategy, as described in Subsection 5.1, with 100 individuals. We use M_{median} as defined in Section 5. The parameter *size* gives the cardinality of the training queries: $S \triangleq$ small $\triangleq 0.01\%$, $M \triangleq$ medium $\triangleq 0.1\%$, $L \triangleq$ large $\triangleq 1\%$, $MIX \triangleq X \triangleq$ mixed $\triangleq S \cup M \cup L$. The query objects are selected uniformly at random.

B$^+$*k,n* uses a kernel estimator with the Cauchy kernel and locally optimized bandwidths as described in Subsection 5.2 using

[1] http://uni-marburg.de/qiFqp
[2] http://icon.shef.ac.uk/Moby
[3] http://wiki.dbpedia.org/services-resources/documentation/datasets#ShortAbstracts
[4] A k-shingle is a sequence of k characters

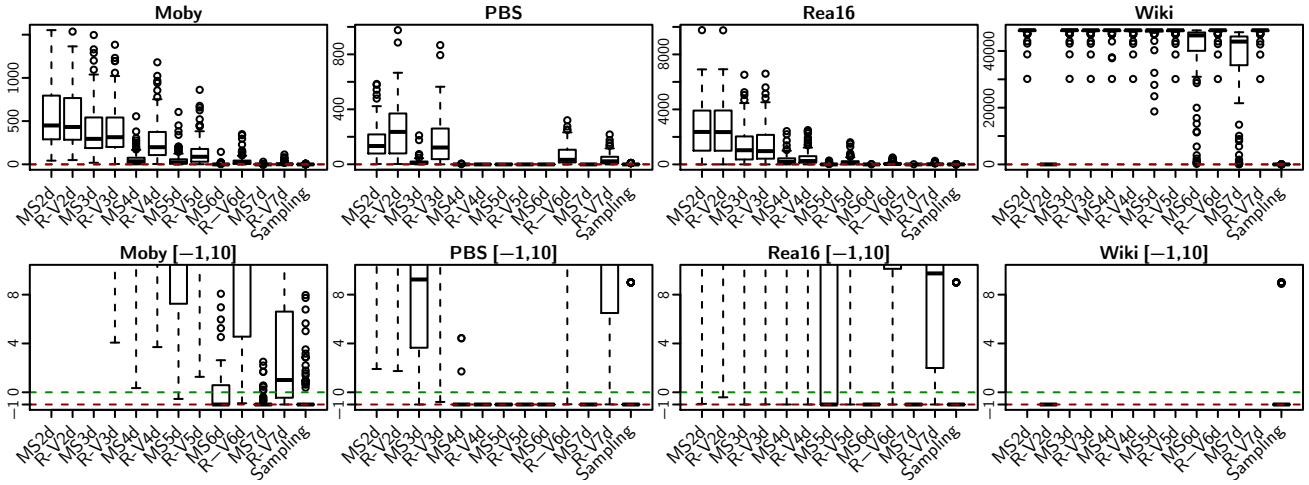

Figure 4: Estimation errors of the spatial histograms on queries of 1% of the data. The green line indicates an error of 0, the red line an error of -1 (zero-estimate). The top row plots show the total error range, the bottom row the interval $[-1, 10]$.

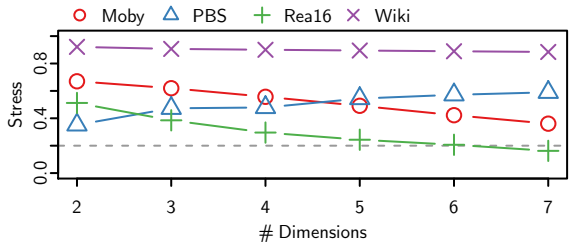

Figure 5: The stress induced by the embedding into a vector space of d dimensions. The dashed line indicates the threshold for a low-quality embedding.

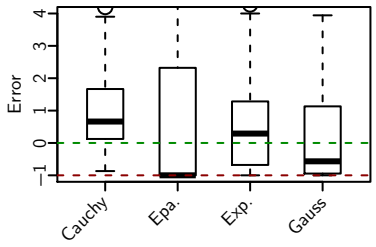

Figure 6: Influence of the choice of kernel function on the Moby data set for a query size of 0.01%.

the same measure as *Global* on the *MIX* training queries. The parameter k expresses that the k^{th}-nearest neighbor is used and n defines the number of clusters.

6.2 Estimation Quality

We now present the results of our experiments of the described estimators on the introduced data sets. We visualize the estimation errors in terms of boxplots.

6.2.1 Baseline Algorithms. Figure 4 shows the estimation errors of MinSkew and R-V histogram for the individual data sets and a query size of 1%. We embedded all data sets into 2, 3, 4, 5, 6 and 7 dimensions using Landmark MDS with 10 landmark points. We can immediately see very large estimation errors. Using a higher target dimension for the embedding leads to lower estimations. However, this does not lead to good estimates. In contrast, the very high estimates turn into zero-estimates ($error = -1$) for higher dimensions.

Figure 5 shows the stress induced by the embedding. Overall, the embeddings are of poor quality with stress values above 0.2 as indicated by the dashed line. We observe that, in general, the stress decreases with the number of used dimensions. For PBS we notice a slightly worse embedding for higher dimension. This reveals a weakness of the Landmark MDS embedding. Only the landmarks profit directly from the higher degree of freedom in a higher dimensional vector space. They are properly embedded using an MDS that minimizes the stress. The remaining elements are only embedded with respect to the landmarks, not with respect to the other objects. Thus, a higher dimensionality does not necessarily guarantee a better embedding.

In addition to the poor embedding quality, the spatial histograms suffer from the curse of dimensionality. This leads to an overall performance degradation with an increasing dimensionality. The vector space becomes sparse leading to almost empty buckets or queries that intersect with no buckets at all. This is apparent as the high overestimations induced by the embedding turn into zero-estimates for higher dimensions. We conclude that spatial histograms are not a reliable choice for cardinality estimation in metric spaces. For the vectorial Rea16 data set in the original 16-dimensional vector space (omitted in Figure 4 due to lack of space), nearly all queries return an estimate of zero. This motivates the idea of using distance-based approaches on high-dimensional data in order to counter the curse of dimensionality.

Figure 4 also shows the performance using a random sample for estimating the cardinality. The resulting cardinality on the sample is scaled up by the constant factor $|X|/|S|$, where S is the sample and X the data set. If the sample is large enough with respect to the query size it reflects the data distribution well and estimates become accurate. However, if the sample is small or the query highly selective, errors increase dramatically. Estimates are either zero because no sample is contained in the query, even though there are qualifying items in the underlying data set. Or, estimates are very high because by chance a sample was included and then scaled up by a large factor. This results in a median of -1 and a very high maximum error.

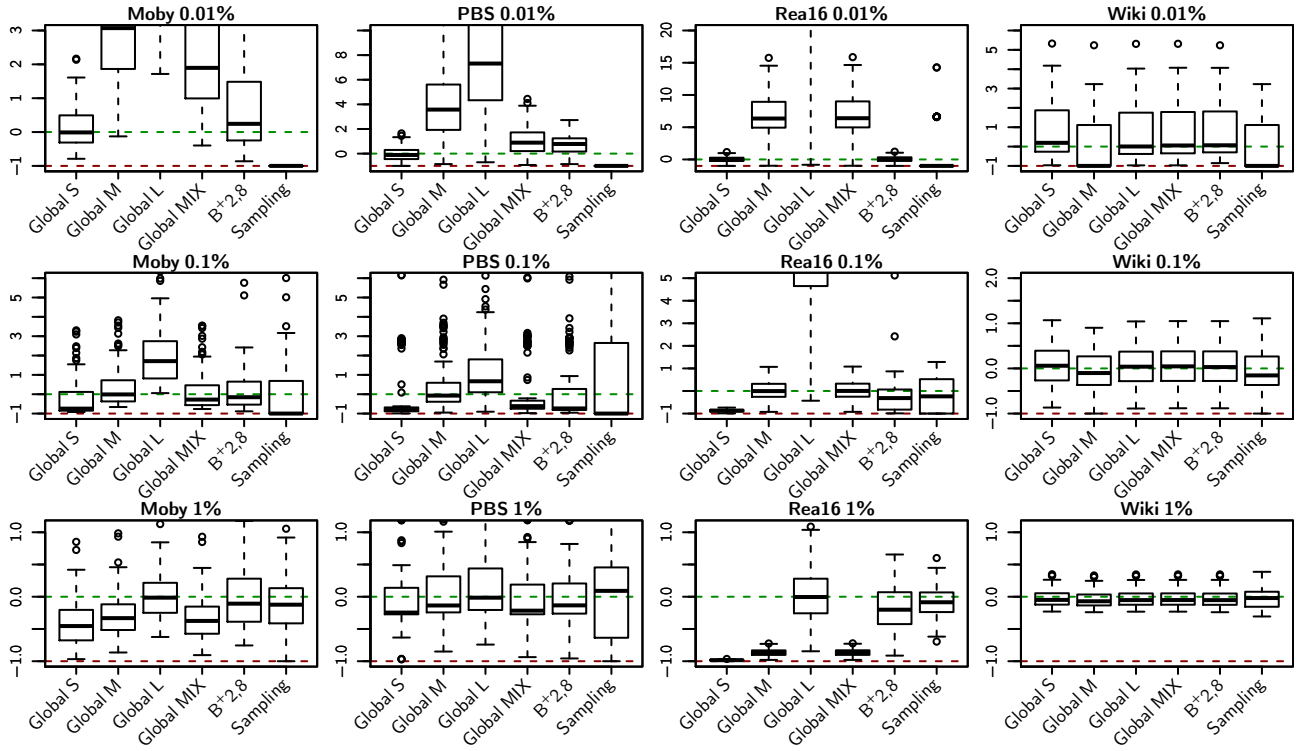

Figure 7: Estimation errors of the kernel estimators with global and local optimized bandwidths for different query sizes.

6.2.2 Kernel-Based Techniques. Figure 6 exemplifies the performance of different kernel functions using an optimized bandwidth as described in Section 5. It is apparent that different kernel functions lead to a different estimation accuracy. However, our experiments confirm that the selection of the bandwidth has a much greater impact on the performance than the choice of the kernel function. As the Cauchy kernel gave the overall best performance in our experiments, we limit our following presentation to this kernel. The reason for its superior performance is that its distribution is more heavy-tailed than that of the other kernels. In contrast to, e.g., the Gaussian kernel, the density does not decrease exponentially in the distance from the mean. This allows the kernel to distribute its weight more effectively.

Figure 7 shows the estimation errors of the kernel estimators with globally and locally optimized bandwidths. We also report the statistics in Tables 2-5 at the end of the paper. In contrast to sampling the kernel estimators are able to give reasonable cardinality estimations also for very selective queries. In particular they overcome the problem of either zero estimates or very high overestimates. For the global bandwidth optimization we generated training queries of different cardinality. In Figure 7 the labels *S*, *M*, *L* and *MIX* indicate the target query size for the bandwidth optimization process.

Kernel estimators optimized for a specific query size give good results on new queries of equal size. This proofs that the bandwidth generalizes well from training to test queries. However, estimators optimized for small queries tend to underestimate the cardinality of larger queries. The optimal bandwidth for larger query sizes also leads to overestimates for smaller queries. Optimizing the bandwidth for all query sizes at the same time leads to a bandwidth that is not optimal for any of the query sizes. This result shows that a global bandwidth is not sufficient for applications where queries vary greatly in size.

The *BalloonEstimator*$^+$ (B^+k, n) uses the same set of training queries, but computes a set of optimal bandwidths. The concrete bandwidth for a given query is chosen based on the local density and the query radius. We restrict our results to the setting $k = 2$ and $n = 8$ that gave slightly better results than for other settings.

Figure 8 displays how many results of good quality are produced by sampling and $B^+2, 8$, respectively. It confirms the results indicated by the boxplots that the kernel approach outperforms sampling for small queries. The plot shows the percentage of estimates within the error bounds given by the x-axis. Thus, it begins with all estimates with error zero, meaning perfect results. It then gradually increases the bounds uniformly in both directions, thus accumulating more results. The x-axis ends with all queries of errors in the interval $]-1, 1[$, as we consider greater errors as a failure of the estimator. Accordingly, we can see the number of estimates not fulfilling our requirements by the height of the plot. For all data sets the smallest queries show no results for sampling with absolute errors below 1. $B^+2, 8$ on the other hand is able to produce high quality results within the shown interval. Moreover, the shape of the curve is very consistent between the different query sizes and data sets, indicating the robustness of the approach. Sampling shows step-wise increments when new sample points are incorporated into the estimate.

The concrete choice of n and k is subject to optimization. We report one setting ($k = 2$, $n = 8$) that worked for all our data sets. There is a general tradeoff between the number of clusters (improving the estimation quality) and the required space (worsening the quality). In general, we observed in our experiments that large values for n and k do not improve the performance. Overall we conclude that the BalloonEstimator$^+$ presents a significant improvement over the global bandwidth approach.

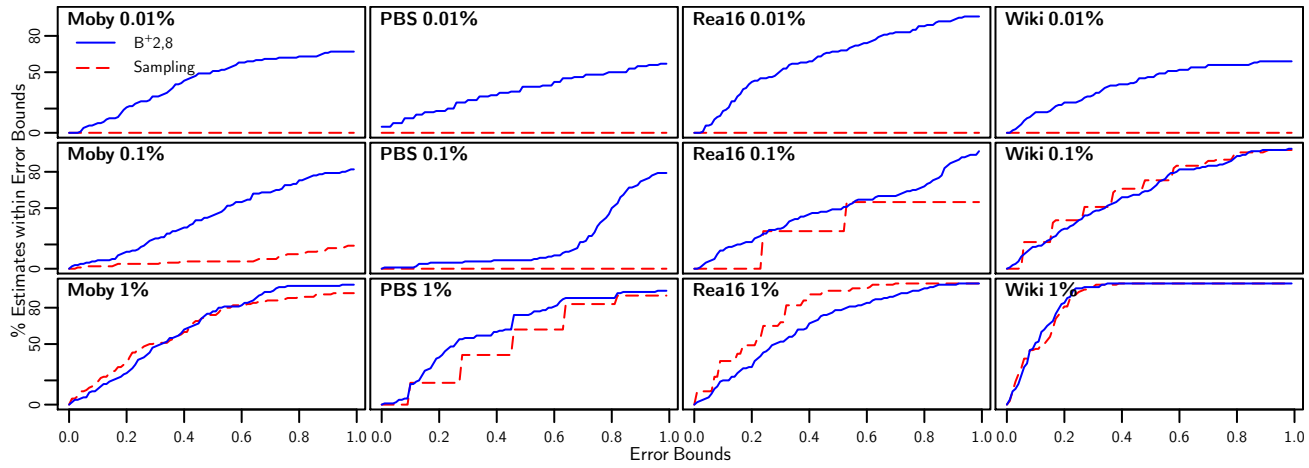

Figure 8: The plotted lines display the number of estimates with absolute errors within the bounds, given by the x-axis.

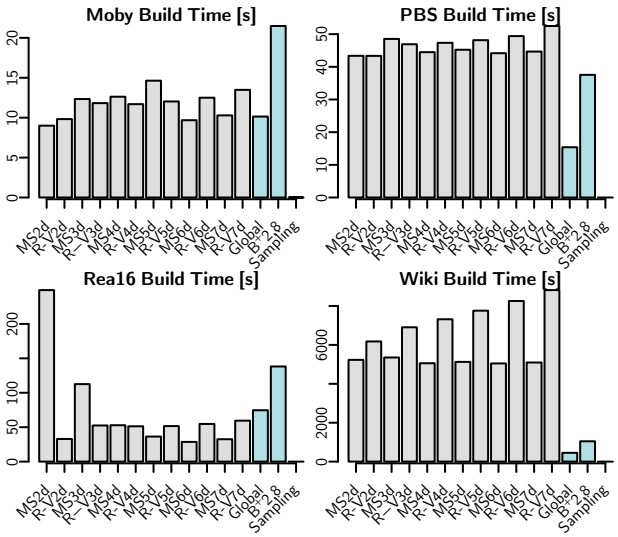

Figure 9: The build time of the estimators in seconds.

6.3 Runtimes

The efficiency of an estimator is the second criterion for assessing its performance. We first discuss the theoretical complexity of the construction of our B$^+$ estimator. Then, we present our empirical findings of the build and query time of all investigated methods.

6.3.1 Construction Complexity of the B$^+$ Estimator. The time complexity of the construction of our B$^+$ estimator is determined by (1) the cost C_P for creating P partitions of the training queries and (2) the cost C_{ES} of executing the ES which is dominated by evaluating the fitness of the individuals. All other operations are negligible because they cause only constant overhead.

The number of partitions P is defined by the user. The cost for partitioning the training queries $\mathcal{Q}$ using hierarchical clustering is dominated by the construction of the dendrogram which requires $C_P = O(|\mathcal{Q}|^2)$ time.

The fitness evaluation of a single individual requires calculating the cardinality estimate for each training query in $\mathcal{Q}$ by incorporating the contribution of all samples S. We denote this cost as $c_{\text{eval}} = O(|\mathcal{Q}| \cdot |S|)$. Given the number of iterations I, the number of parents μ and the number of offspring λ the cost for executing the ES is $C_{\text{ES}} = O((\mu + I \cdot \lambda) \cdot c_{\text{eval}})$. As we execute one ES for each of the P partitions, the final time complexity is $O(|\mathcal{Q}|^2 + P \cdot (\mu + I \cdot \lambda) \cdot |\mathcal{Q}| \cdot |S|)$.

The time complexity depends on the number of iterations I of the ES. The termination condition is a given number of stall iterations $\bar{I}$ and thus depends on the fitness development of the individuals. We can consider I as a random variable. For $\bar{I} = 25$, we observed the following number of iterations in our experiments: min. 79, 1st quantile 86, median 89, mean 93.57, 3rd quantile 101 and max. 136. Note that we also limit the ES to at most 1 000 iterations which gives an upper bound on the runtime.

6.3.2 Empricial Findings. Figure 9 shows the build times of the estimators from Subsection 6.2. The spatial histograms show a more or less constant construction time, regardless of the dimension. Sampling is the cheapest estimator to construct as it only requires a single scan of the data set. In comparison, our kernel-based techniques take more time to build. As our B$^+$ estimator performs multiple bandwidth optimization for the different query partitions, its runtime exceeds the global bandwidth approach. However, the required time for construction is still very reasonable in practical applications, especially as the estimator has to be built only once which makes the query response time much more crucial. We observed a very quick convergence of the Evolution Strategy in our experiments, where an optimum was usually found within very few iterations. We report the average build time for the different used configurations. The bandwidth optimization was performed in parallel, while the spatial histograms were constructed on a single thread. As the ES is very easy to parallelize, we can thus improve the construction time using more CPU cores. We also want to emphasize that a stochastic approach is only one method to bandwidth optimization. If another method turns out to be more efficient in this regard, it does not lower the accuracy of a kernel estimator. However, the build time is already very reasonable for practical scenarios. For Moby and PBS the construction of our estimator took less than a minute and for Rea only slightly more than two minutes. This means the overhead of constructing the estimator while importing the data into a database is insignificant. On the Wiki data set the construction was significantly slower with ~17 minutes. This has two reasons: The first is that the estimator size is much larger, as we choose it in relation to the data set size in our experiment. The second reason is that the distance computations

Table 1: Mean query times of the estimators in ms.

Method	Moby	PBS	Rea16	Wiki
MS2d	0.232	0.210	0.985	1.825
R-V2d	0.075	0.935	1.229	0.752
MS3d	5.114	114.848	438.672	363.161
R-V3d	4.968	723.424	438.997	4 868.446
MS4d	4.391	41.765	158.357	97.783
R-V4d	4.270	0.055	384.439	4 239.452
MS5d	4.072	21.134	35.995	35.169
R-V5d	4.020	0.051	351.146	3 854.658
MSd	0.203	0.227	0.203	0.967
R-V6d	3.551	915.755	328.374	3 609.655
MS7d	0.224	0.278	0.231	0.992
R-V7d	3.480	872.779	313.527	3 429.448
Kernel Global	0.105	1.032	0.209	354.941
$B^+2,8$	0.138	1.071	0.299	348.178
Sampling	0.104	1.412	0.241	351.645

are the most complex of all data sets. This is also visible in the high construction times of the spatial histograms.

Table 1 shows the query response times of the cardinality estimation. The spatial histograms suffer from slow response times for dimensions greater than 2 which is amplified by the more complex calculation of the sphere-bucket intersection. For some experiments the time gets close to zero as no intersecting buckets were found due to the exponentially increasing vector space. Sampling, again, has the benefit of being very fast, as only few distance calculations are necessary. The kernel estimators additionally need to compute an integral for every sample point which results in a longer execution time. However, as reflected by the query times, this turns out to be insignificant overhead in comparison to the distance computations. The B^+ estimator is also very efficient. Our experiments confirm that the determination of the appropriate bandwidth induces only very little overhead. In conclusion kernel-based cardinality estimation is an efficient method with effectively zero overhead in comparison to sampling in practical applications. The only reason to use sampling is its low cost for computing the estimator.

6.4 Impact of Estimator Size

The estimator size limits the number of samples used for calculating the cardinality estimate and thus impacts the quality of the estimates. We expect that a larger estimator size leads to overall better estimates. For estimator sizes of 0.05%, 0.1%, 0.2% and 0.4% of the total data set size we measured the performance of the estimators in 10 runs using different samples. In each run we, again, measure the error of 100 test queries. We compute the trimmed mean of the thus generated 100 error values with a ratio of 10% to accommodate for outliers. Figure 10 shows boxplots of the trimmed means for our $B^+2,8$ estimator and pure sampling with a query selectivity of 0.01%. For both estimators we observe an overall improvement of the estimates with greater estimator sizes. Our $B^+2,8$ estimator is able to already give good results when given only a very limited amount of space. The improvements for larger estimator sizes are relatively small for all data sets. In contrast, we can see that sampling benefits much greater from more available data. It starts with very bad results for such selective queries with a median of zero when only few sample are used. Overall we observe a tendency to underestimates while

Table 2: Estimation errors on the Moby data set.

		Glob S	Glob M	Glob L	Glob X	$B^+2,8$	Sampling
0.01%	Min	-0.790	-0.128	1.717	-0.397	-0.863	-1.000
	Median	-0.011	3.070	11.594	1.896	0.243	-1.000
	Max	7.613	12.167	40.111	8.581	57.842	61.750
0.1%	Min	-0.919	-0.661	0.056	-0.765	-0.884	-1.000
	Median	-0.744	-0.011	1.711	-0.277	-0.155	-1.000
	Max	3.302	3.823	6.348	3.546	6.869	6.007
1%	Min	-0.967	-0.864	-0.624	-0.906	-0.755	-1.000
	Median	-0.455	-0.331	-0.014	-0.375	-0.107	-0.123
	Max	0.852	0.984	1.505	0.931	1.177	1.703

Table 3: Estimation errors on the PBS data set.

		Glob S	Glob M	Glob L	Glob X	$B^+2,8$	Sampling
0.01%	Min	-1.000	-0.852	-0.704	-0.926	-0.852	-1.000
	Median	-0.111	3.574	7.315	0.889	0.782	-1.000
	Max	37.111	41.667	46.333	38.333	37.593	36.037
0.1%	Min	-0.993	-0.949	-0.905	-0.978	-0.956	-1.000
	Median	-0.818	-0.071	0.670	-0.617	-0.735	-1.000
	Max	6.442	7.011	7.551	6.595	7.022	6.299
1%	Min	-0.969	-0.850	-0.742	-0.936	-0.957	-1.000
	Median	-0.246	-0.136	-0.015	-0.216	-0.134	0.091
	Max	1.538	1.507	1.485	1.529	1.546	2.638

the errors of our approach are more centered around zero. With greater estimator size, sampling is able to catch up with our kernel-based approach for some data sets. However, small estimator sizes with respect to the total data set size are highly relevant in practical scenarios because the amount of data in databases is rapidly growing. This makes our estimator superior to sampling in practical scenarios.

7 CONCLUSION AND FUTURE WORK

We presented the first effective and efficient approach to cardinality estimation on metric data. Our approach is based on the application of kernel-density techniques to the one-dimensional distance function. We are able to outperform sampling which suffers from zero-estimates and large overestimates for small queries. Our approach is also superior to spatial histograms on embedded data which suffer both from the poor quality of embeddings and the curse of dimensionality. We presented approaches for determining the bandwidth of the kernel estimator globally and locally adaptive.

In our future work we will further investigate efficient local bandwidth optimization strategies. Furthermore, we will look into adapting our estimator to changing data and query workloads by performing the ES continuously in the background. We will also address the incorporation of query feedback into the estimation process.

REFERENCES

[1] Daniar Achakeev and Bernhard Seeger. 2012. A Class of R-Tree Histograms for Spatial Databases. In *SIGSPATIAL/GIS*. ACM, New York, NY, USA, 450–453.
[2] S. Acharya, V. Poosala, and S. Ramaswamy. 1999. Selectivity estimation in spatial databases. *ACM SIGMOD Record* 28, 2 (1999), 13–24.
[3] Charu C Aggarwal. 2015. *Data mining: the textbook*. Springer.
[4] Christos Anagnostopoulos and Peter Triantafillou. 2017. Query-Driven Learning for Predictive Analytics of Data Subspace Cardinality. *ACM Trans. Knowl.*

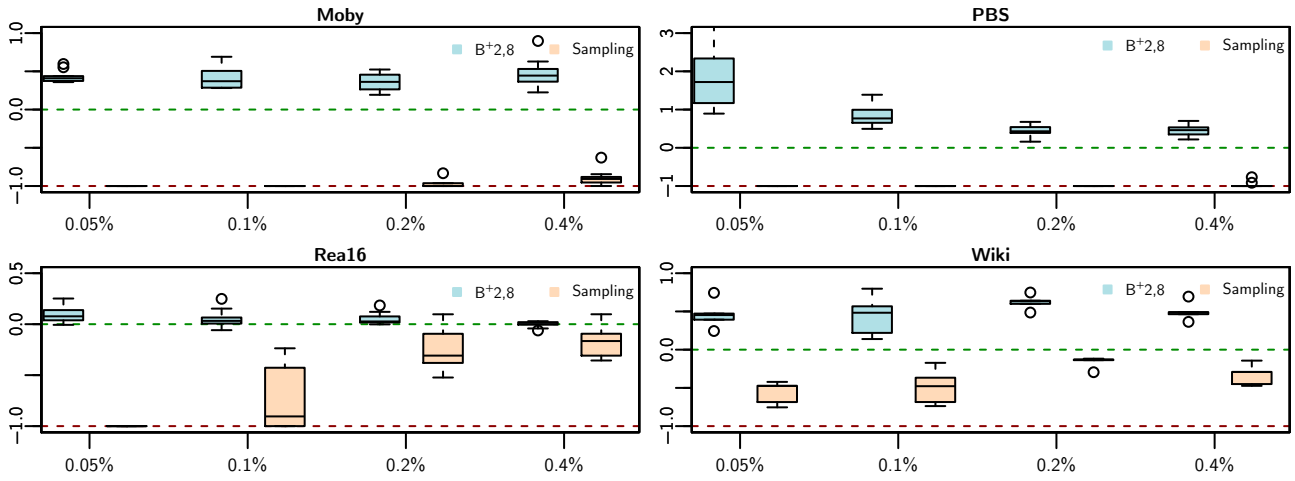

Figure 10: The trimmed mean estimation errors for estimators of varying sizes on a fixed set of 100 queries with target selectivity of 0.01%. The estimator sizes are given in % of the total data set size. For each size, the boxplot includes the results of 10 runs with different samples.

Table 4: Estimation errors on the Rea16 data set.

		Glob S	Glob M	Glob L	Glob X	B+2,8	Sampling
0.01%	Min	-1.000	-0.977	-0.826	-0.977	-1.000	-1.000
	Median	-0.061	6.347	54.966	6.397	-0.004	-1.000
	Max	1.145	15.756	126.191	15.870	1.206	14.267
0.1%	Min	-0.991	-0.925	-0.431	-0.925	-0.990	-1.000
	Median	-0.873	-0.004	6.583	0.003	-0.314	-0.238
	Max	-0.736	1.069	14.708	1.082	5.122	1.287
1%	Min	-0.997	-0.980	-0.845	-0.979	-0.913	-0.695
	Median	-0.983	-0.869	-0.004	-0.868	-0.201	-0.085
	Max	-0.965	-0.725	1.087	-0.723	0.655	0.601

Table 5: Estimation errors on the Wiki data set.

		Glob S	Glob M	Glob L	Glob X	B+2,8	Sampling
0.01%	Min	-0.966	-1.000	-0.973	-0.971	-0.856	-1.000
	Median	0.192	-1.000	0.004	0.057	0.065	-1.000
	Max	5.324	5.237	5.310	5.314	5.237	3.228
0.1%	Min	-0.867	-1.000	-0.888	-0.882	-0.883	-1.000
	Median	0.060	-0.102	0.039	0.046	0.030	-0.154
	Max	1.068	0.901	1.042	1.049	1.048	1.110
1%	Min	-0.230	-0.240	-0.231	-0.231	-0.240	-0.306
	Median	-0.049	-0.070	-0.052	-0.051	-0.052	-0.020
	Max	0.349	0.330	0.346	0.347	0.346	0.387

Discov. Data 11, 4, Article 47 (June 2017), 46 pages.

[5] Norbert Beckmann and Bernhard Seeger. 2009. A Revised R*-tree in Comparison with Related Index Structures. In *SIGMOD Conf.* ACM, NY, USA, 799–812.

[6] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution strategies – A comprehensive introduction. *Natural Computing* 1, 1 (2002), 3–52.

[7] Björn Blohsfeld, Dieter Korus, and Bernhard Seeger. 1999. A Comparison of Selectivity Estimators for Range Queries on Metric Attributes. In *SIGMOD Conference*. ACM Press, New York, NY, USA, 239–250.

[8] Noel Cressie. 2015. *Statistics for spatial data.* John Wiley & Sons.

[9] V. de Silva and J. B. Tenenbaum. 2004. *Sparse multidimensional scaling using landmark points.* Technical Report. Stanford University.

[10] Jochen Van den Bercken, Björn Blohsfeld, Jens-Peter Dittrich, Jürgen Krämer, Tobias Schäfer, Martin Schneider, and Bernhard Seeger. 2001. XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In *VLDB*. Morgan Kaufmann, San Francisco, CA, USA, 39–48.

[11] Todd Eavis and Alex Lopez. 2007. Rk-hist: An R-tree Based Histogram for Multi-dimensional Selectivity Estimation. In *CIKM*. ACM, NY, USA, 475–484.

[12] V. A. Epanechnikov. 1969. Non-Parametric Estimation of a Multivariate Probability Density. *Theory of Probability & Its Applications* 14, 1 (1969), 153–158.

[13] Thomas Fober, Marco Mernberger, Gerhard Klebe, and Eyke Hüllermeier. 2010. Efficient Similarity Retrieval of Protein Binding Sites based on Histogram Comparison. In *GCB (LNI)*, Vol. 173. GI, Bonn, Germany, 51–59.

[14] Jinyang Gao, H. V. Jagadish, Beng Chin Ooi, and Sheng Wang. 2015. Selective Hashing: Closing the Gap between Radius Search and k-NN Search. In *KDD*. ACM, New York, NY, USA, 349–358.

[15] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proc. of the 25th Int. Conf. on Very Large Data Bases (VLDB '99)*. Morgan Kaufmann Publishers Inc., SF, CA, USA, 518–529.

[16] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. 2000. Approximating Multi-Dimensional Aggregate Range Queries over Real Attributes. In *SIGMOD Conference*. ACM, New York, NY, USA, 463–474.

[17] Max Heimel, Martin Kiefer, and Volker Markl. 2015. Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. In *Proc. of the 2015 ACM SIGMOD*. ACM, NY, USA, 1477–1492.

[18] Yannis E. Ioannidis. 2003. The History of Histograms (abridged). In *VLDB*. Morgan Kaufmann, San Francisco, CA, USA, 19–30.

[19] J. B. Kruskal. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29, 1 (1964), 1–27.

[20] Joseph B Kruskal. 1964. Nonmetric multidimensional scaling: a numerical method. *Psychometrika* 29, 2 (1964), 115–129.

[21] Hongrae Lee, Raymond T Ng, and Kyuseok Shim. 2011. Similarity join size estimation using locality sensitive hashing. *Proceedings of the VLDB Endowment* 4, 6 (2011), 338–349.

[22] V. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady* 10, 8 (1966), 707–710.

[23] D. O. Loftsgaarden and C. P. Quesenberry. 1965. A Nonparametric Estimate of a Multivariate Density Function. *Ann. Math. Statist.* 36, 3 (06 1965), 1049–1051.

[24] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. 1998. Wavelet-based Histograms for Selectivity Estimation. *SIGMOD Rec.* 27, 2 (June 1998), 448–459.

[25] S. Muthukrishnan, Viswanath Poosala, and Torsten Suel. 1999. On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In *ICDT*, Vol. 1540. Springer, London, UK, 236–256.

[26] Emanuel Parzen. 1962. On Estimation of a Probability Density Function and Mode. *Ann. Math. Statist.* 33, 3 (09 1962), 1065–1076.

[27] H. Sagan. 1994. *Space-filling curves.* Springer.

[28] Hanan Samet. 2006. *Foundations of multidimensional and metric data structures.* Morgan Kaufmann.

[29] Stefan Schmitt, Daniel Kuhn, and Gerhard Klebe. 2002. A new method to detect related function among proteins independent of sequence and fold homology. *Journal of molecular biology* 323, 2 (2002), 387–406.

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

[31] George R. Terrell and David W. Scott. 1992. Variable Kernel Density Estimation. *The Annals of Statistics* 20, 3 (1992), 1236–1265.

[32] Jeffrey Scott Vitter. 1985. Random Sampling with a Reservoir. *ACM Trans. Math. Softw.* 11, 1 (1985), 37–57.

[33] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. 2013. Quality and Efficiency for Kernel Density Estimates in Large Data. In *Proceedings of the 2013 ACM SIGMOD (SIGMOD '13)*. ACM, New York, NY, USA, 433–444.